To appear, Third IEEE SECON, Reston, Virginia, USA, September, 2006

Energy Efficient Network Reconfiguration for Mostly-Off Sensor Networks

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

***Abstract*— A new class of sensor network applications are *mostly off*. Exemplified by Intel's FabApp, in these applications the network alternates between being off for hours or weeks, then activating to collect data for a few minutes. While configuration of traditional sensornet applications is occasional and so need not be optimized, these applications may spend half their time while awake configuring, so they require new approaches to *quickly* restart after a long downtime, in effect, "sensor network suspend and resume". While there are many network services that may need to be restarted, this paper focuses on the key question of when the network can determine that all nodes are now awake and ready to interact. Current resume approaches assume worst-case clock drift and so must conservatively take minutes to reconfigure after a month-long sleep. We propose two energy efficient reconfiguration protocols to address this challenge. The first approach is *low-power listening with flooding*, where the network restarts quickly by flooding a control message as soon as one node can determine the whole network is up. The second protocol uses *local update with suppression*, where nodes only notify their one-hop neighbors about the network state, avoiding the cost of flooding. Both protocols are fully distributed algorithms. Through analysis and simulations, we show that both protocols are more energy efficient than current approaches. Flooding works best in *sparse* networks with 6 neighbors or less, while local update with suppression works best in *dense* networks (more than 6 neighbors).**

I. Introduction

Sensor networks use small sensor nodes such as Berkeley Motes [7], [11] to sample the physical environment, process and transfer data to remote users. These sensors are usually battery operated, so an important research challenge is efficient management of energy usage to maximize network lifetime.

Sensor network applications vary from micro-habitat monitoring [1], [13], structural monitoring [20] to surveillance for intrusion detection. Most of these applications today assume an *always-on* network. For example, in surveillance applications, the network need to stay active all the time in order to detect any event in real time. To reduce energy consumption when there is no traffic to send, MAC protocols for sensornets (such as S-MAC [22] and B-MAC [14]) put the radio to sleep, even though they preserve the abstraction of an always-on network. To maintain this abstraction, their sleep periods are rather short, ranging from tens of milliseconds to a small number of seconds (the default sleep period in B-MAC is 100ms, and in S-MAC at 10% duty cycle, 1 second).

This research is partially supported by the National Science Foundation (NSF) under the grant NeTS-NOSS-0435517 as the SNUSE project, by a hardware donation from Intel Co., and by Chevron Co. through USC Center for Interactive Smart Oilfield Technologies (CiSoft).

Topology control is a second approach to conserving energy, and is specific to *dense* sensornets [21], [2]. With topology control, some nodes shut down for extended periods of time, but the network colludes to ensure that enough nodes remain active to guarantee coverage and full connectivity. Thus, while individual nodes may not be available, the overall abstraction of a connected network is maintained. Topology control can be even more efficient than MAC approaches since it places nodes asleep for extended periods, avoiding even minimal MAC-layer synchronization or polling costs.

Recently a third category of applications has emerged, that of *mostly-off* applications. In these applications, nodes are only active for brief periods to collect data. For the rest of the time, they are not required for any sensing tasks, and to conserve energy they *all* should turn off. Equipment monitoring for extended periods was the first example application in this category, where nodes only need to check equipment status once a day or a week [16]. A second example is seismic monitoring of underwater oil fields [6], where we expect the application to generate and collect data for dozens of minutes, but perhaps only every 30 days, or even less frequently. For these applications, network lifetime is maximized if the network as a whole shuts down completely between active periods, in effect, "sensor network suspend and resume". While between sensing, all components on a node are shut off except a real-time clock that is able to wake up the node at the next scheduled task time. We therefore consider these *mostly-off networks*.

The goal of this paper is to develop new protocols for efficient network reconfiguration after a long sleep. The main challenges are things that change over time. The most significant of these is clock drift—the fact that typical clocks will drift from true time and each other. As a result, not only must tightly synchronized operations (such as scheduled MAC protocols) recover after sleep, but the network must be careful even to ensure all nodes are active. The exact set of services that need to be reconfigured after sleep vary depending on the application and protocols in use, ranging from determining that all nodes are up, setting a MAC schedule, finding MAC-level neighbors, reestablishing forwarding paths, resetting time synchronization. This paper focuses on the first of these: the need for all nodes to determine when the entire network is up, since it is common to all networks before traffic can be sent.

Current CTOS crystal oscillators have a drift rate of 30–50 parts per million (ppm). When clock drift rate is 50ppm, then clock drift after 30 days could be as long as 130 seconds. In the above application of seismic monitoring of underwater oil fields, nodes agree on the same moment to wake up before they go to sleep for 30 days, and they set up timers to awake themselves later. But due to clock drift, it is simply not feasible for them to reboot at the exact same moment during the next active period. Nodes can wake up any time during the drift period of 260 seconds (on either two directions for possible clock drift).

The central problem here is that nodes must coordinate after waking up. First, senders waking up earlier must wait and delay data transmission until the whole network *resumes* and all other nodes are active and able to receive packets. This delay, *drift delay*, is necessary to guarantee network connectivity before any data transmission. Our goal is to minimize the energy spent during this time. Again, for this application, nodes may only sense and exchange data for 4–10 minutes. In this case, energy spent in drift delay can be as much as half the total energy consumed during the networks entire active life.

Second, nodes must *know* that all the network is up. Once a node is up, it must wait a further time to insure that all other nodes are up (and therefore able to forward the data) before it sends a message. We define this time as *data message delay*, and it effectively doubles the delay after wakeup before nodes can assert all nodes are up (and therefore reconfiguration is done).

The challenge in mostly-off networks is therefore to minimize the energy wasted during drift delay and data message delay. As far as we know, currently there are no network re-configuration protocols specifically designed to reduce this cost. The problem was identified in Intel's FabApp [16], but there nodes simply perform wait, low-power listening (LPL) [3], [14] until all nodes rejoin the network. We will show new approaches can consume 50% less energy than average-case LPL energy, and 66% less than worst case LPL.

We propose two new protocols to efficiently manage energy usage during the reconfiguration period. Our protocols are designed for highly resource-constrained sensor nodes, such as 8-bit motes, and to support small to very large networks (from tens to hundreds or thousands of nodes). They therefore emphasize simplicity and fully distributed operation. Our first protocol is *low-power listening with flooding*, where the network restarts quickly by flooding a control message as soon as one node can determine the network is up. The second protocol uses *local update with suppression*, where nodes only notify their one-hop neighbors about the network state, avoiding the cost of flooding. Both protocols accomplish the goal of letting all nodes know that the network is up. In addition, the flooding approach can also propagate schedule information used by a scheduled MAC protocol (such as S-MAC [22], T-MAC [19], or SCP-MAC [23]). Analysis and simulation results show that both protocols are more energy efficient than current approaches. Flooding works best in *sparse* networks with 6 neighbors or less, while local update with suppression works best in *dense* networks (more than 6 neighbors).

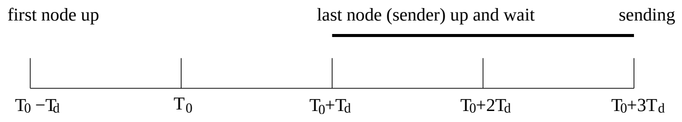

Fig. 1. Worst case $4T_d$ transmission delay in FabApp

The key contribution of this paper is the defintion of these two new protocols and their evaluation and comparision to prior work through analysis and simulation. Important findings are that relatively simple protocols can improve efficiency and overall energy cost, and an understanding of how performance changes as a function of network density.

II. Related Work

A. Reconfiguration in FabApp

In mostly-off networks, when nodes come back from sleeps at the expected wake up time, they wake up asynchronously due to clock drift. Let's assume the wakeup time by ideal clock is T_0 and the maximum clock drift during long sleep period is T_d. Since a clock can drift either faster or slower than the ideal clock, the earliest time that a node wakes up is $T_0 - T_d$, and the latest time is $T_0 + T_d$, as shown in Figure 1. Therefore, the maximum drift delay is $2T_d$.

FabApp tolerates clock drift by requiring that all nodes wait for the maximum drift time before beginning communication [16]. After a node wakes up, it waits for $2T_d$ to make sure that all other nodes are up. To minimize energy consumption during this waiting period, FabApp uses B-MAC, an energy-conserving MAC protocol that samples the channel activity periodically rather than continuously listening.

We define the delay until data communication can begin as the time from when the last node wakes up until the first data message can be sent. In FabApp, it depends on when the first data sender wakes up. As an example, Figure 1 shows the worst case, where the sender wakes up at $T_0 + T_d$. Since it delays its transmission for $2T_d$, nodes who wake up at $T_0 - T_d$ have to keep waiting for a duration of $4T_d$.

B. MAC Protocols

Recent contention based sensor network MAC protocols adopt sleep/wakeup cycles to allow nodes to operate at low duty cycle mode to save energy. Two primary techniques have been considered in MAC layer designs. S-MAC, T-MAC and TRAMA [15] are based on *listening schedules*. Nodes wake up for a brief contention period to coordinate and send data at their neighbors' scheduled wakeup time. S-MAC and T-MAC also attempt to synchronize on same cycles to maximize energy savings. The other technique is *low-power listening* adopted by B-MAC and WiseMAC [3], [4]. In this approach, receivers periodically sample channel activity by taking one or a few signal strength samples. To wake up receivers, sending nodes

	Reconfiguration service
1	Determine when the entire network is up
2	Set up MAC schedules
3	Discover neighbors
4	Set up data forwarding paths
5	Re-establish time synchronization

TABLE I

TYPICAL RECONFIGURATION SERVICES AFTER A LONG DURATION OF SLEEP.

include relatively longer preambles before each packet. SCP-MAC [23] combines the concepts of *low-power listening* and *synchronized schedules* to reduce the cost of long preambles.

Recently Li et al. showed that multiple schedules are common in real networks with for *schedule* MAC protocols [12]. This work also showed how to migrate all schedules in a network to a single common schedule, reducing the cost of multiple schedules. *Schedule based* MAC layer protocols can potentially utilize this protocol, *low-power listening with flooding*, to exchange schedule synchronization information during flooding. There is slight chance that multiple schedules still happen after flooding. These different schedules can be further converged with *global schedule algorithm* (GSA) [12] after reconfiguration.

III. DESIGN OF ALGORITHMS

This section describes the designs of several reconfiguration algorithms we propose. The central idea behind all of our approaches is to determine when all of the nodes in the network know for certain that others are up, so that they can begin general communication. Our algorithms aim to minimize the energy consumption during reconfiguration and quickly bring the network up. We will evaluate the energy performance of each protocol in next section.

Table I listed typical reconfiguration services after a long duration of sleep. The major focus of our algorithms is to quickly finish service 1, so that general communication can start. However we also evaluate each algorithm, and discuss whether it can be leveraged for services 2 and 3. We do not consider services 4 and 5 in this paper.

A. Simple Low Power Listening

The simplest way to ensure that all nodes in the network are up before communication is to wait longer than the possible clock drift time. This is the protocol used in the FabApp (II-A). Our first new protocol is a very simple optimization on that protocol: we short-circuit this waiting when the sender wakes up. Recall that without any coordination, each individual node must wait for $2T_d$ to ensure all other nodes are up.

We define the *Simple Low power Listening* (SLPL) protocol as each node waiting and listening for $2T_d$, and any node hearing data can short-circuit waiting and immediately consider the network configured. This optimization is possible because the $4T_d$ delay occurs due to the worst-case wait time for the whole network, but $2T_d$ is actually sufficient for worst case wait time for any single node.

In SLPL, without hearing any messages from others, the first sender has to wait for $2T_d$ before transmission to ensure that all nodes are active. SLPL works best when the first sender becomes active at the earliest time (T_d before T_0), because other nodes can stop waiting right after they receive the first data message. But if the first active node does not send any message after it waits for $2T_d$, other nodes have to wait until their own timers fire. This explains SLPL spends longer time on reconfiguration than necessary. The worst case of SLPL requires up to $4T_d$ waiting time.

Although SLPL can ensure that the network is up (Service level 1 in Table I), it has two limitations. First, it does not provide enough information to set up MAC schedules if using a scheduled MAC (reaching service level 2). If running a scheduled MAC protocol, additional configuration will therefore be necessary. We will show below that our protocol Low Power Listening with Flooding can also provide schedule information. Second, the channel polling period during reconfiguration must be the same as that in normal data communication, so that nodes can receive possible data messages during reconfiguration. A potential opportunity to save additional energy would be to run with a different (less frequent) polling interval during reconfiguration, then switch to more frequent polling for regular operation.

B. Low Power Listening with Flooding

As illustrated above, we could further cut back on network reconfiguration time to achieve more energy conservation. What we propose is to let the earliest active node send out an explicit control message to inform other nodes that the network is up and reconfiguration can be terminated immediately. This approach, can save more energy compared to SLPL because it can significantly shorten the reconfiguration time.

In *LPL with flooding* each node sets up a timer to wait for $2T_d$ after it reboots. Nodes still run low-power listening while waiting. The first active node sends out a network *up* message immediately when its timer fires. At this moment, all nodes should have become active, since $2T_d$ is the maximum clock drift period. The up message is further flooded throughout the whole network. Nodes can safely stop their timers when receiving an up message. Compared to SLPL, LPL with flooding can significantly reduce the reconfiguration time. It takes at most $2T_d$ plus the message flooding delay.

There are several advantages to exchange explicit control messages during reconfiguration. First, the reconfiguration phase and the data communication phase are separated. After reconfiguration, the application can choose to run any types of MAC protocols, including those that do not use LPL. Second, if the application chooses a MAC based on LPL, the channel polling interval can be independently optimized for both the reconfiguration phase and the data communication phase. Section IV-C describes how optimal parameters can be selected to minimize the energy consumption during reconfiguration. In contrast, in SLPL, nodes must operate on the same polling interval during these two phases, because nodes expect to receive data messages during the reconfiguration. Finally, leveraging the control message exchange, LPL with flooding can accomplish more reconfiguration services as listed in Table I. If the application runs a *scheduled* MAC protocols, such as S-MAC, T-MAC or SCP-MAC, nodes can exchange schedule information with flooding of up messages. This effectively finishes the reconfiguration service 2. Moreover, during the flooding process, nodes are actually able to discover all their neighbors, so the reconfiguration service 3 can be accomplished as well.

The major downside of this algorithm is the cost of flooding. The cost increases as the node density increases, since there will be more overhearing of the redundant up messages. To reduce overhearing, We further propose an optimization during the flooding. When the first node sends out its up message, it puts its channel polling time in the packet. When its neighbors receive the message, they will follow the same polling time described in the message. Essentially all nodes who have received an up message will synchronize their polling times. When they re-broadcast the up message, they intentionally start the transmission when these synchronized nodes have just finished polling. Since all up messages uses long preambles, these nodes will avoid overhearing the long preambles. The *synchronized LPL* scheme can significantly reduce the overhearing cost during flooding.

In summary, LPL with flooding can quickly complete reconfiguration after $2T_d$ since the first node reboots. It significantly reduces the data message delay, since no matter when the first sender wakes up, it can start data transmissions immediately after the flooding. Compared to SLPL, it can significantly reduce energy cost at low to moderate neighborhood sizes.

C. Local Update with Suppression

As stated in the previous section, LPL with flooding can significantly speed up reconfiguration. However such benefit comes at the cost of overhearing redundant up messages during the flooding process. The cost will become significant when the network density is high. In such networks, it is expensive to explicitly synchronize the network up time in the whole networks. To alleviate this problem, we propose the *local update with suppression* protocol, in which we avoid global synchronization by limiting the coordination to one-hop only.

Similar to *LPL with flooding*, in this new protocol, each node sets up a network resume timer of $2T_d$, and runs LPL after reboots. When the timer fires, a node broadcasts the network up message once to its immediate neighbors. As we described above, after a node waits for $2T_d$, it knows for sure that the entire network is up. When the one-hop neighbors receive the up message, they learn that the network is up, and thus cancel their own timers. This single up message effectively *suppresses* all other nodes in the one-hop neighborhood from sending their own up messages later. As these nodes have finished reconfiguration, they are ready to start data transmissions if any. Nodes who hear the data messages can also immediately learn that the network is up and terminate their reconfiguration process.

In a single-hop network, where all nodes can directly hear each other, local update with suppression has about the same performance as the *best* case in SLPL. In both protocols, only the first node waits for $2T_d$, and then sends a message to finish the reconfiguration. The only difference is that here we use an explicit up message instead of a data message. In a multi-hop network, there will be a node in each neighborhood whose timer fires first. These nodes will send up messages in their own neighborhood and suppress all other nodes. The protocol performance depends on the neighborhood size. In general, the benefit of suppression increases as the neighborhood size increases (more nodes). This result is in contrast with the flooding protocol, where its performance decreases as the neighborhood size increases.

Similar to SLPL, in local update with suppression, nodes listen for possible data transmissions during reconfiguration. The protocol has to choose the same polling period as the one used in the regular data communication, and hence we cannot further optimize the parameter for reconfiguration. If the application chooses a scheduled MAC for data communication, this protocol is able to establish local common schedules, which is part of the reconfiguration service 2, as listed in Table I. Since nodes do not coordinate globally, more work is needed to discover neighbors on different schedules and switch them to a single global schedule [12].

The main advantage of local update with suppression is that it significantly reduces the number of control messages, and therefore avoids excessive cost on overhearing. Meanwhile since nodes coordinate among one hop, a late sender can potentially start as early as any of its one-hop neighbors. Thus its overall performance improves in dense networks, where the flooding cost could become prohibitive.

IV. Energy Analysis

In this section we develop analytic models of the FabApp and our protocols. These models help us quickly evaluate and compare performance across a wide range of parameters and to develop best-, worst-, and average-case performance. In Section V we compare our analysis to detailed simulation results, validating our analysis where possible, and extending our results to cases that are intractable analytically.

A. Basic Model

Table II shows our radio energy model, derived from the CC1000 used in Mica2 motes [10]. Energy consumed depends on what state the node is in. Nodes can be in sending, receiving, listening, sleeping or sampling state at any time. The energy in each state includes the cost consumed by both the radio and CPU.

When nodes are sampling the medium, the power consumption is different than listening. The duration of channel sampling is very short, and most of the time is waiting for the radio's crystal oscillator to stabilize (with receiver otherwise turned off). After stabilization, the radio enters receive mode

Symbol	Meaning	Typical Value
P_s	Power consumption in sending	60mW
P_r	Power consumption in receiving	45mW
P_l	Power consumption in listening	45mW
P_{slp}	Power consumption in sleeping	90μW
P_{poll}	Average power consumption in polling channel	5.75mW
t_p	Time needed to poll channel once	3ms
t_{cs}	Average carrier sense time for one packet	8ms
t_{up}	Time to transmit up packet	5ms
T_{lpl}	Default channel sampling period in TinyOS	100ms
T_p	Channel sampling period	Varying
T_d	Clock drift after long sleep	Varying
T_0	Wake up time by ideal clock	Varying

TABLE II

CONSTANTS USED IN ENERGY EVALUATION

very briefly to take one or a few samples of signal strength. Therefore, the average power consumption during channel sampling is much less than that of fully listening. We assume the average power consumption during channel sampling is 5.75mW.

Analysis of multi-hop networks quickly becomes intractable. We therefore explore multi-hop networks in simulation (Section V). Here we consider one hop networks with $n+1$ nodes, all of whom can hear each other. The mean energy cost on each node during reconfiguration can be computed as

$$\begin{aligned} E &= E_l + E_s + E_r + E_{poll} + E_{slp} \\ &= P_l t_{cs} + P_s t_s + P_r t_r + P_{poll} t_p + P_{slp} t_{slp} \end{aligned} \quad (1)$$

where E_l, E_s, E_r, E_{poll} and E_{slp} are the energy consumed in listening, sending, receiving, channel polling (sampling), and sleeping states, respectively. The energy in each state is simply the power consumption of a state multplied by the time spent in that state. Typical values of these parameters can be found in Table II.

Our goal is to minimize this energy consumption. For simplicity, we assume that the activation moments for these $n+1$ nodes are uniformly distributed within $[T_0 - T_d, T_0 + T_d]$. Thus the first node wakes up at T_d before T_0, and the last node wakes up at T_d after T_0. On average nodes wake up at the ideal clock time T_0.

B. Energy Analysis on Idle Listening

First we consider the simplest possible protocol where nodes simply do full-time listening during network reconfiguration. Since we assume nodes reboot uniformly within $[T_0 - T_d, T_0 + T_d]$ the drift delay is $2T_d$. This is the duration absolutely needed for networks to become stable.

In the worst case, the last node to turn on has data to send, and there are no other nodes sending before that. After waking up at $T_0 + T_d$, it still needs to wait for the extra $2T_d$ to guarantee all other nodes become active. Data transmission can only happen at $T_0 + 3T_d$. Thus, in this worst case, the data message delay is $2T_d$ and the whole configuration duration is $4T_d$. Since we assume on average nodes wake up at T_0, the mean duration that each node uses on reconfiguration is $3T_d$. And the mean energy cost is

$$E_{idle\_worst} = 3P_l T_d \quad (2)$$

In the best case, when the first active node has data to send, it can start data transmission at $T_0 + T_d$. Since the data message delay is measured from the moment when the last node is up, *i.e.*, $T_0 + T_d$, the delay becomes zero in this case, and the network is configured at the same time when the first data message is sent. Nodes spend T_d on reconfiguration and consume energy

$$E_{idle\_best} = P_l T_d \quad (3)$$

Besides the best and worst case, on average, the sender wakes up at time T_0 and delay data transmission until $T_0 + 2T_d$. In this case, nodes consume energy

$$E_{idle\_ave} = 2P_l T_d \quad (4)$$

In all cases, idle listening consumes significant amount of energy due to the fact it needs to keep all nodes idling listening during the whole reconfiguration process. In addition to considerable energy consumption, the range of possible energy cost varies significantly.

C. Energy Analysis on Simple Low Power Listening

When nodes perform low-power listening during reconfiguration, the analysis is similar to the idle-listening cases just described, however the cost of listening is greatly reduced because nodes poll the network for activity rather than blindly listening. As explained in Section III-B, reconfiguration with *SLPL* requires same polling periods as data transmission. Since data rate varies with different applications, we use the TinyOS default T_{lpl} of 100ms here.

This analysis corresponds to the FabApp approach [16], with the addition of our optimization to short-circuit configuration on transmission of the first message (Section III-A).

In the best case when the sender wakes up at T_d before T_0, all nodes consume energy

$$\begin{aligned} E_{slpl\_best} &= P_{poll} t_p T_d / T_{lpl} \\ &\quad + P_{slp}(T_{lpl} - t_p) T_d / T_{lpl} \end{aligned} \quad (5)$$

The first part of the equation corresponds to the energy consumption during periodic channel polling, and the second part is the sleep cost.

In the worst case when the sender wakes up at T_d after T_0, each node consumes energy

$$\begin{aligned} E_{slpl\_worst} &= 3P_{poll} t_p T_d / T_{lpl} \\ &\quad + 3P_{slp}(T_{lpl} - t_p) T_d / T_{lpl} \end{aligned} \quad (6)$$

In the average case when the sender wakes up at T_0, nodes consume energy

$$\begin{aligned} E_{slpl\_ave} &= 2P_{poll} t_p T_d / T_{lpl} \\ &\quad + 2P_{slp}(T_{lpl} - t_p) T_d / T_{lpl} \end{aligned} \quad (7)$$

In all cases, *SLPL* requires much less energy than idle listening because it replaces idle listening with much less expensive polling. However, the range of possible energy for LPL-based reconfiguration is quite broad (best-case to worst-case). The goal of our new protocols is to improve both average case and worst case performance.

D. Energy Analysis of LPL with Flooding

In this approach, first active node sends out a control message at the end of its reconfiguration and other nodes flood exactly once to coordinate with their neighbors. Each node spends energy on sending one up message, receiving multiple up messages from other nodes, polling the channel and sleeping for the remaining time.

We assume polling interval for LPL during reconfiguration is T_p. Remember that T_p can be different than T_{lpl}. In order to wake up neighbors, nodes need to flood up messages with preamble T_p.

During flooding, every node need to forward up message exactly once. Let's assume the average carrier sense is t_{cs}, and the transmission time for up packet is t_{up}. The energy a node spends on transmission is

$$P_l t_{cs} + P_s(T_p + t_{up}) \quad (8)$$

A node receives exactly n packets from their n neighbors. And on average it overhears $T_p/2$ preamble for each packet. The energy it spends in receiving is

$$nP_l(T_p/2 + t_{up}) \quad (9)$$

Since nodes reboot in an uniform distribution, the average waiting period before flooding for each node is T_d. Thus low-power listening cost on each node is

$$P_{poll} t_p T_d / T_p \quad (10)$$

The last part of energy is sleep cost:

$$P_{slp}(T_p - t_p) T_d / T_p \quad (11)$$

Substituting Equations (8)–(11) into (1) we obtain the mean energy cost during reconfiguration as

$$\begin{aligned} E_{flood} &= P_l t_{cs} + P_s(T_p + t_{up}) \\ &\quad + nP_l(T_p/2 + t_{up}) \\ &\quad + P_{poll} t_p T_d / T_p \\ &\quad + P_{slp}(T_p - t_p) T_d / T_p \end{aligned} \quad (12)$$

Equation (12) shows a tradeoff with T_p. Increasing T_p reduces the channel sampling frequency, and saves nodes from spending energy on polling. But it also increases the preamble length, therefore increasing transmission and overhearing cost. To minimize E_{flood}, we need to obtain the optimal T_p from the following equation

$$\frac{dE_{flood}}{dT_p} = 0 \quad (13)$$

B-MAC suggests similar approach to optimize polling period based on data rate. But the analysis is based on periodic data traffic and it does not provide a closed form formula. Instead during LPL with flooding network does not generate periodic data and the only traffic is the flooding of up messages.

Substituting Equation (12) into (13), the optimal T_p for reconfiguration is

$$T_p^* = \sqrt{\frac{(P_{poll} - P_{slp}) t_p T_d}{P_s + nP_l/2}} \quad (14)$$

Figure 2 and Figure 3 show how T_p^* changes with average neighborhood size n and T_d respectively. We notice that optimal T_p decreases in networks with higher density in order to offset the energy overhead incurred by flooding. Figure 3 shows that when mostly-off networks are suspended for a longer period of time, the optimal T_p increases as well. This is due to the longer drift periods nodes experience after reboot.

Replace T_p^* in Equation 8, Figure 4 and Figure 5 show that LPL with flooding works well when network density is low. Even reconfiguration cost increases with the increase of density, it still saves more energy than *SLPL* worst case in high density with 12 neighbors. Later on in Section V we use simulation results to validate these analysis.

E. Energy Analysis of Local Update with Suppression

In a single-hop network, the performance of loal update with suppression is similar to the *best* case of the simple low-power listening, as they all finish reconfiguration after the first active node waits for $2T_d$ and sends out a message. The only difference is that an explicit control message is used here, so there is an additional cost on transmitting the message from the first node and receiving it by all other nodes.

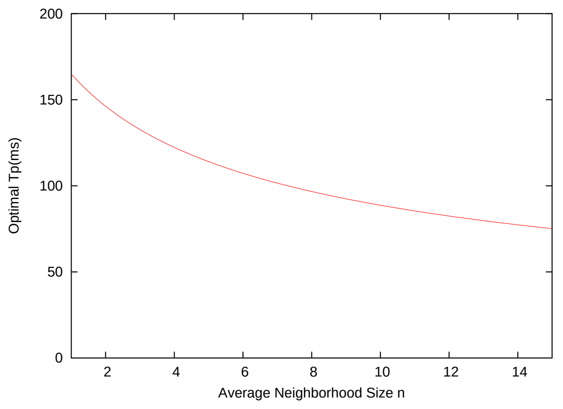

Fig. 2. T_p^* varies with n in LPL with flooding, (Td = 130sec)

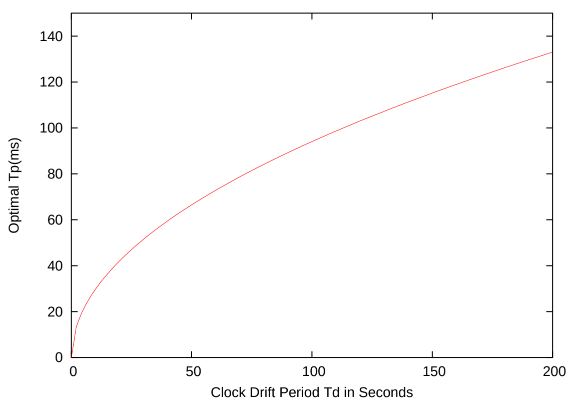

Fig. 3. T_p^* varies with T_d in LPL with flooding, (n = 6)

In multi-hop networks, the performance of local update with suppression can largely vary than the single-hop result. It is intractable to analyze the algorithm in multi-hop networks, because local coordination and suppression are closely related to network topologies and the sequence that nodes turn on. However we expect the performance of local update with suppression improves with the increase of neighborhood size

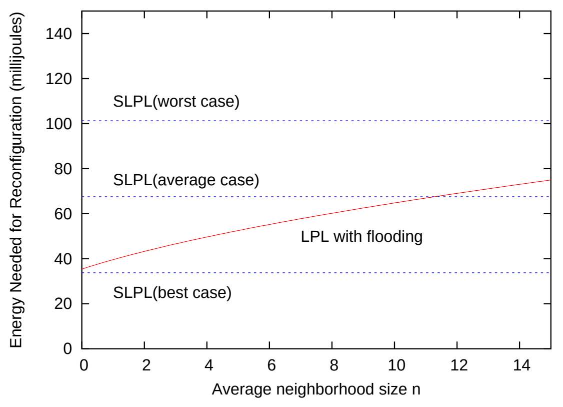

Fig. 4. Optimal E_{flood} for different n in LPL with flooding, (Td = 130sec)

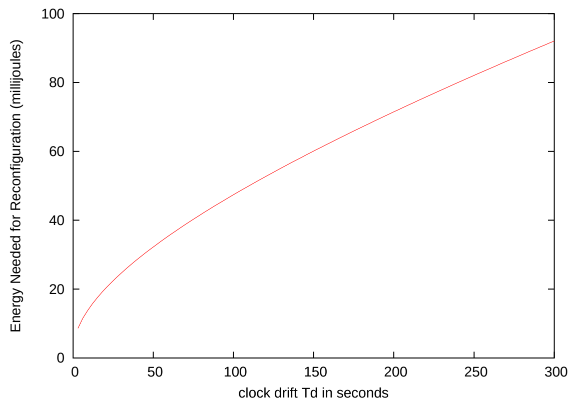

Fig. 5. Optimal E_{flood} for different T_d in LPL with flooding, (n = 6)

due to local updates (quick notice) and suppression (decreased number of control messages). Thus, instead of giving detailed analysis of the energy consumption, we use random topologies to simulate the actual performance of the protocol in Section V.

V. Simulation Results

To evaluate our protocols in more realistic, multi-hop scenarios, we next test our algorithms through simulation. Our results confirm our analysis, and show that both our new approaches can save networks significant amount of energy during reconfiguration. In addition, we demonstrate that *LPL with flooding* is good in low density networks, while with the network density increases, the performance of *local update with suppression* improves.

A. Protocol Implementation and Simulation Setup

We implement both protocols in TinyOS [8] and use Avrora as our simulation platform [18], [17]. Avrora is an instruction-level simulator for the Atmel embedded processor developed at UCLA. As an instruction-level simulator, we are able to test real protocols suitable for deployment, running the same object code we would run on Mica2 motes. However, the simulator gives us the freedom to repeatedly test a large number of topologies.

The simulator uses a simple free-space model of radio propagation. It supports both packet collisions and fading transmission channels. The transmission range of each node is set as 31m in all simulations. We use the radio energy model demonstrated in sectionIV to measure the energy cost during simulations. We count the time spent on each radio state to compute the energy indirectly.

We modify the topology generator *topo_gen*[9] to generate random network topologies. (Originally developed for [5], we extended it to support Mica2 topologies.) The generator places twenty-four nodes randomly in squares of edge sizes ranging from 60–200m. It discards scenarios that are partitioned (assuming any nodes within 31m are connected). Changing area effectively changes the *density* of the topologies. We vary network density from 2 through 12 neighbors, looking at even values. We collect ten different network topologies for average

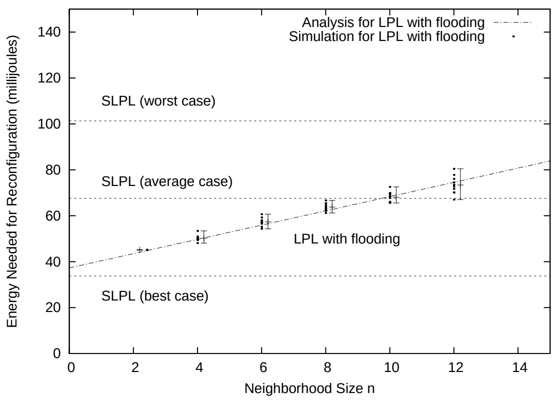

Fig. 6. Mean energy consumption for LPL with flooding in Avrora, (24-node multihop network, T_p=128ms)

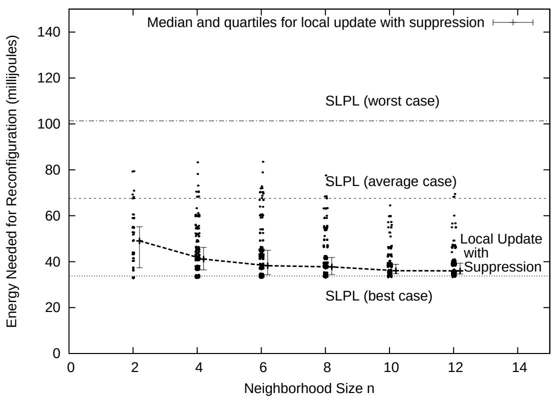

Fig. 7. Mean energy consumption for local update with suppression in Avrora. (24-node multihop network)

neighborhood size around 4 through 12. We consider only two cases for neighborhood size of 2 due to the difficulty in generating connected networks at such low densities.

The purpose of the simulation is to measure the mean energy consumption during reconfiguration after a long sleep. We simulate our underwater seismic monitoring application where nodes sleep for 30 days and then awake. The maximum clock drift after a month-long sleep is T_d of 130s in one direction. Therefore we turn nodes on with a random, uniform distribution in the first 260s of the simulation.

B. LPL with Flooding

In this section we evaluate the performance of our *LPL with flooding* algorithm. As shown in Equations (14), optimal T_p varies based on network drift period and average number of neighbors. When network drift period is 130s, according to Figure 2, the optimal T_p we can choose for LPL with flooding ranges from 150ms to 80ms with 2 to 12 neighbors. In this simulation, we choose T_p as 128ms for simplicity. Nodes start consuming energy when they wake up at a random time. They stop the measurement as soon as they receive the last up message from their neighbors.

Figure 6 shows how the mean energy consumption on each node varies with different neighborhood size for *lpl with flooding*. It compares the analysis (the diagonal line) with simulation (dots show each simulation run, while error bars show the mean, max and min). For context, the three horizontal lines show best, average, and worst case analytical values for SLPL.

The simulations verify our analysis shown Figure 4, matching almost perfectly. It also confirms our expectation, that flooding works well when network densities are low because the cost of overhearing is little, but the cost rises as networks get denser. In all cases, the reconfiguration cost is very predictable.

It is also helpful to compare flooding to SLPL. For sparse networks, flooding consumes less energy than average-case LP, because it allows the network to reconfigure much more rapidly. On the other hand, above densities of 12, SLPL is better on the average, since the cost of overhearing overwhelms the benefits of earlier reconfiguration. Although even there, the flooding is saves energy compared to worst-case SLPL.

We next turn to local update with suppression in search of better performance at higher densities.

C. Local Update with Suppression

We next evaluate how *local update with suppression* performs under different network densities. In this algorithm, senders can start data transmission as soon as they realize the network is stable. They either discover this on their own or on receipt of data or up messages from other nodes. Therefore for the same topology, the duration of reconfiguration varies depending on when the first sender becomes active. Thus in each test case, we simulate all twenty-four possible situations in which each node will be the first sender respectively and collect energy cost for each case. Nodes update their energy usage until that specific sender in the test finishes reconfiguration and is able to start data transmission.

In Figure 7, dots show each simulation run in *local update with suppression*, while error bars show quartiles and medians are connected with a dashed line. The large variance in energy cost for different runs of simulation is because it closely depends on when the first data sender turns on. Local update works reasonably well (better than average LPL behavior) at low densities. It converges on the minimum LPL cost at higher densities by exploiting local information. This improvement is due to the increased probability for the first sender to overhear an up message from larger neighborhood size. Moreover, the number of total control packets drops as well with the increase of neighborhood size due to suppressions. We therefore suggest that local update with suppression is the best choice for reconfiguration in networks with moderate to high density.

VI. Conclusion

In this paper, we present two original algorithms to reduce the energy overhead during periodic reconfiguration for most-off applications. Low-power listening with flooding approach can quickly let the network finish reconfiguration by flooding a control message as soon as one node discovers the network has completely resumed. While in local update with suppression, nodes only notify their direct one hop neighbors about this information to save overhearing overhead. We have implemented both protocols in TinyOS and tested their performance in Avrora. Through analysis and simulations, we show that both protocols are more energy efficient than current approaches. Flooding works best in *sparse* networks with 6 neighbors or less, while local update with suppression works best in *dense* networks (more than 6 neighbors).

In future work, we plan to investigate the robustness of our algorithms in experimental sensor network testbeds to gain experience with different types of node failures. We also plan to evaluate the performance of our algorithms with larger numbers of nodes and more diverse topologies.

ACKNOWLEDGMENTS

We would like to thank Ben L. Titzer (UCLA) for his helps on providing support for Avrora. We also thank Mark Yarvis (Intel Corp.) for his discussions that motivated this research topic.

REFERENCES

[1] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, San Jose, Costa Rica, April 2001. ACM.

[2] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Proceedings of the ACM International Conference on Mobile Computing and Networking*, pages 85–96, Rome, Italy, July 2001. ACM.

[3] A. El-Hoiydi, J.-D. Decotignie, C. Enz, and E. Le Roux. WiseMAC: An ultra low power MAC protocol for the wisenet wireless sensor networks (poster abstract). In *Proceedings of the First ACM SenSys Conference*, Los Angeles, CA, July 2003. November.

[4] A. El-Hoiydi, J.-D. Decotignie, and J. Hernandez. Low power MAC protocols for infrastructure wireless sensor networks. In *Proceedings of the Fifth European Wireless Conference*, pages 563–569, Barcelona, Spain, February 2004. February.

[5] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Lake Louise, Banff, Canada, October 2001.

[6] John Heidemann, Wei Ye, Jack Wills, Affan Syed, and Yuan Li. Research challenges and applications for underwater sensor networking. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, page to appear, Las Vegas, Nevada, USA, April 2006. IEEE.

[7] Jason Hill and David Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[8] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[9] I-LENSE. Topology generator. `http://www.isi.edu/ilense/software/topo_gen/topo_gen.html`.

[10] Chipcon Inc. CC1000 data sheet. `http://www.chipcon.com/`.

[11] Crossbow Technology Inc. Mica2 data sheet. `http://www.xbow.com/`.

[12] Yuan Li, Wei Ye, and John Heidemann. Energy and latency control in low duty cycle MAC protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, New Orleans, LA, USA, March 2005.

[13] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, and David Culler. Wireless sensor networks for habitat monitoring. In *Proceedings of the ACM Workshop on Sensor Networks and Applications*, pages 88–97, Atlanta, Georgia, USA, September 2002. ACM.

[14] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd ACM SenSys Conference*, pages 95–107, Baltimore, MD, USA, November 2004. ACM.

[15] Venkatesh Rajendran, Katia Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *Proceedings of the First ACM SenSys Conference*, pages 181–193, Los Angeles, California, USA, November 2003. ACM.

[16] Nithya Ramanathan, Mark Yarvis, Jasmeet Chhabra, Nandakishore Kushalnagar, Lakshman Krishnamurthy, and Deborah Estrin. A stream-oriented power management protocol for low duty cycle sensor network applications. In *Proceedings of the IEEE Workshop on Embedded Networked Sensors*, pages 53–62, Sydney, Australia, May 2005. IEEE.

[17] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the Fourth IEEE International Workshop on Information Processing in Sensor Networks*, pages 477–482, Los Angeles, California, USA, April 2005. IEEE.

[18] Ben L. Titzer and Jens Palsberg. Nonintrusive precision instrumentation of microcontroller software. In *Proceedings of the LCTES, Conference on Languages, Compilers and Tools for Embedded Systems*, Chicago, Illinois, June 2005.

[19] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the First ACM SenSys Conference*, pages 171–180, Los Angeles, California, USA, November 2003. ACM.

[20] D. Whang, N. Xu, S. Rangwala, K. Chintalapudi, R. Govindan, and J. Wallace. Development of an embedded sensing system for structural health monitoring. In *Proceedings of the International Workshop on Smart Materials and Structures Technology*, January 2004.

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

[22] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, June 2002. IEEE.

[23] Wei Ye, Fabio Silva, and John Heidemann. Ultra-low duty cycle mac with scheduled channel polling. In *Proceedings of the Fourth ACM SenSys Conference*, page to appear, Boulder, Colorado, USA, November 2006. ACM.