WiFlock: Collaborative Group Discovery and Maintenance in Mobile Sensor Networks

Aveek Purohit
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
apurohit@ece.cmu.edu

Bodhi Priyantha and Jie Liu
Microsoft Research
One Microsoft Way
Redmond, WA
{bodhip, liuj}@microsoft.com

ABSTRACT

Low energy neighbor discovery, group formation, and group maintenance is a fundamental service in mobile sensor networks. Traditional solutions consider these protocols separately. In this paper, we introduce WiFlock, an energy-efficient protocol that combines discovery and maintenance using a collaborative beaconing mechanism. WiFlock combines a coordinated synchronized listening and evenly-spaced transmission (SLEST) schedule effectively with one-way discovery beacons to fulfill both purposes. We show that shorter listening duration implies smaller discovery latency and faster group information propagation. Using a novel carrier sensing technique, we achieve a fast wakeup and listen duration of 80μs on a low-power radio. With this listening duration, we evaluate WiFlock on a 50-node test bed with nodes running at 0.2% duty cycles. We show that WiFlock has shorter discovery latency and better scalability than previous approaches.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communications*

General Terms

Algorithms, Design, Wireless Sensor Networks

Keywords

Mobile, Neighbor Discovery, Group Management, Wireless, Sensor Networks, Flocking

1. INTRODUCTION

Mobile sensors are becoming increasingly important in networked sensing systems with applications such as scientific discovery, asset management, and smart environments. These sensors can either be autonomous or be attached to moving entities [1, 10, 12, 19]. These sensor networks must accommodate node mobility. That is, as sensors come and go, a group of nodes needs to discover each other and new neighbors, maintain group membership, and be aware of nodes departure, typically under stringent latency and energy constraints.

Neighbor discovery and group maintenance are not unique to mobile sensor networks. In any ad hoc sensor deployment, the network needs to initiate neighborhood tables and repair link connectivity over time. To encourage nodes to sleep for energy saving, a neighbor discovery protocol usually incorporates long preambles sent by the to-be-discovered node, and periodic waking up and listening by the discovery nodes. These same protocols can be applicable to networks with slow mobility or high energy budgets, where the nodes can afford to communicate frequently.

The challenge comes when the network exhibits "opportunistic flocking" behavior, where scattered mobile nodes occasionally come together for a period of time and then the groups disaggregate again. This phenomena naturally occurs in habitat monitoring [1, 19], where scientists are interested in the encountering of tagged migrating animals. It is also common in asset tracking, smart environments, and search and rescue. When the system is expected to have a long lifetime with a constrained energy budget, the nodes need to carefully trade off between duty cycling and latency.

To motivate the challenges of this flock discovery and maintenance problem, we consider an asset tracking scenario with wireless tags that we call *MeshIDs*. The assets can be servers in data centers, equipment's in hospitals, or products in warehouses. MeshIDs are wireless sensor nodes attached to the assets, and they use multi-hop communication and group formation capabilities to detect encountering. That is, when certain assets are used or shipped together.

Consider a typical logistics workflow. A number of items are picked from different locations to fulfill orders. They are loaded on to trucks driving to various stations such as warehouses or hubs. At each station, some goods are unloaded, and others are added to the truck. The MeshID tags on the items aggregate and propagate neighborhood information so the truck-mount receivers can reliably collect all tag data onboard. In this scenario, each tag needs a low duty cycle to save its battery life since it may spend long periods without any neighbor. However, when they have neighbors, the tags need to quickly discover and propagate the information to the truck reader, so the truck can leave the station as soon as all the required objects are confirmed. Thus, in flocking mobile sensor networks, a group discovery and maintenance protocol should be *energy efficient*, *reactive* (i.e. of low discovery latency), and *adaptive* to scales.

Traditionally, group discovery and maintenance are done in two separate phases: first, a neighbor discovery phase where each node builds its neighborhood table, and then a group maintenance protocol for propagating and aggregating neighborhood tables into a group table. Disco [4] and U-Connect [6] studied low energy neighbor discovery in mobile sensor networks. Under both approaches, nodes

IPSN'11, April 12–14, 2011, Chicago, Illinois.

asynchronously beacon for advertising their existence, and listen for receiving beacons, although they differ in beacon and listen schedule. Group maintenance, when the network cannot afford time synchronization, is usually built on top of low power listening such as BMAC. In BMAC, a node sends long preambles to ensure that the targeted neighbor wakes up, and establishes a point-to-point communication link.

In this paper, we argue that in flocking sensor networks, we have an opportunity to design neighbor discovery and group maintenance protocols jointly. Since these protocols must be executed all the time, a combined protocol can achieve high energy efficiency and low latency. We describe WiFlock that uses a unified and collaborative beaconing mechanism to achieve both discovery and group maintenance goals. The collaborative nature differentiates WiFlock from previous asynchronous neighbor discovery protocols, which mainly considered pair-wise discovery between two nodes. In fact, since group formation is the ultimate goal, we show that it is efficient to perform *one-way* discovery and then use collaborative broadcasting for propagating the information.

The WiFlock protocol uses several techniques for performance improvement.

- When a node is alone, it needs to periodically wake up and check if it has any neighbors. If the node spends majority of its life in the alone mode, then reducing the wake up duty cycle directly translates to increasing node lifetime. We show that WiFlock can achieve $80\mu S$ carrier sense duration in practical environments with moderate interference. This is $\sim 3\times$ less than the state of the art $250\mu S$ reported by U-Connect and leads to much lower node duty cycles.
- When nodes are in a flock, their activities can be synchronized to speed up the propagation of neighborhood and group membership information. The whole network can quickly react to changes in terms of nodes joining and leaving the group. To achieve this, we use a distributed coordination to achieve synchronized listening and evenly spaced transmitting (SLEST) among groups of nodes. We show that it is a lightweight yet scalable technique for joint discovery and group management.
- When nodes within a flock exchange messages for maintaining group membership, WiFlock embeds data messages such as neighbor tables in the long beacon itself, so receivers do not need to keep waiting for the end of the long preambles to establish two way communication.

As a result, WiFlock easily accommodates frequent node mobility by combining the neighbor discovery and group maintenance in to a single long running service, rather than treating them as two distinct and sequential phases. WiFlock is a highly efficient and scalable protocol that can achieve node duty cycles as low as 0.2%. Unlike some of the existing solutions, where the performance degrades quickly as the group size increases, the proposed solution can easily accommodate large groups. We evaluate WiFlock on a test bed of 50 sensor nodes and show that a group can be formed within 3 minutes with 0.2% total duty cycle by using a 80μs listening duration under moderate WiFi interference.

The rest of the paper is organized as follows. Section 2 introduces the basic primitive of two node neighbor discovery on which the group management protocol is built. Section 3 describes our system architecture and collaborative techniques for group management that allow for scalable low-latency continuous group management at very low duty cycles. Section 4 details our hardware platform and the implementation of the various layers of our protocol. Section 5 presents an evaluation of the protocol at scale on a 50 node test bed. We discuss related work in Section 6, and Section 7 summarizes our contributions and concludes the paper.

2. NEIGHBOR DISCOVERY

To accommodate node mobility, the basic operation in neighbor discovery and group management is advertising the node's existence and listening to discover neighbors. However, when to perform these operations greatly impacts the node's energy expense and the network's reactiveness to changes. Therefore, we derive the optimal beaconing duration for advertisement and wakeup period for listening. In this section, we assume that the nodes do not belong to any groups. For many applications, a node spends majority of its time in such a mode.

Nodes use low-power sleep state to reduce energy consumption. The basic operations of a neighbor discovery protocol consists of two parts:

- **beaconing**: In order to advertise its existence, a node wakes up periodically, every q seconds, and transmits a beacon of duration B.
- **listening**: For detecting other nodes' beacons, the node also wakes up periodically, at the period p, and listen to the RF channel for a duration D to detect any beacons from neighboring nodes.

In addition to periodic listening, discovering unknown neighbors involves periodic beaconing. Thus it is advantageous to seek ways to minimize *idle transmit*. We carefully pick the optimal length and period of the transmit beacon to reduce our average power consumption.

Our neighbor discovery protocol builds on the previous work by Kandhalu `et al.` [6], where they introduced the U-connect neighbor discovery protocol. Similar to U-connect, we choose the beacon duration B as:

$$B = \frac{p}{2},$$

which is the minimum length of the beacon that guarantees that at least one node from a pair of nodes will detect the presence of the other. We call this *one-way* discovery, in contrast to pair-wise discovery where both parties discover their counterparts.

Fig. 1 shows the intuition behind this choice. Consider two nodes x and y transmitting the beacons at times t_x and t_y. The time difference is $t_d = t_y - t_x$. Here we consider the case where $t_d < p$; due to periodicity, the same argument applies when $t'_d = n \cdot p + t_d$, where n is an integer. We assume that D is arbitrary small.

In Fig. 1(a), if $B = \frac{p}{2}$ then $t_d = \frac{p}{2}$. Here the beacons of each node barely overlap with the listening so that one (or both) of nodes can hear each other. If $0 \leq t_d < \frac{p}{2}$, then node x can hear node y's beacon, while for $\frac{p}{2} < t_d \leq p$, node y can hear node x's beacon. Thus $B = \frac{p}{2}$ guarantees one-way discovery.

As shown in Fig. 1(b), when $B < \frac{p}{2}$ none of the nodes can hear the other. On the other hand, as in Fig. 1(c), if $B > \frac{p}{2}$, either one of the nodes or both of the nodes can hear each other depending on value of t_d. Although this still guarantees discovery, energy is wasted on the extra overlapping. Of course, in practice, the B can be slighter longer than $p/2$ to accommodate uneven clock drifts in the two nodes.

Note that, when $B = p$, the node behavior degenerates to low power listening (LPL) [9], which guarantees pair-wise discovery and that the listener can discover the advertiser within one beacon period. However the energy expense is significantly higher. The flip side of one-way discovery is that the nodes cannot establish two-way communication immediately. This asymmetry has ramifications on efficient

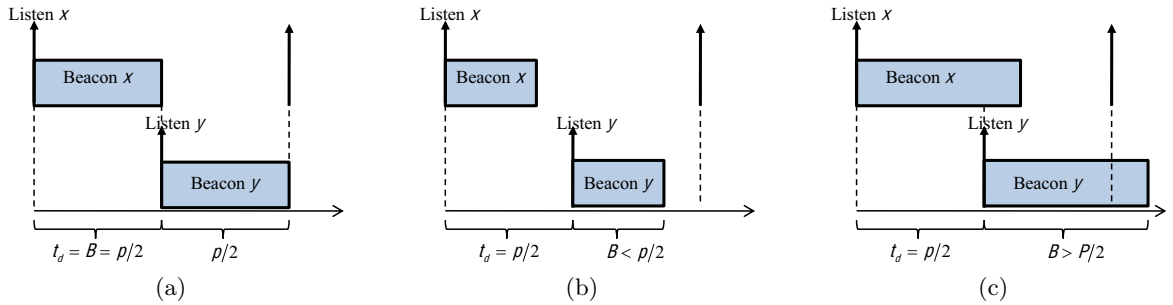

Figure 1: Impact of different beacon durations B vs. beacon period P on ability to discover neighbors and the average energy when two node beacons are separated by $t_d = p/2$. (a) $B = p/2$ (b) $B < p/2$, and (d) $B > p/2$.

group maintenance mechanisms, which we will elaborate in section 3.

Having decided on the beaconing length, we examine how to select the node beaconing period q to minimize detection latency given an energy budget.

Assuming that transmit and receive energy consumptions of a node are approximately the same [13], we analyze the energy consumption of the node using the its transmit and receive duty cycles. We define:

- receive duty cycle: $c_{rx} = \frac{D}{p}$;
- transmit duty cycle: $c_{tx} = \frac{B}{q} = \frac{p}{2q}$;
- total duty cycle: $c = c_{rx} + c_{tx}$; and
- receive duty cycle ratio: $r = \frac{c_{rx}}{c}$.

Then we can show that the worst-case discovery latency, which is equal to the beaconing period q, can be expressed as:

$$q = \frac{D}{2c^2 r(1-r)} \tag{1}$$

From the first and second derivatives of q w.r.t. r, we obtain that when $r = 1/2$, q takes its minimum. Thus, for a given duty cycle (or energy budget), the minimum discovery latency is archived when transmit and receive duty cycles are equal. In this case, the beaconing period q is given by;

$$q = \frac{p^2}{2D} \tag{2}$$

This result establishes two principles. First, it gives the optimal beaconing period that must be chosen. Under the optimal beaconing period, the node spends half the energy listening and half the energy advertising. Second, it shows that the smaller we can make the listening period D the better. Small D means less energy consumed for each wake up for listening and more frequently a node can wake up, and therefore the less often an advertiser needs to beacon.

Figure 2 shows a comparison of the theoretical neighbor discovery latency, vs. the duty cycle, for different wakeup durations D, with $r = 1/2$. We observe, that for total duty cycles $c < 0.5\%$, the discovery latency quickly grows as the wakeup duration increases. We compare the minimum wakeup durations supported by prior neighbor discovery protocols Disco (5ms) and U-Connect (250μs) to that supported by WiFlock (80μs). This confirms that to achieve very small duty cycles, the focus must be to reduce the wake up duration D, to attain reasonable discovery latencies. Hence, when implementing WiFlock, we seek practical ways to shorten D by reducing carrier sensing time (section 4.2).

Supporting Multiple Duty cycles

Although the above discussion assumes nodes with the same energy budget and the same duty cycle, WiFlock can be easily extended to nodes with different energy capacity. A

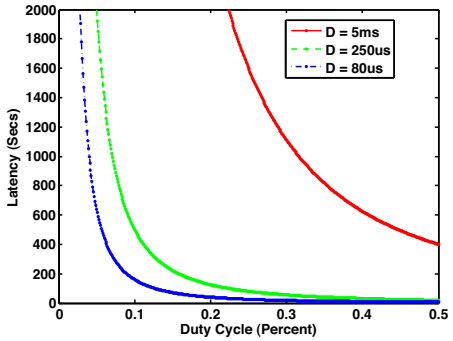

Figure 2: The figure shows the theoretical discovery latency for 2 nodes, with 5ms(minimum in Disco), 250μs (minimum in U-Connect) and 80μs(in WiFlock) wakeup duration, to discover each other as a function of their percentage duty cycle of operation.

moving vehicle or a robot is likely to be willing to use higher duty cycles, while comparatively energy constrained nodes, such as an energy scavenging sensor node, is likely to tolerate long discovery latencies to preserve energy.

Nodes that are willing to spend more energy can transmit longer beacons and listen more often. For example, a node can double the length of its beacon duration to p, which causes the node power consumption to increase by 50%. Since their energy constrained neighbors would sample the channel at frequency p, this may make some of the neighbor discovers bidirectional, reducing the overall discovery time. Similarly, doubling the wake up frequency also increases the power by 50%, which can also decrease discovery time due to removal of some unidirectional discoveries. If a node doubles both beacon length and removal frequency, all its neighbor discoveries will become bidirectional, which will reduce the discovery time further. If a node wants to have finer-grained control over the power consumption, it can selectively increase the beacon length or sampling frequency only for some of the beacons and sample intervals. If the node can afford to more than double its minimum energy consumption, it may increase the beacon frequency, while paying attention to the possibility of increased beacon collisions.

A node running WiFlock may reduce energy consumption by reducing the beaconing frequency and selectively turning off some of the sampling events. However, persistently reducing the beaconing frequency, and specially the sampling rate, may make neighbor discovery time arbitrarily long. In the rest of the discussion we assume that all nodes involved in group formation and maintenance agree with the same minimum duty cycle (such as 0.2% duty cycle in our evaluation).

3. GROUP-WIDE COORDINATION

While the standalone beaconing discussed above is a natural extension of U-connect, it does not solve the group maintenance problem completely. First, the one-way neighbor discovery guarantees only one of the nodes in a neighboring pair to discover the neighboring relationship. This unidirectional nature of discovery does not allow even adjacent nodes to discover all their neighbors. One way to overcome this is to implement explicit handshake mechanisms, where a node must schedule a special acknowledgment frame after every beacon where listeners can register their presence, as employed in [4, 6]. The downside is that this approach is not efficient for large group formation. Point-to-point communication between all group nodes gives rise to collisions and high message overhead for larger groups.

Second, a group may span more than one communication hop. The node-to-node neighbor discovery does not provide a mechanism to propagate neighbor information over multiple hops. Traditional solutions typically employ another group management layer after neighbors are discovered. The group management layer may flood the network of known nodes periodically to keep the membership information fresh and deal with nodes that move away from the group. However, at the same time, since the network is mobile, all nodes still need to run the discovery protocol for new nodes to join the group.

In WiFlock, the continuous group maintenance and neighbor discovery is combined into a single beacon schedule to achieve high energy-efficiency and scalability. To take advantage that multiple nodes are within a small neighborhood as in the flocking configuration, we exploit a collaborative technique for performance improvement: Synchronized Listening and Evenly-Spaced Transmitting (or SLEST for short). Throughout this section, our analysis assumes perfect radio channels. We consider realistic radio channels in our implementation and real test bed evaluation.

3.1 Synchronized Listening

Nodes participating in neighbor discovery transmit periodic beacons of duration $p/2$. These beacons are typically large enough to accommodate more than an entire packet. For example, in our implementation with 0.2% duty cycling, the beacon lasts for 40ms, which can accommodate twenty 128Byte long RF packets at a 256kbps data rate. WiFlock takes this as an advantage and re-use these beacons to propagate group membership information as well.

To achieve high propagation speed of membership information, WiFlock efficiently synchronizes node listen times. As an illustration, Fig. 3(a) shows three neighboring nodes whose listen instances are not synchronized. Here node A can hear the beacons of nodes B and C. Node C can hear the beacons of B. However, node B does not hear the beacons of both A and C, while node C does not hear the beacons of A. Hence, with unsynchronized listening, only unidirectional neighbor discovery is possible among neighboring nodes. In contrast, Fig. 3(b) show the three nodes with their listen instances synchronized. Assuming the listen duration is arbitrarily small and assuming that no two neighboring beacons attempt to beacon at the same time, with synchronized listening, each pair of nodes can hear each other's beacons (Section 4 describes how we deal with finite listen durations, as well as possible simultaneous beacon transmissions).

To realize efficient listen time synchronization, WiFlock uses an approach similar to XMAC [2], where each beacon is transmitted as a sequence of data frames. Each frame acts as a self-contained packet, so that if a node wakes up in the middle of a beacon, it can decode the next frame. Each frame within a beacon includes a *frame offset* n from

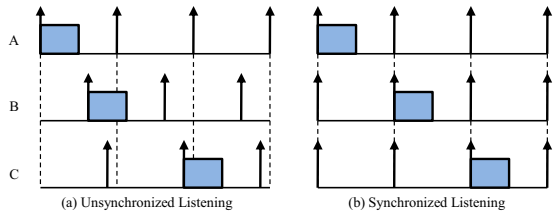

Figure 3: An example showing the difference between unsynchronized vs. synchronized listening among three nodes. The rectangles represent the periodic beacons while the impulses are the listening instances.

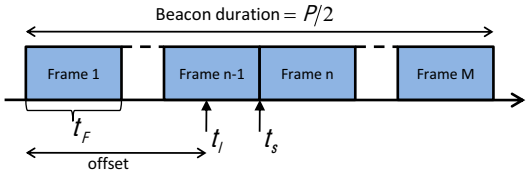

Figure 4: An example of computing the listen offset using multi-frame beacon.

the start of the beacon, thus, by measuring the time interval between its current listen time t_l, the start of the next frame t_s, and using the frame offset n, the receiving node can compute the offset between its current listen time and the start of the beacon by $(n-1)t_F - (t_s - t_l)$, where t_F is the frame duration (Figure 4). Since a beacon transmission starts at the listen time of a node, this offset is equal to the offset between the listen times of the two nodes.

Nodes use these offsets to achieve group-wide time synchronization and adjust their activities. Given time offsets between neighboring nodes, Werner-Allen `et al.` have shown that it is possible for an uncoordinated group of nodes to achieve time synchronization using only peer-to-peer interactions, rather than through a group-wide coordinated effort [17]. However, this way of time synchronization can take a long time to converge. For example, Werner-Allen et al. [17] report that the convergence time for a group of 24 nodes is 284 seconds.

To speed up convergence rate, we use a more coordinated approach. Specifically, a node will synchronize towards the smallest ID it hears. Of course, this has an issue due to the one-way discovery mechanism. Since only one of a pair of nodes can discover the other, a group can bifurcate to two synchronized groups: each occupies one half of listening period p and none can hear the other group. To overcome this problem, we take advantage of the data frames in the beacons and perform on-demand beacon extensions. Each data frame contains the group membership information, i.e. a list of sorted node IDs. While each node always synchronize its listening time towards the smallest ID it can hear, it also monitors the other IDs it receives that are not in the group. If any nodes with a higher ID is discovered, the node will temporarily extends the beaconing to a full period p. The receiving node can then synchronize to this node, and through that, synchronize to the node with the smallest ID. The implementation details, including establishing on-demand bidirectional links, are described in Section 4

3.2 Evenly-Spaced Transmitting

The primary purpose of evenly spaced transmission is to collaboratively improve the group formation latency of a new node joining an already established group or a transient group. As an illustration, take an example of an established 2-node group, with each beacon having a beacon period of

100 time "slots" (each time slot refer to a listening duration). If the nodes are allowed to beacon randomly in the 100 slots, a new node wishing to join the node may have to wait for anywhere between 0 slots to 98 slots before it hears a beacon. However, if the two beacons are evenly spaced, i.e. each is 50-slots apart, an incoming node will now have to wait only between 0 and 50 slots to hear a beacon. Thus, a group can reduce discovery latency through collaboratively spacing out beacons.

Additionally, if a dense group with large number of nodes transmits beacons in an uncoordinated fashion, there can be packet collisions. Evenly spaced transmissions can minimize the number of collisions and hence allow the protocol to scale well to large number of nodes.

The mechanism for collaboratively spacing out beaconing times is as follows. The node with the smallest ID picks an arbitrary time slot as slot number zero. Each node maintains the group membership table, sorted by the node ID. Each node maintains a *current slot* counter, which is computed based on the information it has received so far. Using this current slot counter, the node transmits the slot number of the beaconing slot with its beacon message. It also updates the *current slot* counter based on the slot counter information received with a beacon containing a node id that is equal or less than the smallest node id it has received so far.

Since there are $2q/p$ beacon slots in each beaconing period q, each node i uses its position within the sorted group membership table l_i and the size of the membership table s to compute its transmit slot $s_T(i)$ as follows:

$$s_T(i) = \frac{p \times l_i}{2s}.$$

The node transmits its beacon when its current slot counter reaches this transmit slot value. However, transmitting on a fixed slot can cause collisions, especially during the early stages of building the membership tables. This is because each node computes transmit slots assuming there are only a small number of nodes in the group.

To overcome such persistent collisions, we introduce jitters in the node transmission schedule. Instead of selecting the its computed slot number, the node selects a random slot number uniformly distributed within a small range of slot numbers centered around its computed transmit slot number. The range for random jitter is computed from the number of nodes in the group and the beaconing period. The slot number transmitted with the beacon is the slot number of the actual beacon slot, rather than the computed slot number.

As group membership tables are propagated using beacons, each node merges tables from its neighbors with its own table while keeping the table sorted. Under stable group membership, the membership tables at the nodes rapidly converge. The *current slot* counter value also converge to a consistent value across nodes, since this is always computed with respect to the smallest ID node in the group. Hence, under steady membership, the beacon transmission schedule converges such that node beacons are evenly spaced.

We note that, since a group can span over multiple communication hops, the transmission schedule computed by WiFlock is more relaxed compared to schedule computed by graph coloring using two-hop connectivity [11]. However, when the diameter of the group is small, the evenly spaced schedule would be close to a schedule computed by graph coloring.

We need to be careful again here with the short beacon size of $p/2$. Due to one-way node discovery, if all nodes beacon in the first half of their p length beacon slot, the entire group acts as a single phase node. Thus, a new node joining in either would be able to hear all nodes or would rely on its own beacon to make its presence known to the group. While, subsequent discovery still takes place with synchronization, the advantage of evenly spacing out beacons is reduced.

Thus, we allow nodes part of a synchronized group to randomly pick whether they transmit in the first half of their p length beacon slot or the second. Due to listen time synchronization and slight overlap between beacons, nodes within the group can still all hear each other. This mechanism was shown to improve the average latency of incoming nodes to discover the group in our experiments.

3.3 Handling Node Departure

To handle nodes leaving the group, each node entry in the group membership table contains a time-to-live (TTL) value. Each node initializes this value when advertising itself in the beacon message. The initial TTL value is set based on the freshness requirements (how quickly the node departure should be detected), and the maximum number of hops expected in a group.

The TTL value at each node is updated as follows. Whenever a node receives a membership table from a neighbor, it decrements the received TTL value of each node, and then compares these with the values stored locally. If the local neighbor table does not contain a received node ID, the new ID is added to the table with the updated TTL. If the node already exists, and the received TTL is greater than that in the local table, the local TTL is updated. After every beacon transmit, the node decrements all the local TTL values by 1. Any entry with a TTL equal to 0 is removed from the group table. For example, if we use a value of 7, then with a 3 hop network and a persistence time of the node is 3 time periods at the last hop.

Since the TTL value of a node is decremented at each hop, the initial TTL value determines how far a given node ID is propagated within the group. Since all the nodes need a uniform view of the minimum node ID, it is important that the initial TTL correctly reflects the maximum hop count within the group for correct operation of the synchronized listening and evenly spaced transmissions. On the other hand, a large initial TTL will require a longer time for the node information to be removed once the node leaves the group.

4. IMPLEMENTATION

In this section we present several implementation details of WiFlock, focusing on the hardware platform, the techniques to reduce wakeup and listening period, and the implementation of SLEST.

4.1 Hardware Platform

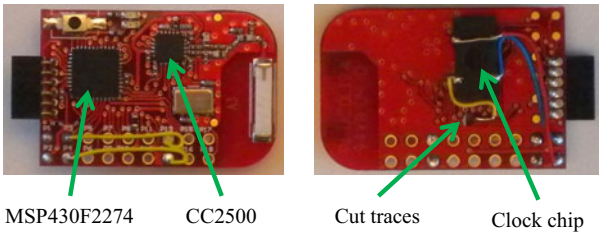

Figure 5: The modified EZ430-RF2500 development platform used for evaluating WiFlock. The figure shows the HW modifications for attaching the 32kHz clock chip.

We use the TI EZ430-RF2500 development tool as our

Mode	Time	Frequency	Current
Idle-to-Rx	88.4μs	1	7.4mA
Carrier Sense	~80μs	1	18.8mA
Rx-to-Idle	0.1μs	1	7.4mA
Calibration	721μs	Variable	7.4mA

Table 1: Breakdown of radio current consumption on every wakeup.

HW platform [15] (Figure 5). This module has a CC2500 radio and an MSP430F2274 microcontroller with 32kB Flash memory and 1kB RAM. It can be interfaced to a PC using a USB connector, which provides both a RS232 communication link and the firmware programming support.

Nodes running WiFlock use low power sleep to save energy. However, the TI evaluation module does not have a real time clock crystal that enables low power sleep with a relatively accurate wakeup timer. To overcome this, we added a 32kHz clock chip [5]. This clock chip consumes $\simeq 2\mu$A current, which is similar to the power overhead of the MSP430F2274 when driving a 32kHz clock crystal. The clock chip is introduced because it is difficult to modify the hardware module to properly attach a crystal.

4.2 Minimizing Discovery Latency

In Section 2, we observe that reducing the wakeup duration D is the most effective way to achieve a reasonable latency under a low duty cycle. The purpose of this wakeup duration is for a node to periodically wake up and detect any ongoing beacon transmissions by detecting the presence of a busy RF carrier. So our aim is to minimize D, without adversely affecting the node's carrier sensing performance.

Radio listen time refers to the duration for which the radio is in receive mode for either carrier sensing or packet receptions, as opposed to the low power SLEEP mode. Table 1 gives the current consumption and time for operations that must be performed by the radio for every wake up. The VCO characteristics of the radio vary with temperature and supply voltage changes. Frequency synthesizer self-calibration can be performed manually when node operating conditions change. The current consumption for the radio Rx-mode, which includes carrier sense and radio packet reception states, is 18.8 mA for the CC2500 [13]. Thus, apart from beacon transmissions, the Rx-mode dominates the power budget of the radio.

Parameters Affecting Carrier Sense Times

Carrier sensing is done by measuring the Receive Signal Strength Indicator (RSSI) value, and comparing it against a threshold to determine the presence of the RF carrier. The duration to perform a valid carrier sense depends on the time required by the radio to provide a valid value for RSSI, i.e. the RSSI response time. The time to obtain a valid RSSI value depends on a number of configurable parameters of the radio. For example, in CC2500, the RSSI response time depends on receiver filter bandwidth, data rate, modulation format, and AGC(Analog Gain Control) module parameters [14]. Specifically, the following factors impose the lower bound on the RSSI measurement time:

AGC Filter Length. The RSSI value is a byproduct of the AGC module, which uses the estimated RSSI to keep the signal level input at the demodulator constant, regardless of the signal level at the antenna. Since the RSSI signal is used to set the gain of an internal amplifier, it is generated by low-pass filtering the input signal to prevent frequent AGC gain switching. The length of this low pass filter affects the time required for the AGC to report a valid RSSI value.

AGC Wait Time. The AGC module has a configurable wait time after each gain adjustment to stabilize the control loop. A very short wait time may cause the AGC module to report unstable gain values, prolonging the final settling time. While a long wait time takes longer to generate a valid RSSI signal since the AGC ignores all samples during the wait time.

Signal Power. AGC settling time and hence RSSI response time depends on the power of the input signal. A stronger input signal can lead to a longer AGC settling time than a weaker input signal, since a strong signal may require multiple gain changes. While the theoretical minimum response time can be computed from the equations in [14], the stochastic nature of received power makes empirical evaluations necessary.

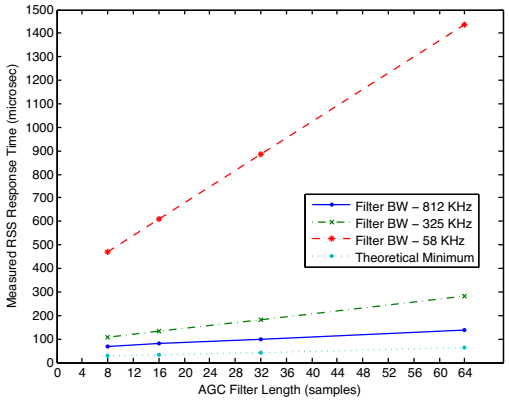

Figure 6: Measured RSSI response time as a function of filter length and filter bandwidth. A higher filter bandwidth reduces the RSSI response time. A lower filter length corresponds to lower RSSI response times.

In general, the larger filter bandwidth, the higher the data rate, and lower the AGC filter length, the lower is the theoretical minimum RSSI response time. Figure 6 shows the measured RSSI response time as a function of the AGC filter length and filter bandwidth.

Empirical Evaluation

We empirically evaluate the effect of RSSI response time setting, obtained by varying AGC parameters, on the accuracy of carrier sensing. Carrier sensing is performed through comparing the detected energy (RSSI value) to a fixed threshold. We establish this threshold as the 95th percentile RSSI value observed by a node in a quite environment with no valid transmissions. Carrier sensing accuracy is measured in terms of percentage error, defined as the number of valid transmissions missed by a receiver node to the total number of transmissions by the sender.

Figures 8 & 7, show the percentage carrier sensing error for nodes with different RSSI response time settings. The experimental setup included a pair of nodes, configured as a receiver and a sender. The sender was set to continuously transmit data at maximum power of 0dBm. The receiver was programmed to wakeup every 0.5 seconds and stay awake until a valid RSSI reading was obtained, i.e. the radio was on for the RSSI response time. The receiver node logged the RSSI value to the PC through a USB port. The distance between the sender and receiver was varied in order to change the signal strength at the receiver node. The experiment was repeated with 12 different combinations of AGC filter length and wait time. 1000 readings were obtained for each setting and distance. The carrier sense threshold was established by obtaining the 95%tile value of 1000 RSSI readings obtained at the particular settings, with the sender node turned off.

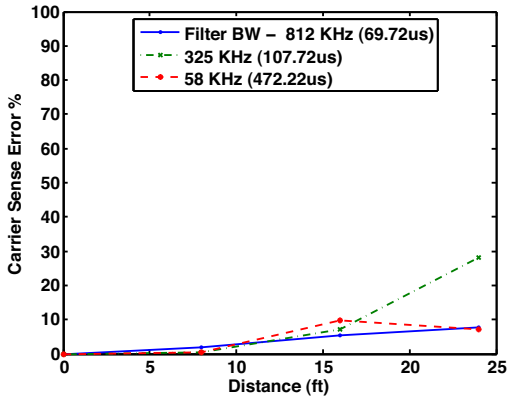

Figure 7: Error rate in detecting a transmitting carrier at different filter bandwidths as a function of distance, with a filter length of 8 samples with CS threshold of 95%tile. Filter bandwidth does not show a clear effect on carrier sensing error rates.

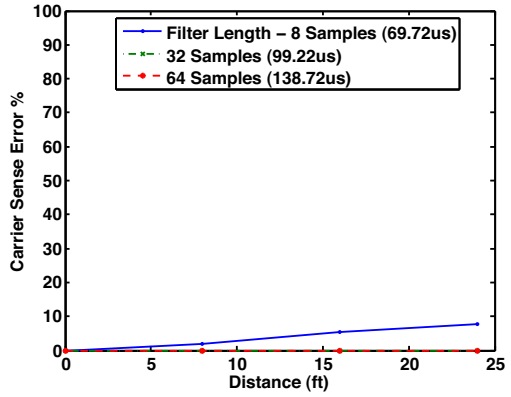

Figure 8: Error rate in detecting a transmitting carrier at different filter lengths as a function of distance, with filter bandwidth 812kHz and CS threshold 95%tile. 8 sample filter length provides very short carrier sense times of $< 70\mu$s, but can cause more errors as signal strength drops.

Every RSSI value below the threshold was considered as correctly detected carrier, while the number of missed detection contributed to the percentage carrier sensing error. We observed that selecting a wider bandwidth did not have a pronounced effect on carrier sense errors, all other factors being constant. However, a smaller filter length (8 samples) caused about 10% missed detections when the sender was moved 25 feet away resulting in a weak signal at the receiver. Thus, there is a trade-off between energy consumption cost due to a higher CS time vs. the latency cost of not reliably detecting some valid transmissions.

Our experiments show that with a AGC filter bandwidth of 812kHz and filter length of 8 samples, a carrier sense time of 70μs is achievable. This allows us to use small listen slots of 80μs duration, with some guard against interference sources. This is a 3X improvement over prior neighbor discovery protocols [6], providing the corresponding improvement in latency for similar duty cycles. This allows us to achieve reasonable discovery latencies even with very low duty cycles such as 0.2%. We note that these parameter settings should only be used for carrier sensing. They may not be optimal for data communication. WiFlock nodes update these parameters with a different set of values when transmitting or receiving data.

Node Listen-Wakeup Scheme

Energy detection based carrier sensing suffers from the possibility of *false wake ups*, due to interference from other sources such as WiFi, which may occupy overlapping channels. Depending on the nature of interference, the radio may detect energy and stay in receive mode for a long time in the absence of valid transmissions. Thus, we employ a tiered scheme for countering this effect and minimizing node listen durations.

First, each node uses the short duration energy detection (80μs) to detect the presence of an ongoing transmission. Second, if an ongoing transmission is detected, the node keeps the radio in the receive mode for the duration of a single frame, while continuously searching for the sync word; if the carrier becomes free during this search, the radio and the node goes back to sleep. Third, if a sync word is detected the radio remains in the receive mode until an entire packet is received or until the carrier becomes free.

In Section 5, we evaluate the robustness of this tiered scheme against overlapping WiFi interference, and obtain a typical carrier sensing duration of 80μs under WiFi interference.

4.3 Multi-Frame Beacons

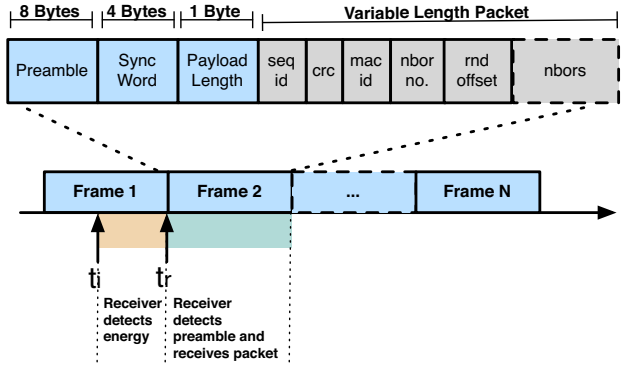

Figure 9: Format of the WiFlock beacon with multiple contiguous frames with their contained fields. A receiver waking up at time t_i tracks the beacon frame until the next frame boundary at time t_r, after which it can detect the preamble and eventually receive the packet.

At very low duty cycle (for example,$\sim$0.2%) operation, which is desired by the WiFlock protocol, the advertising beacon needs a very long preamble of length ($P/2$ or $40ms$). A listening node using a busy carrier preamble such as in U-Connect [6], on average, would track (radio remains in receive mode) the preamble for $20ms$ for every packet reception. This is equivalent to receiving 20 128-byte packets at a baud rate of 256Kbps. With special preamble architecture, consisting of back-to-back frames with group advertisement packets, we reduce radio on times as well as provide a mechanism for implicit listen phase synchronization.

The problem of long preambles has been mentioned and solutions proposed in MAC schemes like X-MAC [2]. X-MAC takes the approach of sending a strobed preamble with encoded destination addresses. The strobed preamble allows nodes to minimize the preamble tracking time while the encoded destination addresses allow nodes to avoid receiving unintended packets.

In contrast, WiFlock beacon architecture addresses different objectives and has following features –

- It has a predictable data transmission requirement, where nodes advertise group information, that are contained in

relatively short frames, periodically. It operates at a very low duty cycle, which allows sending multiple data packets (frames) instead of just packet header information. This reduces the maximum preamble tracking time of listening nodes to the length of a single frame.

- The WiFlock beacon consists of multiple frames with no gap between them. This enables the nodes to use a short listening period to detect an on going transmission, since there is no danger of the listening interval falling in to a carrier free gap.

The beacon is implemented as a very long continuous transmission that includes multiple frames, where each frame is a qualified CC2500 packet preceded by a preamble (hence, a receiver waking up in the middle of this transmission can receive the next frame). The beacon is 10% longer than the required $p/2$ beacon length. The additional length ensures that a node that wakes up at the very end of the $p/2$ period can still receive a valid frame. The additional length also helps mitigate effects due to clock drift.

The long gap-less beacon is realized through the radio's [13] infinite length packet mode, which allows arbitrary length packets. The infinite length packet mode is a feature of a range of newer Texas Instrument's RF transceiver chips including CC1100, CC1100E, CC1101, CC1150, CC2500, and CC2550. Infinite packet length mode can be used to transmit and/or receive until TX or RX mode is turned off manually.

The nodes employ a random back-off mechanism before beginning transmission if another transmission is detected. In addition, a random jitter is introduced in beacon transmissions to account for exposed terminal and hidden node issues. The back off and jitter moves the beacon transmission by multiples of p long beacon slots.

Figure 9, shows the structure of a WiFlock beacon. Each beacon consists of multiple frames. Each frame, similar to a single CC2500 packet transmission, has a preamble, a sync word and payload length field, followed by the variable length packet, with a frame sequence id and group information. Since each frame acts as a fully qualified CC2500 packet, a receiver node radio can detect and receive the packet embedded in the frame as per its normal link layer protocol. The sequence id field in each packet, embedded in each frame, specifies the corresponding frame's position in the beacon. The receiver uses this sequence id for listen synchronization. The length of the fields in each frame can be configured by the user as per application requirements. The experimental setup used for this paper was configured to use a 1-byte sequence id, 2-byte CRC, 1-byte random offset, 1-byte neighbor number, and 1-byte neighbor mac-id fields.

For large groups where the group information does not fit in one frame length (256 Bytes), the nodes encode frames with portions of the group information in a cyclical fashion. For example, the first 243 bytes of a 486-byte group table is encoded in frame-1 of the beacon, the second half is encoded in frame-2, frame-3 encodes the first half again, and so on. A receiver after receiving a particular frame figures out that it has received a part of the group table and must stay on to receive subsequent frames. The receiver may need multiple beacons to obtain the entire group table from a node.

4.4 Propagating Group Information

WiFlock uses synchronized listening to enable fast group information propagation. Listing 1 shows the pseudo code the nodes use for propagating the neighbor table and achieving synchronized listening. The synchronized listening attempts to synchronize to the smallest id node. However, the unidirectional discovery may prevent the smallest beacon

Listing 1: Pseudocode for phase sync algorithm

```
// Synchronize if hear a sending node that
      is sync'd to a lower id node
if (urSyncNeighbor < mySyncNeighbor) {
        syncListen(rxFrameSequenceId);
} else {
        // Extend beacon if the sender
             must sync
        ExtendBeaconFlag = TRUE;
}

// Compute offset and advance next wakeup
void syncListen(rxFrameSequenceId) {
        del = rxFrameSequenceId *
             BEACON_FRAME_TIME;
        if ( del >= BEACON_OFFSET_TIME &&
             del <= BEACON_TIME )
                del = del -
                     BEACON_OFFSET_TIME;
        else
                del = 0;
        if (del == 0)
                return;
        advanceNextWakeup(del);
}
```

information to propagate from node to node. The top half of the code shows the logic used to get around this. If a node i receives a beacon form a node j with a smallest id (the first node of the frame) that is large than its own smallest id, the node i sets its `ExtendBeaconFlag`. The next time the node i beacons, it uses a beacon of duration p, which lets node j hear that beacon, hence receive the current smallest node information at i. The bottom half of the code shows how nodes use listen time offset to synchronize to the smallest ID node.

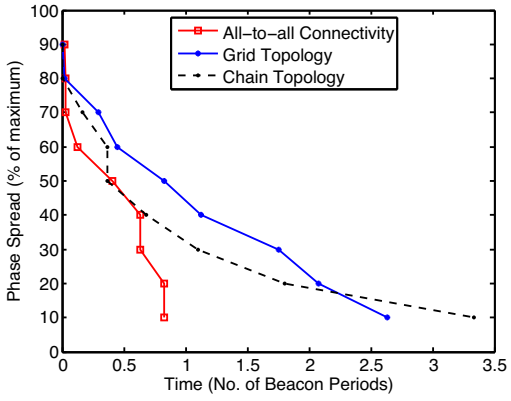

Figure 10: Percentage time spread achieved vs. beacon periods for group of 6 nodes under three network topologies - all-to-all connectivity, grid and chain.

We used a 6 node test bed to evaluate the synchronization performance under 3 network topologies - all-to-all connected topology, a grid topology where a node could communicate with 2 or 3 other nodes depending on its position, and a chain topology. Each node was programmed to set a GPIO pin to high when its radio was in receiving mode. Using a logic analyzer connected to all 6 nodes, we observed the timing of node listen intervals. We determine the degree of synchronization within the group through *time spread*, which is defined as the ratio of the maximum phase difference between any two nodes in the group, to the listen period. As WiFlock has a 10% overlap in transmit beacons, we assume that nodes are synchronized when the time spread is <10%. From Figure 10, synchronization was achieved within 1 beacon period for the all-connected case,

while around 3 beacon periods were required for the chain topology.

4.5 Sorted Group Tables

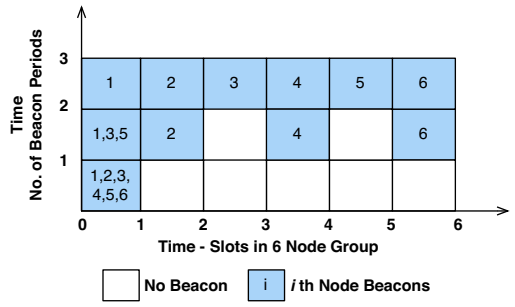

Figure 11: Example of nodes assuming evenly spaced beacon positions from a 6 node experiment with all-to-all topology.

WiFlock uses the ranking based on node mac-id's to evenly spread the node beaconing within the node beacon period. To minimize the processing overhead, nodes maintain a sorted group table locally as well as in the beacons. Starting from the node's own ID, whenever a node obtains a group table from an advertised beacon, it performs a computationally inexpensive ($O(n)$) merge operation to keep the table sorted. The number of comparisons performed is nearly optimal in this algorithm, thereby minimizing processing overhead on receiving every group table update.

Figure 11 visualizes how the sorted tables progress among a group of 6 nodes. The nodes were instrumented with a logic analyzer to obtain the timing. Here, the beacon period is divided to 6 equal sized slots for easy visualization (each slot shown contains multiple beacon slots). Initially, with no information about the other nodes, all the nodes transmit in the first slot. The random jitter introduced during the beacon transmissions prevent persistent collisions and allows the group information to propagate. Nodes 5 and 3 beacon before node 1 in the second period, and therefore do not shift to their respective slots. Nodes 2 hears three other nodes beaconing ,including the lowest-id node 1. As node 4 has not beaconed yet, node 2 computes its position relative to a group membership of five nodes only. Eventually, node 4 beacons in its correct position since it has heard beacons from all other nodes in the group. By the third beacon period, the group table is consistent among all nodes and the beacons are evenly spread out.

5. PERFORMANCE EVALUATION

In this section, we empirically evaluate the performance our WiFlock implementation. Through a series of experiments, we study the energy consumption, group formation latencies, and scalability of our solution in realistic deployment conditions.

5.1 Radio Listen Time

Radio listen time duration D is a critical parameter in WiFlock, which not only determines the energy spent on the listening model, but also affects the beacon length. While in Section 4.2 we show that the carrier sensing time can be made as low as 70μs, we must evaluate the choice in realistic environments, especially with WiFi interference. WiFi transmissions can cause the radio to detect energy and stay awake unnecessarily.

To assess the effect we setup a 802.11 network consisting of a802.11 b/g access point and a laptop equipped with an 802.11 wireless radio in infrastructure mode. This laptop was used to generate a constant stream of 1,500-byte UDP packets using the *iperf* tool. The *iperf* tool was used to vary the bandwidth utilized by the 802.11 transmissions. Another laptop, connected to the access point through an Ethernet cable, acted as the traffic sink for the WiFi network.

A WiFlock node was placed adjacent to the transmitting laptop. The node radio was set to the same frequency band as that of the 802.11 channel to force maximum interference. The node was instrumented with a logic analyzer to obtain accurate timing when the radio changed modes. The listen time was averaged over 1000 wake up's of the radio. Each experiment was performed 10 times at different times over the course of a day.

Figure 12 shows the average time the radio spent in receive mode for idle listening, i.e. with no valid WiFlock beacons, as 802.11 traffic is increased for 0 to 10 Mbytes/second. We observe that the radio idle listening time increases with increasing 802.11 traffic.

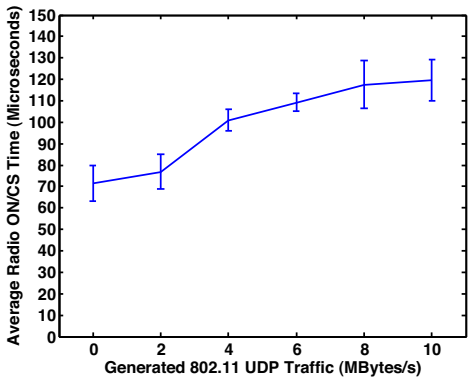

Figure 12: The effect of interfering WiFi traffic on energy-detection based carrier sensing scheme. The average radio idle listen times increase as generated 802.11 UDP traffic is increased.

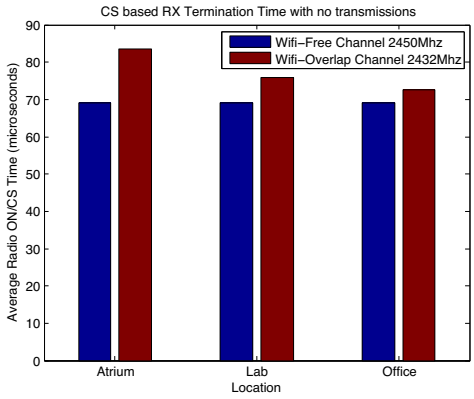

Figure 13: The energy detection based carrier sensing performance in common interfering environments. The presence of WiFi in real environments increases the average idle listen time by 10-20%.

However, Figure 13 shows the average radio idle listening time in typical environments where 802.11 traffic is not generally constant. We perform the experiment at 3 different location and in 2 channels with and without overlapping WiFi channels.

The channel without Wi-Fi overlap had nearly constant average ON times of 70μs, the ON time increased to 84μs at the location with maximum activity. Based on these observations, we select 80μs duration as the *expected* listen

slot size to accommodate possible WiFi interference. The 80μs is $\simeq$3X smaller than U-Connect, the previous state-of-art neighbor discovery protocol[6].

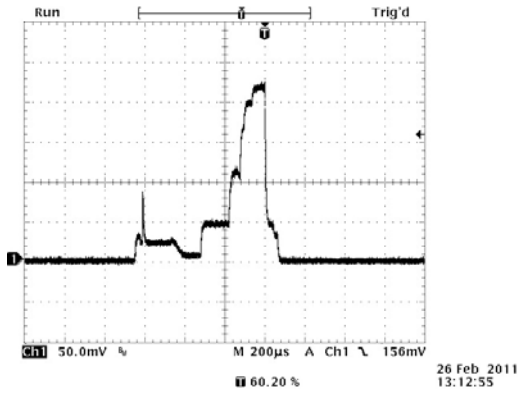

Figure 14: Oscilloscope trace showing current consumption (in mV using a 10.3 Ω resistor in series with power supply) during every node wakeup for listening.

Figure 14 shows the current consumption during every node wakeup. The MSP430 micro-controller consumes ~2.8 mA when operating at 8 MHz. The CC2500 radio consumes 1.5 mA when in *idle* mode. The radio transition from *idle* mode to *Rx* mode takes about 100 μs drawing ~8 mA of current. Carrier sensing takes ~80 μs drawing 19 mA of current.

We also measure the average radio listen time for a packet reception to evaluate the benefit of including data frames in the advertisement beacons. The time is averaged over 1000 periodically received packets by a listening node from a corresponding number of transmitted beacons by a transmitter. Each advertisement beacon contains 20 packets, and each packet is 128 bytes long. The packets are transmitted at a baud rate of 250Kbps. Compared against the average theoretical radio listen time for LPL-like beacons employed by U-Connect, the WiFlock beacon structure reduces the time for which the radio must track the beacon before successfully receiving data contents. Average beacon decoding time is about 7.4ms, a 5X improvement over LPL.

5.2 Group Discovery Latencies

To evaluate group discovery and formation latency, we setup 50 nodes with listening duty cycle of 0.2% and listen slot durations of 80μs. All the nodes were located within a single hop from each other. Of these 50 nodes, 20 nodes were connected to a PC via USB. The PC can turn on and off a group of selected size from the twenty nodes (the remaining 30 nodes were manually switched for the 50 node experiment). Each active node logs updates to its group table to the PC, where they are time stamped. Node groups of different sizes are turned on with random offsets over a 40 second period. The one beacon period interval for nodes joining is to ensure nodes are not synchronized in the beginning. The experiment provides a snapshot of every node's group table at any given time. We repeat the experiment 20 times for each group size.

Table 2 shows average and standard deviation for group formation latencies, i.e. the time for the group information to be propagated to every node in the group, with various group sizes. Since, every member of the group must discover every other member of the group, this corresponds to the worst latency observed by any node during an instance of the experiment.

Figure 15 shows the average rate of discovery, i.e. time nodes to discover a fraction of the entire group.

Group Size	Avg. Latency	Std. Dev.
5	113s	35s
10	114s	30s
15	119s	12s
20	120s	22s
50	168s	16s

Table 2: Group formation latencies for groups of varying sizes, with nodes operating at 0.2% duty cycle.

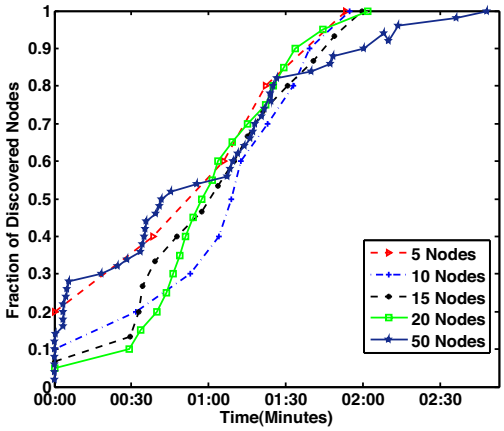

Figure 15: The average time for a number of nodes, operating at 0.2% duty cycle, to discover a fraction of the group. The evenly spaced beaconing reduces packet collisions and scales well with increased group sizes.

From Figure 15 we can see that the average time it takes for a set of nodes to discover and form a group stays rather constant when the group size increases. For less than 20 nodes, at 0.2% duty cycle nodes consistently discovered the entire group within an average of 120 seconds. Even with 50 nodes, the average discovery latency is less than 3 minutes. The results show that the synchronized listening and evenly spaced beaconing allows the protocol to scale to a large number of nodes without affecting latencies.

Neighbor discovery protocols such as U-Connect and Disco provide theoretical worst-case bounds on discovery latencies for a pair of nodes. While they support discovery in clusters of nodes, they do not optimize for or address problems of group maintenance such as absence of all-to-all connectivity or the high probability of collisions as group sizes become large. In contrast, WiFlock goes beyond neighbor discovery to provide a service for discovery and maintenance of groups of mobile wireless nodes. It uses synchronization and sharing of neighbor tables to make continuous neighbor discovery efficient and scalable to large clusters of nodes.

Nevertheless, for pair-wise discovery, WiFlock provides a 3X improvement in theoretical worst-case latency over U-Connect due to shorter carrier sense times. For example, at 0.2% duty cycle the theoretical worst-case latency for a pair of U-Connect nodes to discover each other is 250 seconds, while that for WiFlock nodes is 80 seconds. U-connect experimental results suggest the latencies may exceed the theoretical maximum when the number of nodes in the same collision domain is high. In practice, in a large group of 50 nodes with collisions and varying network topologies, WiFlock achieves performance well within the worst-case for U-Connect.

5.3 Group Maintenance Latencies

An already established group can collaborate on their beaconing to improve the efficiency for the group to react to changes such as discovering new nodes.

A new coming node, depending on its distance from the

group and location, may have connectivity with only a fraction of the established group. In static networks, the network topology may be known and can be used by the network to predict the connectivity of an incoming node[3]. However, no such link quality assumptions can be made in flocking sensor networks where all nodes are mobile. Therefore, we empirically evaluate the average case and worst case latencies for a node approaching a group when only a fraction of the group nodes are in communication range of the new node.

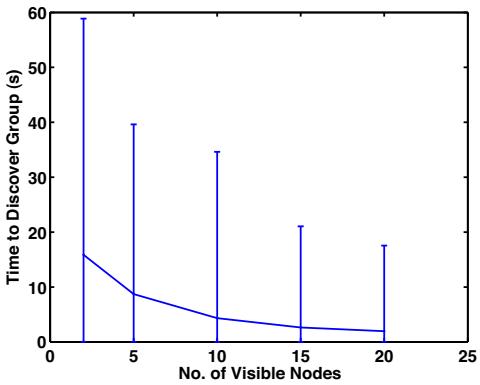

Figure 16: Average discovery latencies when a mobile node joins an established group of 20 nodes, when only a fraction of the group has connectivity to the incoming node. The error bars show the maximum and minimum latency observed.

Figure 16, shows the average latency for an incoming node to discover an established 20 node group, when it has connectivity with only a fraction of group nodes. The experimental setup is similar to the one used in section 5.2. A group of 20 nodes is given enough time to to form an established group. A new node is then turned on and the radio connectivity is simulated by a ID filter. That is, the exact node id's that the new node is "supposed" to hear is provided to the node by the PC before each experiment. For example, to simulate a situation that the new coming node can hear n nodes in the group, a random n id's is picked and sent to the new coming node. The time for the new node to discover the group is logged. We repeat 1000 runs for each n. Obviously, greater exposure to the group shortens the latency of new discovery.

6. RELATED WORK

Continuous neighborhood management for devices not subject to energy constraints can be attained simply by periodic transmission of advertisement beacons. Neighboring devices with always-on receivers can periodically update their neighborhood tables to the presence of co-located devices.

If an out-of-band time synchronization mechanism exists such as GPS[19], FM transmitters [11], neighbor discovery can be achieved with higher efficiency. However, for applications such as in asset tracking, where *tags* are constrained due to both cost and energy, such out-of-band mechanism lose their appeal.

For energy constrained but static networks neighborhood management can be attained as two-phase process, where larger latency and energy consumption may be tolerated for initial neighbor discovery while switching to a more energy efficient common mode of maintaining neighborhood information. In some cases, for mobile nodes the initial neighbor discovery phase can be periodically repeated. Its relatively higher latency and energy consumption may be acceptable if node lifetime requirements are relatively less strict. As an illustration, neighborhood maintenance in mobile ad hoc networks is generally a function of the routing protocol such as AODV [8]. Nodes usually operate in the always listening mode. When a node needs to communicate with another node, a RREQ (route request) packet is flooded to other nodes. This causes high control traffic overhead and delay when initiating communication for the first time. An established route maintenance procedure is subsequently adopted using the RERR (route error) message to notify other nodes of the loss of neighbors. In mobile networks with highly transient neighborhoods such protocols have a high energy cost, having to constantly initiate route requests.

Quorum-based protocols [16] or variants there-of such as BMAC [9], SMAC [18] provide useful primitives for communication between a pair of devices. In Quorum, a sequence of beacon intervals is divided into sets starting from the first interval such the each continuous n^2 is called a group, where n is a global parameter. In each group, intervals are arranged in a $n \times n$ array in a row-major manner. Nodes arbitrarily pick a row and a column, and any pair of nodes will have two intersections allowing them to discover each other. While these protocols address the basic primitive behind discovery they do not concern themselves with the very low-energy operation as desired by our motivating application. Also, for a given duty cycle their implementations tend to provide order of magnitude higher latencies than our proposed solution. As they are essentially based on nodes beaconing at random times, they do not address network effects like collisions which intensify in large groups. This is also the case for "birthday protocols" for static ad-hoc networks, proposed in [7], which also have long tails on discovery probabilities.

Dutta et. al. propose Disco, a low-power asynchronous neighbor discovery protocol [4]. The authors address the rendezvous of two mobile nodes with support for nodes operating at dissimilar duty cycles. Nodes select a pair of primes such that the sum of their reciprocals is equal to their desired duty cycle. Nodes wakeup at multiples of the prime numbers. The worst case latency is the product of the minimum of the primes picked by the nodes in operation. Disco also analyzes discovery latencies in clusters and suggests extending the work by using gossip. We adopt some of these ideas in our protocol. However, Disco requires large wake-up slots in the order of milliseconds due to the need for bidirectional communication and susceptibility to clock drift. This results in large latencies for our desired duty cycle of operation.

Kandhalu et al. propose U-Connect, a protocol for asynchronous neighbor discovery for both symmetric as well as asymmetric duty cycle nodes [6]. They establish that U-Connect is an 1.5-approximation algorithm for the symmetric asynchronous neighbor discovery problem, whereas existing protocols like Quorum and Disco are 2-approximation algorithms. We use a modification of U-Connect as the basic neighbor discovery primitive. The flip side to the U-Connect protocol is the creation of asymmetric links where only one node can hear the other depending on the phase of its beacon. To discover each other, nodes must perform a pair-wise id exchange every time a node hears another node, creating message traffic in $O(n^2)$. This must be frequently repeated to maintain a continuously up-to-date neighborhood group. Disco also uses a pair-wise message exchange for discovery making it less attractive for scalable, low-latency and energy efficient neighborhood management.

Continuous neighbor discovery is mentioned in [3], in the context of maintaining node connectivity information in face of disruptions in wireless communication, synchronization or

change in transmission power for established static networks. The authors propose an algorithm for continuous neighbor discovery after an initial neighborhood has been established using a broadcast SYNC message which is heard by all nodes. This assumption of all nodes being either synchronized or awake at the same time when initially deployed, is not applicable to a network of mobile nodes with transient neighborhoods.

7. CONCLUSION

This paper presents WiFlock, a collaborative group formation, and group maintenance protocol for mobile sensor networks. It is a fundamental yet challenging building block for applications such as asset tracking, habitat monitoring, and smart environments. In this paper, we show that the neighbor discovery process and the group maintenance process must be designed together to achieve better energy-efficiency, reactiveness, and scalability. Since a sensor node can spend most of its time alone, optimizing beaconing and listening periods and durations is critical. Through analysis and implementation, we show how to achieve ultra low duty cycles ($\sim$0.2%) without sacrificing discovery latency.

The one-way discovery mechanism, although optimizing energy efficiency for neighbor discovery, brings challenges in group membership propagation and maintenance. We designed a synchronized listening and evenly spaced transmitting (SLEST) mechanism to archive group-level coordination by reusing neighbor discovery beacons. Our experiments show that the protocol scales well with network density. Several extensions can be built on top of WiFlock and possibly improve its performance.

Accommodating Large Groups. WiFlock as presented in this paper assumes that all the group information can fit into a single frame, which in our case has a maximum size of 128 bytes, limiting the group size to a about 50 nodes (assuming 4 bytes information per node plus frame header). However, WiFlock can accommodate larger group sizes by splitting the group table across multiple frames and multiple beacons. Since correct operation of the synchronized listening and evenly spaced transmissions only depend on the smallest node ID that a node has received so far, it not necessary to repeat the smallest node ID across multiple frames and beacons. However, to prevent premature timeout of node information, a larger initial node TTL value needs to be used with large groups. For large groups spanning multiple beacons, the mapping of nodes rank to the beacon slot should be done with proper attention to the number of slots within a given beacon period.

Dedicated RSSI Measurements. Currently, we use the RSSI value reported by the AGC module of the radio to detect the presence of an on going beacon. This RSSI value is a byproduct of the AGC module, where the primary function is to use the RSSI value to maintain a constant amplified signal inside the radio. Because of this, the carrier detection time is limited by how the AGC module is designed. In contrast, a dedicated carrier sensing module optimized for faster carrier detection might be able to reduce the wakeup duration, hence the duty cycle, below what is reported in this paper.

8. ACKNOWLEDGEMENTS

Special thanks to our shepherd Dr. Prabal Dutta and the anonymous reviewers for their constructive comments.

9. REFERENCES

[1] N. Banerjee, M. D. Corner, D. Towsley, and B. N. Levine. Relays, Base Stations, and Meshes: Enhancing Mobile Networks with Infrastructure. In *MobiCom '08*. ACM.

[2] 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*. ACM.

[3] R. Cohen and B. Kapchits. Continuous Neighbor Discovery in Asynchronous Sensor Networks. *IEEE/ACM Transactions on Networking*, (99):1–1, 2010.

[4] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *SenSys '08*. ACM, 2008.

[5] Epson Toyocom Corporation. SG-3030 Crystal Oscillator. `http://www.eea.epson.com/portal/pls/portal/docs/1/1219458.PDF`.

[6] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *IPSN '10*. ACM, 2010.

[7] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *MobiHoc '01*. ACM, 2001.

[8] C. Perkins, E. Belding-Royer, and S. Das. RFC3561: Ad hoc On-Demand Distance Vector (AODV) Routing. *Internet RFCs*, 2003.

[9] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04*, pages 95–107. ACM, 2004.

[10] A. Purohit and P. Zhang. Demo: SensorFly - A Controlled-mobile Aerial Sensor Network. In *SenSys '09*, pages 327–328. ACM, 2009.

[11] A. Rowe, R. Mangharam, and R. Rajkumar. Rt-link: A global time-synchronized link protocol for sensor networks. *Ad Hoc Netw.*, 6(8):1201–1220, 2008.

[12] A. A. Syed, W. Ye, and J. Heidemann. T-Lohi: A new class of MAC protocols for underwater acoustic sensor networks. In *INFOCOM 2008*, Phoenix, Arizona, USA, April 2008. IEEE.

[13] Texas Instruments Corporation. CC2500 Low-Cost Low-Power 2.4 GHz RF Transceiver. `http://focus.ti.com/lit/ds/swrs040c/swrs040c.pdf`.

[14] Texas Instruments Corporation. Design Note DN505 : RSSI Interpretation and Timing. `http://focus.ti.com/lit/an/swra114d/swra114d.pdf`.

[15] Texas Instruments Corporation. eZ430-RF2500 Development Tool User's Guide. `http://focus.ti.com/lit/ug/slau227e/slau227e.pdf`.

[16] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks. *Comput. Netw.*, 43(3):317–337, 2003.

[17] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *SenSys '05*, pages 142–153. ACM, 2005.

[18] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. *INFOCOM 2002*, pages 1567–1576, 2002.

[19] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *SenSys '04*, pages 227–238. ACM, 2004.