PowerNap: An Energy Efficient MAC Layer for Random Routing in Wireless Sensor Networks

Onur Soysal
Google, Inc.
Mountainview, CA
onursoysal@google.com

Sami Ayyorgun
Telcordia Technologies, Inc.
1 Telcordia Dr., Piscataway, NJ
sami@research.telcordia.com

Murat Demirbas
Dept. of Computer Science & Engineering
University at Buffalo, SUNY, NY
demirbas@buffalo.edu

***Abstract*—Idle-listening is the biggest challenge for energy-efficiency and longevity of multihop wireless sensor network (WSN) deployments. While existing coordinated sleep/wakeup scheduling protocols eliminate idle-listening for simple traffic patterns, they are unsuitable to handle the complex traffic patterns of the random routing protocols. We present a novel coordinated sleep/wakeup protocol POWERNAP , which avoids the overhead of distributing complex, large sleep/wakeup scheduling information to the nodes. POWERNAP piggybacks onto the relayed data packets *the seed of the pseudo-random generator* that encodes the scheduling information, and enables any recipient/snooper to calculate its sleep/wakeup schedule from this seed. In essence, POWERNAP trades off doing extra computation in order to avoid expensive control packet transmissions. We show through simulations and real implementation on TelosB motes that POWERNAP eliminates the idle-listening problem efficiently and achieves self-stabilizing, low-latency, and low-cost relaying of data packets for random routing protocols.**

I. INTRODUCTION

Energy efficiency is a major challenge in wireless sensor network (WSN) deployments [1]–[3]. In multihop WSN deployments, the radio is often the critical component that drains most of the energy. Even while "idly-listening" to the channel in order to detect any potential transmissions addressed to itself, the radio wastes the same level of energy as transmitting a packet. In order to achieve longevity of WSNs, it is crucial to shut down the radios and power them back only when they are "needed". Since when the radio will be needed for receiving a packet is hard to foresee, this simple requirement spawns several challenges for the MAC layer. Low power listening (LPL)-based MAC protocols (such as BMAC [4] and AMAC [5], [6]) employ the concept of micro-sleep periods to catch packets destined to the radio while alleviating idle-listening. Coordinated sleep/wakeup scheduling protocols (such as TDMA [7], [8] and SMAC [9]) employ a rendezvous-based solution to wake the radio just-in-time for a reception and eliminate idle-listening completely. Rendezvous-based solutions also eliminate the latencies introduced in LPL-based solution and achieve low-latency relaying. On the flip side, however, rendezvous-based solutions require predictable and simple traffic patterns to be applicable.

[0]This material is based upon work supported by the National Science Foundation under Grant No. 0916504.

Next to idle-listening, an equally critical problem for the longevity of WSNs is the unproportional use of some nodes for relaying, which depletes the energy at these frequently used nodes and leads to partitioning of the network. Recently, to address load balancing of routing, random routing methods such as [10], [11] have been proposed. These methods have the advantage of simplicity. They achieve load balancing and allow multiple paths to be utilized by essentially addressing the packet to a random neighbor instead of a specific neighbor everytime. The resulting complex traffic pattern, however, renders the existing rendezvous-based MAC solutions inapplicable.

To address this gap, we present a rendezvous-based MAC protocol for random routing.

A. Our contributions

We introduce POWERNAP : an energy-efficient and low-latency MAC protocol for random routing in WSNs. POWERNAP targets the uncertainties in random routing protocols, specifically in BSTeR [10]. POWERNAP eliminates idle-listening efficiently even for complex packet arrival patterns, and hence provides significant improvements in energy-efficiency, reliability, and latency simultaneously.

The big insight in POWERNAP is that "random is not truly random". We exploit the predictability/repeatability of pseudo-random sequences for enabling the nodes to determine exactly when they are expected to wakeup for the relaying duties. Since in the implementation a pseudo-random number generation is used, all nodes can re-generate the exact same sequence of numbers if they seed the random-number generator with the same seed. Thus, by learning the current relay node and the current seed, any node can trace which next nodes will be chosen as relays for arbitrary depths and determine its wakeup time for serving as a relay. After this point, the node will re-scynchronize continuously as part of its normal-mode packet relaying, without any additional cost.

In essence, POWERNAP trades off doing extra computation to derive the pseudo-random scheduling information so that it can avoid costly control packet transmissions. Instead of sending the very lengthy sleep/wakeup scheduling information inside control packets, we just include the pseudo-random seed in the packet, and let the snooper or the recipient derive the sleep/wakeup schedule from this seed. This is a very good

tradeoff to make because transmitting a single byte through the cc2420 radio consumes energy equivalent of approximately 10,000 instructions on the msp430 microcontroller.

We designed POWERNAP to be self-stabilizing [12]. The self-stabilization property of POWERNAP stems from the fact that when a node gets out of synchrony for any reason, it can snoop any packet (even ones addressed to other nodes) and snap back to its schedule immediately. Since self-stabilization ensures that, in-spite of unanticipated transient faults, the system will recover to legitimate states, it is highly desirable for distributed algorithms. Another advantage of self-stabilizing systems is that, they do not need any initialization: they are self-configuring by design.

We implemented POWERNAP on the TelosB motes. Through a series of experiments with an 11 mote deployment, we investigated delay, throughput, and energy efficiency of POWERNAP and compared POWERNAP with LPL and always-awake (insomniac) MAC protocols. Our experiments concluded that POWERNAP matched the insomniac MAC protocol in terms of low-latency and end-to-end bandwidth results, i.e., POWERNAP performed very close to the optimal for low-latency and high bandwidth. Furthermore, POWERNAP outperformed both LPL and insomniac in terms of energy-efficiency by several folds. Finally, using Prowler simulations [13] we also experimented with multi-source extensions, scalability, and reliability issues extensively.

Outline of the rest of the paper. We present related work next, and the POWERNAP protocol in Section III. We describe self-stabilization of POWERNAP in Section IV, and the multi-source extension to POWERNAP in Section V. We present our experimental results in Section VI, and our simulation results in Section VII.

II. RELATED WORK

MAC layer protocols. LPL-based MAC layer implementations such as [4]–[6] are oblivious to the routing layer, and impose few restrictions on the traffic. However, in these methods, there is a trade off between latency and energy efficiency. The transmitter needs to use longer preambles or wait for the beacon from the receiver in order to transmit. Increasing the duration of preamble or the beacon interval increases the energy efficiency at the cost of latency.

Coordinated sleep/wakeup MAC layer implementations such as [7]–[9] synchronize transmissions to slots and only allow nodes to transmit at these time intervals. With this approach nodes can improve their energy efficiency using the information about transmission schedules of their neighbors. This, of course, requires the traffic pattern to be simple and predictable. Otherwise, due to the significant overhead of frequent distribution of large scheduling information among the nodes, this approach becomes inapplicable.

Routing layer protocols. Recent studies [10], [11], [14] show that load-balanced routing techniques improve on both energy-efficiency and delivery rates. In the collection tree protocol [14] and Arbutus [15], load balancing over multiple paths is achieved by close monitoring of the network resources with reactive adjustments to routing decisions based on link quality and available energy. Alternatively, random routing protocols [10], [11], [16] have been proposed recently to achieve load balancing with less overhead. The performance of these simple routing protocols, with regards to latency, throughput, energy efficiency, are closely related with the MAC layer implementations. However, this problem has not been addressed in the previous literature.

The idea of using predictability of random number generator was briefly noted in [17] to minimize the communication required for decisions. The authors propose sharing random number generator state for each node in 2 hop neighborhood to improve leader election process. Our study greatly extends upon this idea to share the random number generator state to whole network without incurring significant costs in communication. Our work addresses design, implementation, and deployment challenges of this idea in a multihop WSN.

III. POWERNAP PROTOCOL

A. Assumptions and Model

We investigate WSNs where multiple data generation sources are relaying data to a single sink over multihop routes. The network topology is connected but otherwise can be an arbitrary graph. For brevity we first concentrate on the single-source single sink routing in a multihop network. (In Section V we present the generalization to the multiple source multiple packet protocol.) Source node generates packets with a predefined and computable pattern. For example, the source node may be generating a new packet with a period of T minutes, with respect to a function of time, or in the most general case with a pre-announced quasi-random scheme (with known probability mass functions and seed).

Each node has a set of next nodes to be used as relays. When a node needs to forward a packet, it samples a predetermined probability distribution to decide on one of the next nodes. Although more complicated schemes are plausible, we concentrate on a simple representative scenario with simple relay probabilities. Each node keeps a table containing the probabilities of relaying through each of its neighbors. This probability distribution is sampled at each packet relay to decide on the target of message.

It is important to note that even this simple scheme can lead to significant improvements in load balancing when the relay probabilities are carefully chosen. The challenge with this relay model is the complexity of the resulting traffic. The traffic pattern has two important challenges: Large probability of overhearing packets and highly varying duration between packet arrivals.

POWERNAP protocol aims to accurately predict packet arrivals to prevent overhearing and to reduce idle listening duration. This requires successful prediction of relay duration (time required for receiving and forwarding a single packet), routes and packet generation. In this section we underline our assumptions and models we propose for predicting each component. We then combine these predictions to predict packet arrivals.

Relay duration can be estimated quite precisely as the uncertainties in relay duration is mainly caused by two factors: uncertainty in time measurements and uncertainties introduced by OS. Time measurements depend on the clocks on sensor motes. The clocks on these systems, can have offset and skew as mentioned in [18]. OS uncertainties are caused by non-predictable delays in packet routing. These include execution of sensor interrupts and application code. We show how we determine the characteristics of this duration experimentally in Section VI.

If the packet generation is known, prediction of routes are trivial in deterministic routing schemes. However when the routes themselves are truly random, exact prediction of paths is impossible. At this point we make a key observation about the random relaying schemes such as BSTeR. These protocols in essence do not require true randomness. They are equally useful with pseudo-randomness provided by random number generators. The advantage of this distinction is that, pseudo-random sequences can be re-calculated, given the initial conditions and the algorithm of random number generator is known.

We use the linear shift random number generator in TinyOS. We have the same code in each mote so the algorithm is already known. In Section III-B, we discuss how to disseminate and maintain a shared random number generator state.

A general expression for packet generation pattern is quite difficult as it depends on the amount of data generated by the application. Furthermore, the packet generation pattern can be influenced by external events. A predictor that can take into account all these factors would be quite beneficial. We leave development of such predictor for future studies and concentrate on simple data generation patterns for this study. Following BSTeR assumptions, the source motes make sure only a single packet is on route to sink any given time. We later relax this assumption to provide a more powerful protocol.

We combine our knowledge about packet generation with route computations and real delay prediction to estimate packet arrival times. Note that only real uncertainty is introduced by the relay duration as it is non-deterministic by nature. The random-like behavior in packet routes can be exactly predicted by calculation as we discuss next.

B. Protocol description

State maintained at each node. We start our discussion of the POWERNAP protocol with its state. Aligned with our self-stabilization approach, we keep the state of protocol small and repairable. If state of any mote is corrupted for some reason, for instance a packet loss or a wrong prediction of packet arrival, the mote resets its state and waits for overhearing any packet. As we describe below, a single packet reception is sufficient to properly update and correct the state.

State (S) of POWERNAP protocol consists of two main parts: static state(S_s), and dynamic state(S_d). Dynamic state is updated online while routing, whereas the static state is precomputed and does not change afterwards.

We assume static state is not corruptible (stored in flash memory). Static state consists of the topology and routing information. This is stored as a table containing a row for each node and a separate column for each neighbor node. The columns represent the probability-mass-functions (PMFs) calculated by the random routing protocol, for example BSTeR [10].

Dynamic state is employed for determining the sleep durations and it consists of the random number generator state (S_{RNG}) and the current packet location (X_{packet}). Dynamic state is updated with each packet reception. Each packet contains the random number generator state and the node to receive the packet. This extra data allows a mote receiving/snooping a packet to reconstruct dynamic state.

Calculation of next packet arrival time. Here we describe the algorithm for calculation of next packet arrival time. We use experimentally determined constant T_{relay} for expected duration of a single packet relay. Parameter T_{packet} is used to control the amount of time between packets. We define the following simple functions for the calculations:

- $S' = \textit{nextState}(S)$ function returns the next state S' given state S.
- $i = \textit{getReceiver}(S, v)$ computes node i to receive packet, given state S and the next sample number v from the pseudo-random number sequence.
- $\textit{isInvolved}(S)$ whether or not the mote is involved in the next relay given current state S.
- $v = \textit{rand}(S)$ returns the pseudo-random number v from the sequence given state S and as a side-effect updates the state S.

We present the state prediction algorithm in Algorithm 1. This algorithm computes the next relay/packet generation event and is essential for computing the required sleep duration. It uses the random number generator state and packet location to compute next location and the updated random number generator state.

Through successive applications of state update we predict next packet arrival time, which is in turn used for determining the sleep duration. This procedure is outlined in Algorithm 2. The algorithm approximates the time through simulating the relays of packets using Algorithm 1. Node only needs to be awake when a packet arrives and when it needs to relay the packet.

Using these support functions POWERNAP protocol computes the sleep duration, and turns off the radio until the expected arrival time of the next packet. A robust implementation needs to work with present uncertainties in prediction. We introduce a *grace* duration (T_{grace}) around the expected packet arrival time to deal with these uncertainties. In essence, the radio is turned on T_{grace} earlier than expected packet arrival. This change allows dealing with noise in the relay duration.

When a node fails to receive a packet while expecting one, it assumes it missed the packet. Node then waits until snooping a packet which will allow it to synchronize its sleep wakeup schedule.

Algorithm 1 NEXTSTATE(S)

Require: A valid state S
Ensure: The value of state S' after one packet relay

1: $S' \leftarrow S$
2: **if** $S.X_{packet} ==$ SINK **then**
3: $S'.X_{packet} \leftarrow$ SOURCE
4: **else**
5: $v \leftarrow$ *rand*(S')
6: $S'.X_{packet} \leftarrow$ *getReceiver*(S', v)
7: **end if**
8: **return** S'

Algorithm 2 SLEEPDURATION(S)

Require: A valid state S
Ensure: The value of desired sleep duration after reception of last packet

1: $T \leftarrow 0$
2: **while not** *isInvolved*(S) **do**
3: $S \leftarrow$ *nextState*(S)
4: **if** $S.X_{packet} ==$ SOURCE **then**
5: $T \leftarrow duration + T_{packet}$
6: **else**
7: $T \leftarrow duration + T_{relay}$
8: **end if**
9: **end while**
10: **return** D

IV. SELF-STABILIZATION

Our approach for dealing with reliability problems in POWERNAP is to use self-stabilization: we recover the system within bounded steps from arbitrary states to states from where it exhibits desired behavior. Arbitrary state corruption provides a clean abstraction of how several distributed systems are perturbed by their diverse environments. As such, self-stabilization provides a uniform mechanism to deal with unanticipated faults and avoids a case-by-case handling of faults, which is itself a significant source of algorithm design problems.

In order to prove self-stabilization of POWERNAP we make the following assumptions. We assume that all nodes eventually participate in the protocol, that is, the maximum sleep duration is bounded. We assume that faults are transient, which include arbitrary state corruption, node crash&recovery, transient link failures, and loss of synchronization[1] among nodes. (Later in this section we discuss how POWERNAP deals with permanent node crashes.) Finally we assume that the state of the source node cannot be corrupted. This assumption can be realized by using the flash memory at the source node to store checkpoints for the node's state.

[1]Note that we refer to both node state and the timing between nodes with the term *synchronization*

The standard approach to proving self-stabilization is to exhibit an "invariant" condition such that if the system is properly initialized then the invariant is always satisfied and if the system is placed in an arbitrary state then continued execution of the system eventually reaches a state from where the invariant is always satisfied. For the eventual recovery proof, it suffices to identify a variant function f on the system that is bounded from below and is monotonically decreasing outside the invariant states. Showing such a variant function asserts convergence to the invariant states.

We use $C(t)$ to denote the state of the network at time t, and define it as follows:

$$C(t) = (C_1(t), C_2(t), \ldots, C_n(t))$$

$$C_i(t) = \begin{cases} 1 & \text{mote } i \text{ can compute its sleep duration properly} \\ 0 & \text{mote } i \text{ can not compute its sleep duration properly} \end{cases}$$

The "invariant state" of the network is when all motes can compute their sleep duration properly. In the invariant state all packets are properly forwarded and motes sleep for the maximum possible durations.

Theorem 1. POWERNAP *is self-stabilizing.*

Proof: We first observe that the network state changes only when a packet is sent from node a to node b. We treat packet losses as mis-predictions in the receiver node. Note that when a node mis-predicts the packet arrival, any node scheduled to receive from this node will also be mis-predicting as well since the packet will not be sent at the appropriate time. Hence we formulate updating of the state as follows:

$$C_i(t+1) = \begin{cases} C_a(t) & i = b \text{ and packet expected from mote } a \text{ to mote } b \\ C_i(t) & \text{otherwise} \end{cases}$$

We now define the variant function $f(t)$. This function corresponds to the distance of the network state from the invariant state:

$$f(t) = n - \sum_{i=1}^{n} \prod_{j=1}^{i} C_j(t)$$

Note that $f(t)$ is bounded below at 0, which corresponds to the invariant state. Next we show that $f(t)$ is non-increasing and is eventually decreasing under the algorithm actions. To this end, we investigate $f(t+1)$.

$$f(t+1) = n - \sum_{i=1}^{n} \prod_{j=1}^{i} C_j(t+1)$$

After expanding $f(t+1)$ and simplifying (the details of which are relegated to the technical report version of this paper [19]) we show that $f(t+1) \leq f(t)$, and conclude the self-stabilization of POWERNAP . ■

Using this result we give an upper bound on the error recovery. When the network has n nodes and the maximum time between packet arrivals to a node is T_s, the total time required for recovery from error in the network is always less than or equal to $2(n-1)T_s$. This property follows from the fact that within $2T_s$ time the closest node to the source that cannot compute its sleep time will be receiving a packet. Even in the worst case of the node sleeping for T_s maximum sleep duration unnecessarily, during the second T_s time it will be guaranteed to receive a packet. This process will be repeated for all the nodes in the network other than the source so finally all nodes will be able to predict their state properly.

Extensions to snooping. Snooped packets can be utilized to improve stabilization performance. When only packets from upstream nodes are snooped, these snooped packets can be used for state update without any complications. When both upstream and downstream messages are snooped, there is the hypothetical danger of unsynchronized state spreading to upstream nodes. However, such a scenario is unlikely when the number of out-of-synch nodes is low. Since most of the packets would be sent by the synchronized nodes, most of the snooped packets help to synchronize the node.

For better synchronization performance, we used snooping without any restrictions in our experiments. During our experiments in Section VI and VII, we have not observed any synchronization problems due to this promiscuous snooping scheme.

Effects of node deaths. Node deaths also introduce a complication for self-stabilization. Normally, the nodes that are affected by a dead node and drop out-of-synchrony are able to snoop a packet from another path and synchronize. However, even after self-stabilization, the dead node will continue to inject faults to the system: every time a packet is scheduled to be relayed by the dead node the nodes downstream the path will fall out of synchrony inevitably. Assuming that POWERNAP can stabilize from these faults faster than the faults are re-introduced, these faults can still be treated as virtually transient, and our stabilization proof holds.

The advantage of POWERNAP is that a node death does not make any of the downstream nodes obsolete. Due to our path diversity and self-stabilization property, all nodes remain usable and carry their loads in the relaying protocol. However, dealing with dead nodes is eventually the duty of the network and routing layers. The routing service should perform periodic purging of the dead nodes so as to prevent their hits on the performance. After determining the good neighbors, the routing service can restart POWERNAP from a cold-start. Due to quick self-stabilization time of POWERNAP the cold-starts do not impose a large penalty on the system.

V. MULTIPLE SOURCE EXTENSION

The single source version of POWERNAP synchronizes all the nodes to the source node, i.e., the nodes obtain routing state information from nodes closer to the source through packet reception. When there are multiple sources, this approach is problematic since it requires explicit coordination among sources. To mitigate this problem we require nodes to update routing state information from nodes closer to the sink. This requires sink to broadcast information about the protocol state.

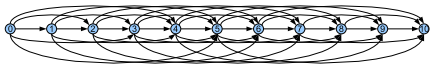

Fig. 1. Topology used in experiments. Each circle corresponds to a mote with shown id. Each arrow corresponds to a possible packet relay.

We use packet acknowledgments for this purpose through piggybacking the dynamic part of the protocol state. This way sink can disseminate the protocol state information to nodes. We also assume all nodes generate packets in multiple sources scenario. The order of the nodes are also encoded in the dynamic state, together with random number generator state and current packet location. For simplicity we use a round robin schedule between nodes. More sophisticated schedules are plausible as long as they can be precomputed. Since the sink is the main authority on dynamic state and any packet reception allows a node to compute its sleep/wake up schedule, the protocol does not need a long bootstrap process. As we show in Section VII, this process is quite fast even for relatively large topologies.

Routing decisions, apart from originating from multiple source nodes, is the same with single source POWERNAP protocol. We only route a single packet at a time and we try to maximize throughput via sending packets as fast as possible. We enforce this constraint to eliminate contention in the network. If multiple packets are allowed to be routed simultaneously in the network, the resulting collisions cause significant packet losses. We investigate this problem in simulations in Section VII.

VI. EXPERIMENTS

A. Experimental setup

We implemented POWERNAP on TelosB motes to accompany a random routing protocol. We conducted our experiments with 11 TelosB motes. The topology used in the experiments is shown in Figure 1. In this topology each node has up to 5 relaying neighbors that are closer to the sink (denoted by mote 10). For simplicity's sake, in our implementation, a node chooses to forward the packet to any of its relaying neighbors with equal probability. (We note that better PMFs can be used to further improve performance and load balancing. We relegate the computation of optimal forwarding probabilities to studies such as [10].) Although the depicted topology is linear, POWERNAP is oblivious of this property so it does not rely on any implications of 1D deployments.

We evaluate each configuration with 50 runs. Each run consists of 50 packet routings with different initial random seeds. During each experiment run, the motes maintain multiple log information. In addition to packet logs, each mote keeps track of the duration it spent in each radio state. The information recorded at the motes are reported to the basestation at the end of each run. The basestation then uses a Java application to calculate the energy use at each mote. More specifically,

the Java application combines the duration a node spent in each protocol state with the power usage information at the corresponding states to to arrive total energy use figures. (We used a 10 Ω resistor connected to the moteto measure the amount of current used at each state of the protocol.)

B. Metrics

The results of experiments are summarized using following metrics:

- *Packet Reception Rate* is the ratio of the packets received at sink to the packets generated at source.
- *End-to-end Delay* is the time between creation of packet to its delivery to the sink.
- *Throughput* is the amount of payload data delivered to sink per second.
- *Routing Efficiency* is the payload transferred from source to sink per Joule of energy spent by the network.

C. PowerNap implementation

Our implementation of POWERNAP uses hardware acknowledgments in CC2420 radios. CC2420 can send these hardware acknowledgments quite fast since no communication between MCU and radio is required. When acknowledgment is not received the packet is repeated up to a maximum of 3 retries.

One of the challenges with the hardware acknowledgments is that when radio is set to use hardware acknowledgments, hardware address resolution is also required. This prevents nodes from snooping packets in order to synchronize with the routing schedule. We resolve this issue with another timeout which switches from hardware acknowledgments to snooping after the node detects it missed its expected packet.

The memory footprint of the protocol is an important concern due to potentially large routing table. We managed to reduce the size of routing table to 24 bytes per node where each node has at most 6 nodes to forward packets to. This way POWERNAP can accommodate up to 400 nodes.

To tune our POWERNAP implementation, we first determined the time needed for relaying. Our experiments showed that $5.5ms$ for each relay is sufficient. We fixed this value in our implementation and used it in all of experiments. Next we investigated our main control parameter, routing grace(T_{grace}). Routing grace corresponds to the time window around the expected packet arrival which is used to deal with the clock differences and operating system related uncertainties. The size of this window is crucial: too small a time-window leads to packet losses and too large a time-window wastes energy.

Figure 2 summarizes the effect of routing grace on data transfer efficiency. The maximum efficiency is obtained with a routing grace period of $2ms$. Above this period the latencies do not improve as well, therefore we fix the grace period as $2ms$ for our POWERNAP implementation. The technical report version contains more details about these experiments [19].

D. Comparison results

For comparison purposes, we implement two protocols, namely *insomniac* and *LPL* protocols, to compete with POWERNAP . Both protocols use the same random routing protocol that POWERNAP works with. The PMFs were also chosen the exact same way, so the traffic pattern resulting from the routing protocol is exactly the same in all three protocols for a fair comparison.

In the insomniac protocol, the motes do not perform any sleep-wakeup. Since all nodes are awake all the time, this protocol serves as a lower-bound in terms of achievable end-to-end delay in our network, as well as determining the upper-bound for the end-to-end data rates. The downside of the insomniac protocol is its lack of energy-efficiency.

The LPL protocol is an implementation of BMAC [4] on CC2420 radios. Since CC2420 is a packet-based radio, instead of sending preamble bytes for the determined preamble duration, for our implementation it was more convenient to repeat the message back-to-back for the same duration. Since we also used hardware acknowledgments in our implementation, the packet repeating is stopped upon receiving an acknowledgment.

Through our experiments all protocols achieved higher than 95% packet reception rates for all protocols. For additional details about the experiments see the technical report version [19].

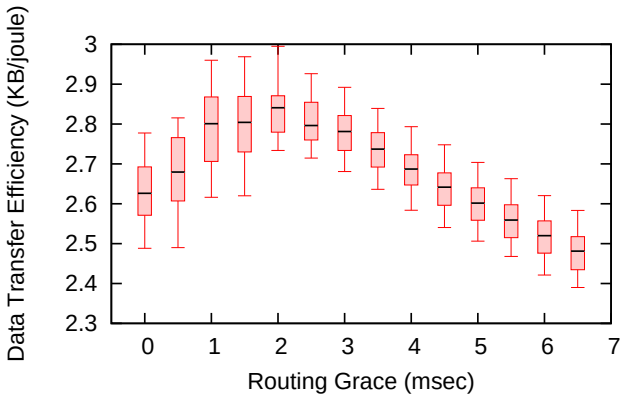

Fig. 2. Transfer efficiency vs. routing grace

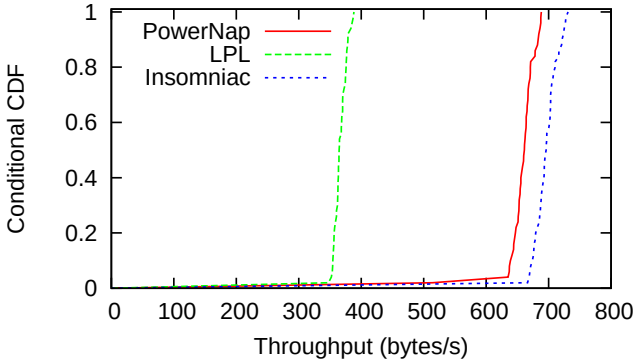

Fig. 3. Distribution of end-to-end data rate

Figure 3 shows that POWERNAP performs close to insomniac in terms of end-to-end throughput. The small difference between the two can be attributed to occasional repeated packets in POWERNAP . On the other hand, the LPL protocol

manages to deliver only half of the data rate of PowerNap . LPL requires longer delays at each hop due to microsleeps. Since we only allow a single packet to be relayed at a time, this increased delays also reduce the throughput.

End-to-end latency results support the above conclusion as the latencies of the LPL protocol is almost two times that of PowerNap in Figure 4. On the other hand, the end-to-end delay for PowerNap is comparable to insomniac protocol with the exception of a heavier tail at the corresponding to a few delayed packets due to repeats. It should be noted that our implementation of LPL is intended for higher throughput rather than energy efficiency. Other implementations of LPL (such as in [6], [20]) result in latencies in order of 20ms per message transmission whereas our implementation only requires 10ms.

Our final comparison focuses on the data transfer efficiency of protocols. This is a crucial metric as it corresponds to the lifetime of the system under same load. The data transfer efficiency difference between the LPL protocol and PowerNap is more than 400%, as demonstrated in Figure 5. PowerNap also improves over the insomniac protocol by 300%.

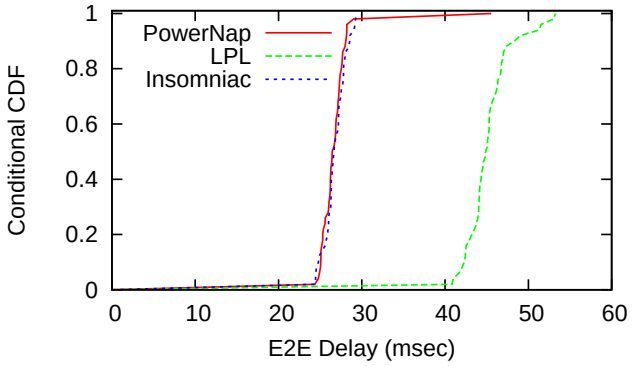

Fig. 4. Distribution of end-to-end delay

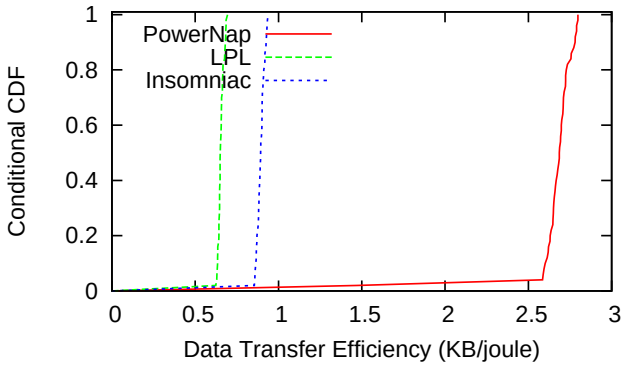

Fig. 5. Data transfer efficiency

To summarize, the experiments we conducted indicate that PowerNap protocol is efficient with respect to delay, energy use, and end-to-end bandwidth.

VII. Simulation Results

In this section we provide results on our multi-source extension to PowerNap using Prowler [13] simulations. For performance reasons we use the Java implementation JProwler which offers same radio models with improved scalability. We used the Gaussian interference model in JProwler. The interference in this simulation has a static and a dynamic component. Static component reflects multi-path effects and reflections in a link and stays constant after the topology is constructed. Dynamic component on the other hand models the transient effects in channel quality common in low-power radios. While deciding on success of reception, interference of transmissions are accumulated to a total noise strength and compared against received signal strength.

In our simulation experiments, we employ all nodes in the network as sources. These sources take turn in sending packets as we discussed in Section V. In our simulations, we use the same values for message duration, optimal grace periods, and calculation of energy consumption in line with our experiments on TelosB motes.

We choose random topologies in a square area with approximately constant node density. In order to make topology connected we enforce a lower bound on minimum allowed distance between nodes. The sink is placed at one of the corners. We run simulations for 100 seconds and repeat each simulation 100 times with different random number generator seeds.

In the beginning of each simulation run, nodes are in an unsynchronized state and they synchronize gradually by snooping the routed messages. Since PowerNap is self-stabilizing, we can afford to start the network from an unsynchronized state (which we call a “cold-start”).

We first investigate the time required for all nodes to synchronize with respect to increasing number of nodes in the network. Figure 6 shows an approximately quadratic increase of synchronization time with respect to the number of nodes(n).

This is due to two factors: the increase in network diameter and the increase in round duration. Round duration is the time between two packet generations at the same node. Round duration effects the synchronization time since it affects the spreading speed of synchronization.

Since we use constant network density, network diameter increases by $\sqrt{n}$ as number of nodes increases. Round duration, on the other hand, increases by a factor of $n\sqrt{n}$ since it depends both on number of nodes and the network diameter. These two factors result in the quadratic increase in synchronization duration.

Figure 7 shows that end-to-end delay increases proportionally to $\sqrt{n}$ as it depends only on the network diameter. As a result, we observe a corresponding drop in throughput with increased number of nodes in Figure 8. This is mainly due to our rule of allowing only a single packet to be routed at a time; as the network grows the increased number of hops per packet takes its toll on throughput.

Efficiency of packet transfers suffers more from increased number of nodes. Figure 9 indicates decrease proportional to $n\sqrt{n}$, caused by reduced data rate and increased contribution of sleeping nodes. Even though the energy cost of sleeping

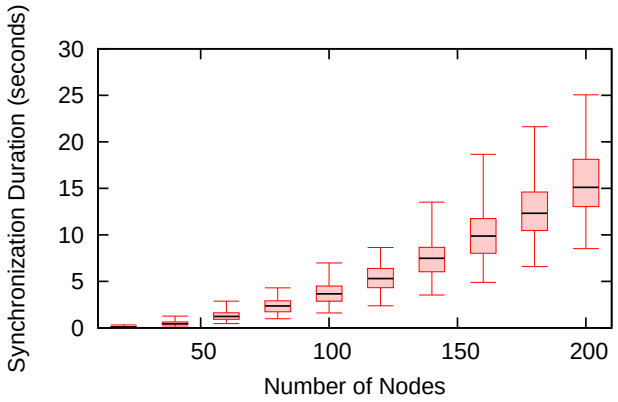

Fig. 6. Synchronization from a cold start

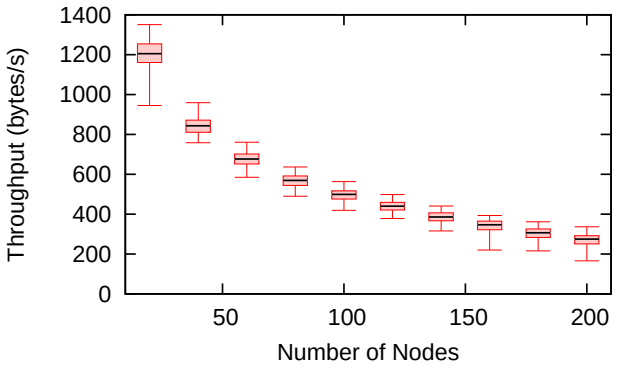

Fig. 8. Average end to end throughput with increasing number of nodes

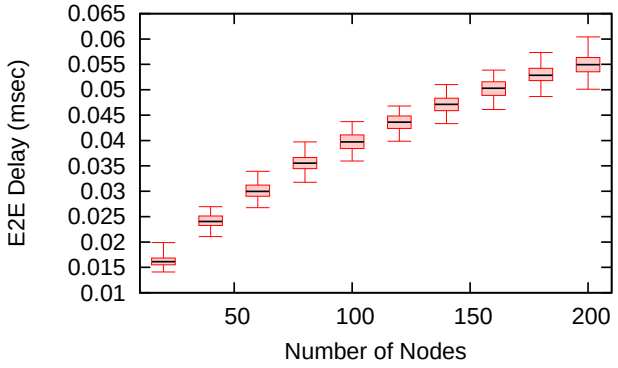

Fig. 7. Average end to end delay for packets with increasing number of nodes

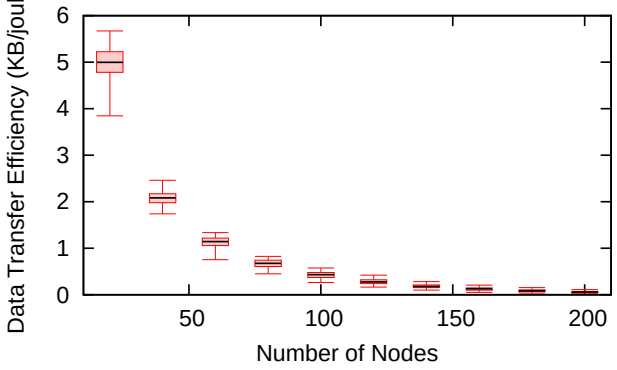

Fig. 9. Data transfer efficiency of the network with increasing number of nodes

is small compared to listening, as number of nodes increases this energy cost starts to dominate total energy cost.

Next we consider relaxing our single packet rule, and present the problems created by allowing multiple packets to be sent by multiple sources simultaneously.

Figure 10 depicts the throughput of this scenario for 100 node topologies. As expected the throughput increases with increased concurrency. However, as shown in this improvement comes with significant cost in packet losses. Moreover, the message losses effect nodes far away from the sink more detrimentally as their packets need to traverse through more hops. Effectively increased concurrency also reduces fairness of the protocol. We relegate addressing the fairness and performance issues to the routing and network layer protocols which are more capable of dealing with the coordination.

In our final set of simulations we investigate effect of node failures on packet deliveries. For these experiments, we turn off a random subset of nodes after 10 seconds of the simulation run. We then, compare POWERNAP with a deterministic convergecast in terms of packet delivery performance from each source. Figure 11 shows that increasing the ratio of failures reduces the packet delivery performance. This result is expected as more paths are disturbed as the number of failing motes increase. However, we note that the crashed nodes do not necessarily disconnect the network due to the inherent path diversity in POWERNAP . We observe that even after the crash of 10% of the nodes, approximately 80% of the nodes can still send some data to the sink, in contrast to the 60% ratio achievable by the deterministic convergecast.

VIII. CONCLUDING REMARKS

POWERNAP handles randomly-distributed arrival of packets without the overhead of distributing large sleep/wakeup scheduling information to the nodes. To this end, POWERNAP piggybacks onto the relayed data packets the seed of the pseudo-random generator that encodes the current sleep/wakeup scheduling information and enables the recipient (and any potentially unsynchronized snooper) to compute its sleep/wakeup schedule from this seed. Our coordinated sleep/wakeup protocol is readily applicable to related problems such as maintaining a sentry monitoring service [21], [22] in an energy-efficient manner.

A key concept in the design of POWERNAP was to trade off doing extra computation in order to avoid expensive control message transmissions. Since the transmission of a message (or for that matter idle-listening for the same duration) wastes several orders of magnitude more energy than doing computation for the same duration, trading off computation with communication is a viable solution for energy efficiency in WSNs.

REFERENCES

[1] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *in 7th*

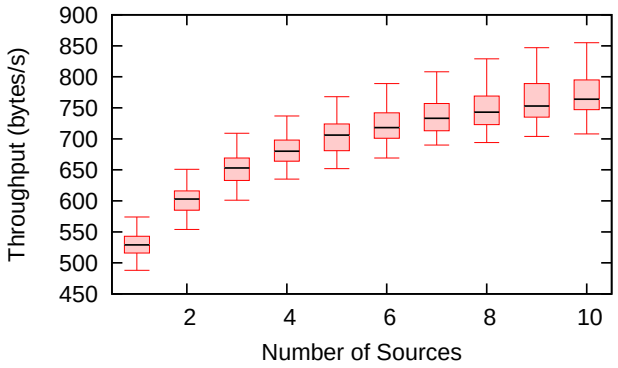

Fig. 10. Average end to end throughput with increasing number of concurrent transmissions

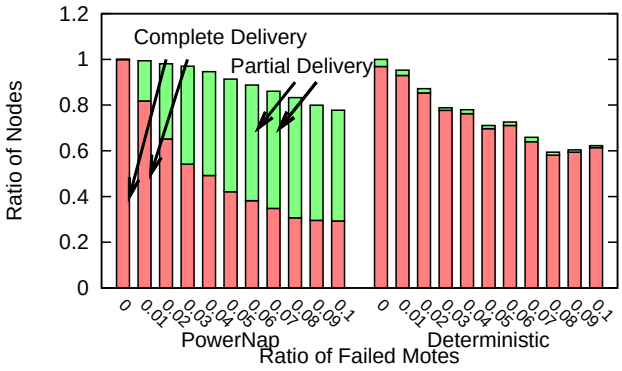

Fig. 11. Change in packet delivery performance with increasing ratio of failing motes. Dark bars correspond to the ratio of nodes whose packets are delivered without loss whereas light bars to show the ratio of the nodes which deliver some (at least 25%) but not all of their packets.

USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006), 2006.

[2] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. M. Mainwaring, and D. Estrin, "Habitat monitoring with sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 34–40, 2004.

[3] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y.-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks (Elsevier)*, vol. 46, no. 5, pp. 605–634, 2004.

[4] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 95–107.

[5] P. Dutta, S. Dawson-Haggerty, Y. Chen, C. Liang, and A. Terzis, "Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless," *Sensys*, 2010.

[6] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis, "Koala: Ultra-low power data retrieval in wireless sensor networks," in *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 421–432.

[7] S. S. Kulkarni and M. Arumugam, "Ss-tdma: A self-stabilizing mac for sensor networks," in *Sensor Network Operations, IEEE Press.*, 2005.

[8] L. van Hoesel and P. Havinga, "A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches," in *1st International Workshop on Networked Sensing Systems (INSS*. Tokio, Japan: Society of Instrument and Control Engineers (SICE), 2004, pp. 205–208. [Online]. Available: http://doc.utwente.nl/64756/

[9] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOMM*, 2002, pp. 1567–1576. [Online]. Available: citeseer.ist.psu.edu/ye01energyefficient.html

[10] S. Ayyorgun, J. Ai, and S. Shankar, "Towards a self-organizing stochastic-communications paradigm for wireless ad-hoc/sensor networks," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, 29 2008-Oct. 2 2008, pp. 36–47.

[11] G. Barnes and U. Feige, "Short random walks on graphs," *SIAM Journal on Discrete Mathematics*, vol. 9, no. 1, pp. 19–28, 1996.

[12] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, vol. 17, no. 11, 1974.

[13] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi, "Simulation-based optimization of communication protocols for large-scale wireless sensor networks," *IEEE Aerospace Conference*, pp. 255–267, March 2003.

[14] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2009, pp. 1–14.

[15] D. Puccinelli and M. Haenggi, "Arbutus: network-layer load balancing for wireless sensor networks," *IEEE Wireless Communications and Networking Conference*, pp. 2063–2068, 2008.

[16] S. D. Servetto and G. Barrenechea, "Constrained random walks on random graphs: Routing algorithms for large scale wireless sensor networks," *ACM International Workshop on Wireless sensor networks and applications*, pp. 12–21, 2002.

[17] S. Nath and P. B. Gibbons, "Communicating via fireflies: geographic routing on duty-cycled sensors," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 440–449.

[18] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 39–49.

[19] O. Soysal, S. Ayyorgun, and M. Demirbas, "Powernap for longevity in wireless sensor networks," University at Buffalo, SUNY, Tech. Rep. 2009-08, 2009.

[20] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2006, pp. 307–320.

[21] J. Hui, Z. Ren, and B. H. Krogh, "Sentry-based power management in wireless sensor networks," in *IPSN*, 2003, pp. 458–472.

[22] M. G. Gouda, Y. ri Choi, and A. Arora, "Sentries and sleepers in sensor networks," in *In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS*, 2004, pp. 384–399.