On Designing Neighbor Discovery Protocols: A Code-Based Approach

Tong Meng Fan Wu Guihai Chen
Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
mengtong@sjtu.edu.cn; {fwu, gchen}@cs.sjtu.edu.cn

Abstract—**In mobile wireless networks, the emerging proximity-based applications have led to needs for highly effective and energy-efficient neighbor discovery protocols. However, existing works cannot realize the optimal worst-case latency in symmetric case, and their performances with asymmetric duty cycles can still be improved. In this work, we investigate asynchronous neighbor discovery through a code-based approach, including the symmetric and asymmetric cases. We derive the tight worst-case latency bound in the case of symmetric duty cycle. We design a novel class of symmetric patterns called Diff-Codes, which is optimal when the Diff-Code can be extended from a perfect difference set. We further consider the asymmetric case, and design ADiff-Codes. To evaluate (A)Diff-Codes, we conduct both simulations and testbed experiments. Both simulation and experiment results show that (A)Diff-Codes significantly outperform existing neighbor discovery protocols in both the median case and worst-case. Specifically, in symmetric case, the maximum worst-case improvement is up to 50%; in both symmetric and asymmetric cases, the median case gain is as high as 30%.**

I. Introduction

Nowadays, the transfer of data between neighboring nodes in mobile wireless networks has been increasingly indispensible owing to the rapid growth of diverse demands in people's everyday life. For instance, a college student may want to discuss a math problem with other students in the library using his/her tablet; a video game fan is likely to have a car race on the smartphone with other people in a Starbucks coffee shop. These motivate the appearance of proximity-based applications. Although central servers can be employed, proximity-based applications' potential can be better exploited providing the ability of discovering nearby mobile devices in one's wireless communication vicinity due to four reasons. First, users can enjoy the convenience of local neighbor discovery at any time, while the centralized service may be unavailable duo to unexpected reasons. Second, a single neighbor discovery protocol can benefit various applications, by providing more flexibility than the centralized approach. Third, communications between a central server and different mobile nodes may induce problems, such as excessive transmission overheads, congestion, and unexpected reaction delay. Last but not least, searching for nearby mobile devices locally is totally free of charge.

Therefore, a distributed neighbor discovery protocol for mobile wireless networks is highly needed in practice. Generally, there are three challenges in designing such a neighbor discovery protocol.

- The first one is energy efficiency. Due to limited battery power, a mobile node can only periodically turn on its wireless interface with a certain duty cycle. In some applications, nodes may agree on the same duty cycle for fast neighbor discovery (symmetric case). However, mobile nodes may need to adopt different duty cycles independently, according to their remaining battery power levels (asymmetric case). Therefore, both the symmetric and asymmetric neighbor discovery should be considered.
- The second challenge is effectiveness, *i.e.*, the neighbor discovery protocol should not only guarantee successful discovery between neighboring nodes, but also realize a short latency at the same time. On one hand, the probabilistic approach (*e.g.*, Birthday Protocol [13]) in static sensor networks does not meet this requirement, because it fails to provide a worst-case discovery latency bound, and thus leads to confusion between discovery failure and non-existence of neighbors. On the other hand, the discovery latency should be short enough, so that the users will not lose patience before finding a neighbor, and the interval when two mobile nodes are within each other's communication range can be captured.
- In an ideal case, neighboring nodes can discover each other immediately if they turn to awake simultaneously upon synchronized clocks. Without a central server, the synchronization can be achieved through GPS [14]. Nevertheless, it is too energy consuming for mobile devices. Thus, how to deal with asynchronization is the third challenge to the design of a neighbor discovery protocol.

We consider asynchronous deterministic neighbor discovery, aiming at high energy efficiency as well as low discovery latency. Most existing neighbor discovery protocols (*e.g.*, Disco [6], U-Connect [9]) cannot realize the optimal worst-case latency provided in [20]. Furthermore, although Searchlight [1] is approximate to the optimum as in [20] with symmetric

This work was supported in part by the State Key Development Program for Basic Research of China (Grant No. 2014CB340303, 2012CB316201), in part by China NSF grant 61272443, 61133006, 61073152, and in part by Shanghai Science and Technology fund 12PJ1404900 and 12ZR1414900. The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government.

F. Wu is the corresponding author.

duty cycle, its performance in the asymmetric case still need to be improved.

In this work, through an in-depth study on the problem of asynchronous neighbor discovery, we derive a tighter lower bound of optimal worst-case latency (or duty cycle). Then, we adopt a code-based formulation of the neighbor discovery problem, and design Diff-Codes for the symmetric case, which is optimal when the Diff-Code can be extended from a perfect difference set. Furthermore, by considering the connection between awake periods of two nodes, we extend Diff-Codes to ADiff-Codes to deal with asymmetric neighbor discovery.

The detailed contributions of this work are listed as follows.

- We demonstrate the feasibility conditions of an asynchronous neighbor discovery protocol, from the perspective of 0-1 code.
- We formulate the problem of asynchronous neighbor discovery with symmetric duty cycle mathematically. By the formulation, we derive the lower bound for optimal worst-case latency, and design Diff-Codes. We show that a Diff-Code is optimal when it can be extended from a perfect difference set.
- We further investigate the feasibility conditions with asymmetric duty cycles, and design ADiff-Codes, which can be constructed as long as two pattern codes' lengths are relatively prime.
- To evaluate the performance of our designs, we not only conduct comprehensive simulations, but also prototype them using USRP-N210 testbed. Evaluation results show that (A)Diff-Codes significantly reduce the discovery latency in both the median case and worst-case. Specifically, in symmetric case, the maximum improvement is up to 50%; in both symmetric and asymmetric cases, the median case gain is as high as 30%; and ADiff-Codes outperform state-of-art protocols in more than 95% situations.

The rest of the paper is organized as below. In Section II, we briefly introduce the related works. In Section III, we explain the system model. The feasibility conditions for symmetric neighbor discovery are presented in Section IV. Then in Section V, we propose the construction of Diff-Codes. In Section VI, we extend to the asymmetric case, and design ADiff-Codes. In Section VII, we provide the results of simulations and testbed experiments. Finally, the paper is concluded in Section VIII.

II. Related Works

The problem of neighbor discovery was initially studied in static wireless sensor networks. Recently, it has also been investigated in mobile wireless networks. Existing neighbor discovery protocols generally fall into two categories, including probabilistic protocols and deterministic protocols.

A. Probabilistic Protocols

McGlynn *et al.* [13] introduced a family of "birthday protocols", which forms the foundation of most probabilistic neighbor discovery protocols. In birthday protocols, time is slotted, and each node probabilistically determines the state for each slot from transmitting, listening, and energy-saving, independently. A node makes itself known by its neighbors when it is the only transmitting node in its vicinity in a slot. Based on [13], Keshavarzian and Uysal-Biyikoglu [10] proposed a random protocol for link assessment. Vasudevan *et al.* [18] reduced the probabilistic algorithm for neighbor discovery to the Coupon Collector's Problem. In [11], Khalili *et al.* further realized the mechanism of channel status detection, and designed algorithms providing feedback of reception. Additionally, Vasudevan *et al.* [17] discussed probabilistic neighbor discovery with directional antennas. Later, Zeng *et al.* [19] extended the solution to multipacket reception networks.

Birthday protocols support both symmetric and asymmetric cases, and have satisfying median case performance. However, because of the lack of worst-case latency bound, these probabilistic protocols inevitably incur the problem of long tail. The discovery latency may be arbitrarily long, which makes the probabilistic protocols unsuitable for mobile wireless networks. Therefore, deterministic approaches are usually adopted for neighbor discovery by mobile devices.

B. Deterministic Protocols

A deterministic protocol establishes a pattern scheduling the periodical operations of each node. A code-based protocol is presented in [10] utilizing constant-weight codes [3], [5], but it assumes synchronization among nodes. Moreover, Zheng *et al.* [20] applied optimal block designs in the case of symmetric duty cycle. The authors concluded that their approach reduces to an NP-complete minimum vertex cover problem in asymmetric case. Whereas we prove that the bound in [20] can be further lowered. Besides, our designs fit for both symmetric and asymmetric cases with low complexity.

In a class of quorum-based protocols [12], [16], a cycle contains m^2 consecutive slots, where m is a global parameter. A node is either awake or sleeping in a slot. Both transmitting and listening happen during awake slots, so that two neighboring nodes discover each other when they are both awake. These m^2 intervals are arranged as an $m \times m$ matrix. Each node picks a row and a column of slots, during which the node stays awake. Such a protocol ensures that two different nodes will have exactly two intersecting awake slots during each cycle. However, quorum-based protocols are normally restricted to the symmetric case. Although [12] allows the existence of two different duty cycles, its application is still limited.

Another important type of deterministic protocols that can handle both the symmetric and asymmetric cases is the prime-based protocol (*e.g.*, Disco [6] and U-Connect [9]). In Disco, each node chooses a pair of prime numbers (p_1, p_2), and turns awake only at multiples of p_1 and p_2. U-Connect uses only one prime number p. Each node wakes up at p's multiples, as well as $\frac{p+1}{2}$ slots every p^2 slots. Although prime-based protocols improve the worst-case latency bound, they underperform birthday protocols in the median case. In response to that, Bakht *et al.* [1] leveraged the regular relationship between the patterns of two nodes, and designed Searchlight. Compared with previous protocols, Searchlight [1] performs much better in symmetric case, but it needs to be improved in the case of asymmetric duty cycles. By contrast, our designs in this work are superior to existing neighbor discovery protocols regardless of duty cycle symmetry.

III. System Model

We focus on deterministic asynchronous neighbor discovery for mobile wireless networks. Similar as existing works (*e.g.*, [1], [6], [9]), we assume that time is divided into equal-size slots. Owing to restricted energy budget, each node (*i.e.*, a mobile device) performs duty-cycled operations. That is, it sleeps during most slots, while turning awake during a few remaining slots, which are called active slots. To be specific, in an active slot, a node transmits beacons at the beginning and the end, respectively, and listens for other nodes' transmissions in between. While in a sleeping slot, a node does not send or receive, and consumes negligible energy. Thus, two neighboring nodes can discover each other when their active slots overlaps. Moreover, the neighbor discovery problem involves two cases: the symmetric case, where all the nodes have the same duty cycle, and the asymmetric case, where different duty cycles are adopted.

In deterministic neighbor discovery, there is an established active-sleep pattern scheduling a node to alternate its state periodically between active and sleeping. We formulate the active-sleep pattern as a 0-1 code. A pattern code $C = c_0 c_1 \cdots c_{n-1}$ determines an active-sleep pattern containing n slots in a cycle. Specifically, bit c_i corresponds to the slot whose index is i,[1] *i.e.*, slot i is an active slot if $c_i = 1$; otherwise, $c_i = 0$. The weight of code C with length n equals the number of active slots in a cycle. This is to say, the duty cycle is decided by the length and the weight of pattern code C.

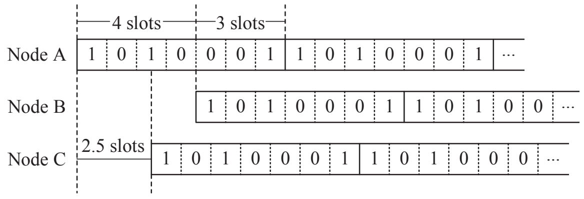


Fig. 1. Example: the slot offset between two nodes

In asynchronous neighbor discovery, there is an offset between a pair of neighboring nodes' active-sleep patterns. In the case of symmetric duty cycle, we define slot offset d_{AB} between node A and node B as the interval in unit of slot between slot 0's in their common pattern. The value of d_{AB} is not necessarily an integer. Assume code C is of length n. Then a slot offset d_{AB} is equivalent to $d_{BA} = n - d_{AB}$. For clarity, we take the slot offset to be $min(d_{AB}, d_{BA})$, which is bounded by $\frac{n}{2}$. Referring to an example in Fig. 1, given symmetric pattern code "1010001", slot offset d_{AB} is 3, while d_{AC} equals 2.5.

[1]All the slot indices in this paper are taken module n. We omit "mod n" for concise representation.

Moreover, we define the cyclic shift, $C^{(j)}$, of pattern code C, to compensate for slot offsets between neighboring nodes, where j is in unit of slot. For an integer j, $C^{(j)}$ is calculated by cyclically shifting C right by j bits. For example, in Fig. 2, where active slots are in gray, cyclic right shift of 10100101 by 3 is 10110100.

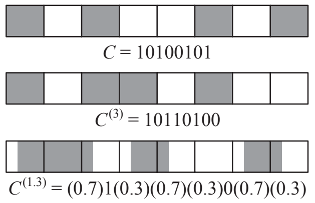


Fig. 2. Examples on Cyclic Shifts of a Pattern Code

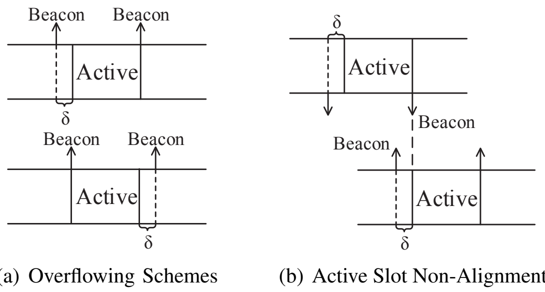
Additionally, we relax the value of $c_i^{(j)}$ to a closed interval $[0, 1]$, to address the case of partial shift. Thus, $c_i^{(j)}$ denotes the active proportion of the node in slot i regarding the shifted code $C^{(j)}$. After the cyclic shift by a non-integer j, the value of $c_i^{(j)}$ is determined by both $c_{i-\lceil j \rceil}^{(0)}$ and $c_{i-\lfloor j \rfloor}^{(0)}$. For instance, in Fig. 2, the result of cyclic right shift of C by 1.3 slots is $(0.7)1(0.3)(0.7)(0.3)0(0.7)(0.3)$. In the following, we define two operators denoted by $\odot$ and $\oplus$ in (1) and (2), for the definition convenience of partial cyclic shift as Equation (3).

$$a \odot C = (a \cdot c_i)_{i=0,1,\cdots,n-1}\,, \tag{1}$$

$$C \oplus C' = (min(c_i + c'_i, 1))_{i=0,1,\cdots,n-1}\,, \tag{2}$$

$$C^{(j)} = \left[(1 + \lfloor j \rfloor - j) \odot C^{(\lfloor j \rfloor)}\right] \oplus \left[(j - \lfloor j \rfloor) \odot C^{(\lceil j \rceil)}\right]. \tag{3}$$

All of our designs are based on non-alignment of active slot boundaries. However, although quite rare, perfect slot alignment cannot be ignored. Therefore, like [1], we implement overflowed active slots to realize active slot non-alignment. As depicted in Fig. 3(a), an active slot is made either to start a little bit earlier (which we adopt in this work) or to end later. As a result, the width of an active slot is increased by δ, while the preceding (or succeeding) slot is shortened correspondingly. With such an overflowing scheme, adjacent active slots of two nodes can overlap even in the case of perfect slot alignment (as in Fig. 3(b)).



(a) Overflowing Schemes (b) Active Slot Non-Alignment

Fig. 3. Overflowed Active Slots

What's more, utilizing the operator $\ominus$ as in (4), we define $C(j)$ in Equation (5) to indicate the actual active proportion of the node in each slot after cyclic shift by j, integrating the overflowed active slots.

$$a \ominus b = max(a-b,0), \tag{4}$$

$$C(j) = C^{(j)} \oplus \left[(\delta \ominus (j - \lfloor j \rfloor)) \odot C^{(\lfloor j \rfloor -1)}\right] \oplus \left[\delta \odot C^{(\lceil j \rceil -1)}\right]. \tag{5}$$

IV. Pattern Feasibility Validation in Symmetric Neighbor Discovery

In this section, we demonstrate the feasibility conditions for the active-sleep pattern code in the case of symmetric duty cycle.

If a pair of nodes has a slot offset d, slot 0 of one node will coincide in time with both slot $\lfloor d \rfloor$ and $\lceil d \rceil$ of the other node. It leads to the following lemma.

Lemma 1. *Two neighboring nodes use the same pattern code C, and their slot offset is d. They can discover each other if and only if the following condition is satisfied,*

$$\exists i \in N, \quad c_i(0) + c_i(d) > 1. \tag{6}$$

According to Lemma 1, there is the following corollary, which is the basis of symmetric pattern feasibility condition.

Corollary 1. *A pattern code C of length n schedules the operations of two neighboring nodes. If there exists an integer i and an integer $j \le \lfloor \frac{n}{2} \rfloor$ such that $c_i^{(0)} = c_i^{(j)} = 1$, the two nodes can discover each other as long as their slot offset d takes its value from the closed interval $[j-1, j+1]$.*

Proof: According to the given conditions, $c_i(0) + c_i(j) = 2$. In addition, with overflowed active slots, the following relation holds,

$$d \in [j-1, j+1] \Rightarrow c_i(0) + c_i(d) \ge 1 + \delta.$$

The above situation is also demonstrated in Fig. 4. By Lemma 1, the pattern is feasible for any $d \in [j-1, j+1]$. □

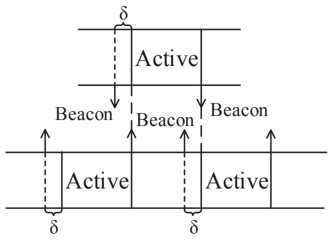


Fig. 4. Two Aligned Active Slots Cover a Range of Slot Offsets

Furthermore, when random slot offset is considered, works on quorum-systems (*e.g.*, [8]) focus on such feasibility that a pattern should suffice the condition in Corollary 1 for any integer i. However, when active slots are overflowed, the constraint for a feasible pattern can be relaxed.

Theorem 1. *A feasible pattern code C for symmetric neighbor discovery should satisfy that,*

$$\exists i, \; (c_i^{(0)} + c_i^{(j)} = 2) \vee (c_i^{(0)} + c_i^{(j+1)} = 2) = TRUE, \tag{7}$$

where $j \in \{0, 1, \cdots, \lfloor \frac{n}{2} \rfloor\}$.

Proof: Assume pattern code C is feasible, but there exists an integer slot offset d dissatisfying (7). Then code C should be feasible under the slot offset of $d - \frac{1}{2}$. By lemma 1, there should be an integer i satisfying the inequality,

$$c_i(0) + \frac{1}{2}(c_i(d) + c_{i+1}(d)) > 1.$$

Clearly, $c_i(0)$ equals either 1 or δ. The case of $c_i(0) = \delta$ leads to $c_{i+1}^{(0)} = c_{i+1}^{(d)} = 1$, which indicates that d satisfies (7). On the other hand, when $c_i(0) = 1$, then either $c_i(d)$ or $c_{i+1}(d)$ equals 1. If so, d satisfies condition (7), as well. Therefore, the above assumption always produces contradiction, which proves the correctness of the theorem. □

Specifically, an algorithm for feasibility validation can be built from Theorem 1. It examines all the possible integer slot offsets with condition (7), yielding the complexity of $O(n^2)$.

V. Symmetric Pattern Code Construction

In this section, we formulate the design of symmetric active-sleep patterns into a code construction problem, aiming to minimize the code weight for a given code length. We derive the lower bound for the formulation, which breaks through the generally accepted result presented in [20]. We also propose a novel class of active-sleep patterns called Diff-Codes, and analyze its theoretical performance. The construction of Diff-Codes takes advantage of perfect difference sets [2], [15], and guarantees optimality when the Diff-Code is extended from a perfect difference set. Last, for comprehensiveness, we provide a heuristic algorithm for seeking the Diff-Code with a target duty cycle.

A. Problem Formulation

The definition of the code construction problem is as follows: for a given n, construct a 0-1 code C of length n with as few 1-bits as possible, while ensuring that C is feasible for symmetric neighbor discovery. According to condition (7) in Theorem 1, the mathematical formulation is as below.
Objective:

$$Minimize \qquad w = \sum_{i=0}^{n-1} c_i, \tag{8}$$

Subject to:

$$\sum_{i=0}^{n-1} c_i(c_{i+j} + c_{i+j+1}) \ge 1, \quad \forall j \in \{0, 1, \cdots, \left\lfloor \frac{n}{2} \right\rfloor\}, \tag{9}$$

$$c_i \in \{0,1\}, \quad \forall i \in \{0, 1, \cdots, n-1\}. \tag{10}$$

The lower bound of the above primal problem can be calculated via the Lagrange dual problem [4] by relaxing constraint (10). We first define the Lagrangian L,

$$\begin{aligned} L(\boldsymbol{c}, \boldsymbol{\lambda}, \boldsymbol{\omega}, \boldsymbol{\nu}) \; &= \; (\mathbf{1} + \boldsymbol{\omega} - \boldsymbol{\nu})^T \boldsymbol{c} + \mathbf{1}^T \boldsymbol{\lambda} - \mathbf{1}^T \boldsymbol{\omega} \\ &\quad - \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j \left[\boldsymbol{c}^T (A_j + A_{j+1}) \boldsymbol{c}\right], \end{aligned} \tag{11}$$

where the element of matrix $A_j = (a_{kl})_{n\times n}$ is defined as below.[2]

$$a_{kl}|_{A_j} = \begin{cases} 1, & \text{if } k = l + j \text{ or } k = l + j - n; \\ 0, & \text{otherwise.} \end{cases} \quad (12)$$

Then Theorem 2 gives the optimal solutions to both the above Lagrange dual problem and the relaxed primal problem. Its proof implements the KKT optimality conditions (referring to [4] for the details).

Theorem 2. *An optimal solution to the relaxed problem of formulation* (8) *is* $\boldsymbol{c}^* = (\frac{1}{\sqrt{2n}})_{n\times 1}$, *corresponding to the objective function value of* $\sqrt{\frac{n}{2}}$; *an optimal solution to the dual problem satisfies that* $\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j^* = \frac{\sqrt{2n}}{4}$ *and* $\omega_i^* = \nu_i^* = 0$.

Proof: First of all, we prove the strong duality. On one hand, when $c_i^* = \frac{1}{\sqrt{2n}}$, both constraint (9) and (10) are satisfied. Hence, $\boldsymbol{c}^*$ is feasible for the primal problem and the objective can be calculated to be $\sqrt{\frac{n}{2}}$. On the other hand, the non-negative solution $(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*)$ is feasible for the Lagrange dual problem. Specifically, $\boldsymbol{c}^*$ leads to equality in constraint (9), which yields the following equation,

$$\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j^* \left[1 - \sum_{i=0}^{n-1} c_i^* \cdot (c_{i+j}^* + c_{i+j+1}^*) \right] = 0. \quad (13)$$

Next, with $(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*)$, the differential of Lagrangian L is,

$$\begin{aligned} & \nabla_{\boldsymbol{c}} L(\boldsymbol{c}, \boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*) \\ = \ & (\boldsymbol{1} + \boldsymbol{\omega}^* - \boldsymbol{\nu}^*) - \\ & \qquad \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j^* (A_j + A_{n-j} + A_{j+1} + A_{n-j-1})^T \boldsymbol{c} \\ = \ & \boldsymbol{1} - (\frac{4 \sum \lambda_j}{\sqrt{2n}})_{n\times 1} = \boldsymbol{0}, \end{aligned} \quad (14)$$

which determines the objective $g(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*) = \sqrt{\frac{n}{2}}$. Therefore, with $\boldsymbol{c}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*)$, the solution to the relaxed primal problem equals to that of Lagrange dual problem. The strong duality holds.

Second, the feasibility of $\boldsymbol{c}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*)$, and equation (13) and (14) together satisfy the KKT conditions. That verifies the optimality of $\boldsymbol{c}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*)$. □

Theorem 2 indicates that a symmetric active-sleep pattern with a cycle length of n slots should have at least $\sqrt{\frac{n}{2}}$ active slots each cycle. This lower bound is tighter than the bound of $\sqrt{n}$ provided by Zheng *et al.* [20], because we exploit the power of active slot non-alignment in the asynchronous case. Consequently, compared with the active-sleep patterns in [20], which is identical with perfect difference sets, we achieve much better patterns.

B. Asymptotically Optimal Pattern via Perfect Difference Set

Referring to the pattern feasibility in Section IV, and the definition below, an $(n, w, 1)$-perfect difference set [2], [15] already corresponds to a feasible symmetric pattern code of length n and weight w.[3]

Definition 1. *An* (n, w, λ)*-difference set contains w elements. It is a subset of* $Z_n = \{0, 1, \cdots, n-1\}$, *and each* $d \in Z_n \backslash \{0\}$ *appears exactly* λ *times as the difference of two distinct elements from it under module n. Specifically, a difference set with* $\lambda = 1$ *is called a perfect difference set.*

However, being a perfect difference set is a stricter constraint than condition (7) in Theorem 1. For example, a pattern code $C = 10100010000000$ can be verified to be feasible, although it does not correspond to a difference set. To this end, we propose to double the length of a perfect difference set while maintaining its weight. The details can be described as below: an active slot is extended to two consecutive slots including one active slot followed by another sleeping slot; a sleeping slot is extended to two successive sleeping slots.

Assume pattern code C' is extended from C whose $\mathcal{D}(C^{(0)})$ is a perfect different set. Then C' can be verified to satisfy Theorem 1. In fact, such a pattern code C' is the best integer solution to our formulated problem, which is meanwhile approximate to the lower bound given by Theorem 2.

Theorem 3. *Pattern code* C' *of length* $2n$ *and weight w, which is resulted from extending a perfect difference set with length n and weight w as shown above, is the optimal integer solution to the code construction problem under code length* $2n$.

Proof: Because C' is extended from a perfect difference set, its length $2n$ is even, and satisfies that $w(w-1)+1 = n$. Apparently, n is odd.

For pattern code $C'^{(0)}$, the total number of distinct active slot pairs is $\frac{1}{2}w(w-1)$. By Theorem 1, for any pattern code of length $2n$, the weight w should satisfy the inequality,

$$\frac{1}{2} \cdot w(w-1) \geq \left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2} - \frac{1}{2}. \quad (15)$$

Considering that the length and weight of C' lead to equality in (15), its optimality is verified. □

Nevertheless, a perfect difference set requires specific value of its length and weight [7]. When $p^s \leq 1600$, where p is a prime number and s is a positive integer, there only exists such form of perfect difference sets that $w = p^s + 1$. Thus, as w approaches 1600, the worst-case discovery latency (*i.e.*, the code length) of the extended pattern code C' will be bounded by $2n$, with a magnitude of as high as 10^6 slots. That is unbearable for realistic applications. Hence a practical symmetric active-sleep pattern should be based on such a perfect difference set that $w = p^s + 1$.

C. Diff-Code Construction

Although doubling the length of a perfect difference set can generate the optimal schedule, it's only suitable for specific

[2]The notation "mod n" is omitted. The same with the following corollaries.

[3]An active-sleep pattern code $C = c_0 c_1 \cdots c_{n-1}$ with length n and weight w is equivalent to a set $\mathcal{D}(C^{(0)}) = \{i \mid c_i^{(0)} = 1\}$ composed of the indices of w active slots in a cycle.

code lengths. Therefore, we present the construction of Diff-Codes for any target code length in Algorithm 1. The core idea is to make use of the optimal code with similar length.

The first step (lines 1-6) in the algorithm is to build an initial, but not necessarily feasible code C of the target length n. The active slots in $C^{(0)}$ are determined by the optimal Diff-Code $C_1^{(0)}$, whose length n_1 is the largest among all the optimal Diff-Codes shorter than n. An intuitive method of initializing $C^{(0)}$ is to assign slot i active as long as slot i is active in $C_1^{(0)}$. However, we notice that for $i_1 < i_2 < n_1$, such that the i_1th and i_2th slots are active in both $C^{(0)}$ and $C_1^{(0)}$, if $i_2 - i_1 \le \lfloor \frac{n_1}{2} \rfloor$, code C and C_1 will both be feasible under the slot offset of $i_2 - i_1$. Otherwise, C_1 will satisfy the slot offset of $i_1 - i_2 + n_1$, while C is not necessarily feasible under the same slot offset. Therefore, the active slots of $C^{(0)}$ are initialized as below: for any two active slots i_1 and i_2 in $C_1^{(0)}$ ($i_1 < i_2$), they are made active in $C^{(0)}$, only if $i_2 - i_1 \le \lfloor \frac{n_1}{2} \rfloor$.

Algorithm 1: Diff-Codes Construction

```
   Input: An optimal code $C_1$ of length $n_1$.
   Output: The Diff-Code $C$ of length $n$ ($n > n_1$).
1  $\mathcal{D}(C^{(0)}) \leftarrow \emptyset$;
2  foreach $i_1, i_2 \in \mathcal{D}(C_1^{(0)}), i_1 < i_2$ do
3      if $i_2 - i_1 \le \lfloor \frac{n_1}{2} \rfloor$ then
4          $\mathcal{D}(C^{(0)}) \leftarrow \mathcal{D}(C^{(0)}) \cup \{i_1, i_2\}$;
5      end
6  end
7  $Q \leftarrow \{i \mid 1 \le i \le \lfloor \frac{n}{2} \rfloor,\ i \in N^+\}$;
8  foreach $i_1, i_2 \in \mathcal{D}(C^{(0)}), i_1 < i_2$ do
9      $tmp \leftarrow i_2 - i_1$;
10     $Q \leftarrow Q \backslash \{tmp - 1, tmp\}$;
11 end
12 while $Q \ne \emptyset$ do
13     $next, \alpha \leftarrow -1$;
14     foreach $i_1 \pmod n \notin \mathcal{D}(C^{(0)})$ do
15         $S_{i_1} \leftarrow \emptyset$;
16         foreach $i_2 \in \mathcal{D}(C^{(0)})$ do
17             $tmp \leftarrow (i_1 - i_2) \bmod n$;
18             $tmp \leftarrow min(tmp, n - tmp)$;
19             $S_{i_1} \leftarrow S_{i_1} \cup (Q \cap \{tmp - 1, tmp\})$;
20         end
21         if $|S_{i_1}| > \alpha$ then
22             $\alpha \leftarrow |S_{i_1}|$; $next \leftarrow i_1$;
23         end
24     end
25     $\mathcal{D}(C^{(0)}) \leftarrow \mathcal{D}(C^{(0)}) \cup \{next\}$;
26     $Q \leftarrow Q \backslash S_{next}$;
27 end
28 return $C$;
```

In the next step, we complete the construction greedily. According to $C^{(0)}$, we determine the set Q of all the unsatisfied integer slot offsets, *i.e.*, slot offsets under which the condition (7) in Theorem 1 is not satisfied (lines 7-11). Algorithm 1 then iteratively assigns the most *energy-efficient* slots to be active (lines 12-27). By energy-efficiency, we mean the increment in the number of satisfied integer slot offsets after the assignment.

The algorithm will return until C is a feasible Diff-Code. The number of iterations is less than the code weight. In each iteration, the algorithm traverses at most n sleeping slots, and calculates their index offsets to those already assigned active slots, which is bounded by the code weight. Considering that the code weight is also smaller than n, Algorithm 1 induces the time complexity of $O(n^3)$.

In addition, there may exist more than one perfect difference set with identical length and weight. As a result, the performance of a Diff-Code is related to which perfect difference set is chosen for construction. According to previous explanation, we prefer such element pairs (i_1, i_2) $(i_1 < i_2)$ that $i_2 - i_1$ is at most half the set length, which we denote as *offset preserving* pair. Hence, we need to determine the perfect difference set containing the most offset preserving pairs for each length. To achieve that, we implement the multiplier property, *i.e.*, multiplying each of the elements of an (n, w, 1)-perfect difference set D by p also generates a perfect difference set, as long as p is relatively prime to n. To be detailed, we multiply D by prime divisors of n, conduct cyclic shift after each multiplication, and choose the set containing the most offset preserving active slot pairs for Diff-Codes construction. To avoid excessive computations, we only pick p from numbers that are smaller than 50. The evaluation results show that the above processing can achieve superior performance.

D. Theoretical Analysis

By fixing the code length to be n, we show the theoretical bound of Diff-Codes' duty cycle. An optimal pattern code directly extended from a perfect difference set with weight w will satisfy $2[w(w-1)+1] = n$. Thus, the weight w of a Diff-Code with length n is at least $\frac{1+\sqrt{2n-3}}{2}$, which is approximately the lower bound of $\sqrt{\frac{n}{2}}$ in Theorem 2 when n is fairly large. Because an active slot is overflowed by δ, the corresponding lower bound of duty cycle is $(1+\delta)\frac{1}{\sqrt{2n}}$. On the other hand, an optimal Diff-Code whose duty cycle $c = \frac{(1+\delta)w}{2[w(w-1)+1]} \approx \frac{1+\delta}{2w}$ yields that $n \approx \frac{(1+\delta)^2}{2c^2}$ for a large w. Therefore, a Diff-Code should contain at least $\frac{(1+\delta)^2}{2c^2}$ bits to realize a duty cycle of c.

	Duty Cycle (worst-case latency n)	Worst-Case Latency (duty cycle c)
Disco	$\frac{2}{\sqrt{n}}$	$\frac{4}{c^2}$
U-Connect	$\frac{3}{2\sqrt{n}}$	$\frac{9}{4c^2}$
Searchlight-S	$(1+\delta)\frac{1}{\sqrt{n}}$	$\frac{(1+\delta)^2}{c^2}$
Optimal Diff-Code	$(1+\delta)\frac{1}{\sqrt{2n}}$	$\frac{(1+\delta)^2}{2c^2}$

TABLE I
WORST-CASE BOUNDS COMPARISONS

In Table I, we compare Diff-Codes with existing protocols, *e.g.*, Disco [6], U-Connect [9] and Searchlight [1], where

Searchlight-S is the stripped version of Searchlight in [1]. The table indicates that in the best cases, Diff-Codes can improve the worst-case latency bound by as high as 50% compared with Searchlight-S. As for Disco, the reduction of the worst-case latency is more than 80%. What's more, any Diff-Code constructed by Algorithm 1, even not optimal, can outperform other protocols, as presented in Section VII.

VI. Asymmetric Pattern Codes Design

We further extend Diff-Codes for symmetric neighbor discovery to ADiff-Codes that deal with the asymmetric case. We begin with the asymmetric feasibility conditions, and then present the design of ADiff-Codes.

A. Feasibility Conditions in Asymmetric Case

In symmetric case, the slot offset is bounded by $\frac{n}{2}$, where n is the length of the symmetric pattern code. Nevertheless, with asymmetric duty cycles, two neighboring nodes, who conform to pattern code C_1 of length n_1 and pattern code C_2 of length n_2, will have their slot offset up to $min(n_1, n_2)$. This is because the slot offset of d and $n-d$ are equivalent in symmetric case, but with duty cycle asymmetry, slot offset d is different from n_1-d or n_2-d. Then referring to Theorem 1, we can illustrate the condition for feasibility of asymmetric pattern codes as below. The proof of Theorem 4 is straightforward providing Theorem 1, and thus is omitted due to limited space.

Theorem 4. *Assume that there are two neighboring nodes A and B. Node A uses pattern code C_1 of length n_1, while B operates with C_2 of length n_2, where $n_1 \leq n_2$. They can discover each other in all the cases if the following condition is satisfied,*

$$\exists i,\ (c_{1i}^{(0)} + c_{2i}^{(j)} = 2) \vee (c_{1i}^{(0)} + c_{2i}^{(j+1)} = 2) = TRUE, \quad (16)$$

where $j \in \{0, 1, \cdots, n_1 - 1\}$, and $i < lcm(n_1, n_2)$. We note again that $\pmod{n_1}$ and $\pmod{n_2}$ are omitted for clarity.

B. ADiff-Codes Construction

A series of ADiff-Codes contains several pattern codes. Each of these patterns is feasible in symmetric case, and any two of them guarantee asymmetric feasibility. By Theorem 4, ADiff-Codes series can be constructed on basis of symmetric Diff-Codes with a similar greedy algorithm as Algorithm 1. However, inspired by the theorem as below, we present a more elegant method in this work.

Theorem 5. *There are two distinct Diff-Codes, say, C_1 with length n_1 and C_2 with length n_2 ($n_1 < n_2$). If n_1 and n_2 are relatively prime, the two Diff-Codes are feasible for asymmetric neighbor discovery.*

We do not provide its proof due to limited space, which implements the fact that under module n, if we multiply each element in the set $Z_n = \{0, 1, \cdots, n-1\}$ by q, the resulted set, denoted by qZ_n, is identical with Z_n itself, as long as q is relatively prime to n. By the above theorem, it's intuitive to construct an ADiff-Codes series. The only task is to select a set of numbers (*e.g.*, $n_1, n_2, \cdots$), any two of which are relatively prime. Then the ADiff-Codes series will contain all the Diff-Codes with corresponding lengths.

VII. Evaluation

We not only conducted comprehensive simulations, but also prototyped our designs on USRP-N210 testbed, to evaluate the discovery latencies with various specific symmetric and asymmetric pattern codes. The discovery latency is the number of slots for two neighboring nodes to discover each other since they enter each other's transmission range. For comparison, we used deterministic protocols, including Disco [6], U-Connect [9], and Searchlight-S [1], and a probabilistic protocol, Birthday [13].We first present how the worst-case latency bound changes with the symmetric duty cycle for various deterministic neighbor discovery protocols. Then, we compare the discovery latencies of different neighbor discovery protocols.

Suppose there are two nodes A and B with cycle length of n_A and n_B, respectively. Node A may be in any slot from index 0 to $n_A - 1$, at the instant it enters B's transmission range. The case is similar for node B. Thus, there are overall $n_A n_B$ different combinations of the two nodes' slot indices at the beginning of the discovery process. In the simulations, to get the cumulative distribution function (CDF) of discovery latencies of a deterministic pattern, we traverse all the possible initial combinations, and determine the discovery latency for each case. To achieve the worst-case latency bound, we set the slot boundaries of different nodes to be perfectly aligned for Disco and U-Connect, and set an interleaving of a half slot width for Searchlight-S and (A)Diff-Codes. In addition, the CDFs of Birthday protocol in simulations are calculated by the expression of discovery latency and possibility in [13]. In the testbed experiments, we randomly generate the initial indices of two USRP-N210 nodes, and compute the CDF over 200 runs.

A. Worst-Case Latency Bound of Symmetric Neighbor Discovery

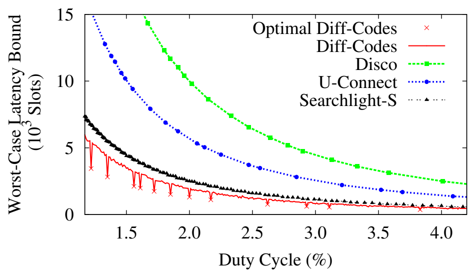


Fig. 5. Worst-Case Latency Bound *vs.* Duty Cycle

Fig. 5 demonstrates the worst-case latency bound restricted by duty cycle of various protocols. Noting that there may exist more than one pattern yielding the same duty cycle for Disco and Diff-Codes. We use adjacent prime numbers to generate Disco patterns, in which case Disco achieves better symmetric case performance. As for Diff-Codes, we select the pattern with the smallest worst-case bound regarding to each duty cycle. We observe that Diff-Codes achieve tremendously tighter worst-case latency bounds compared with the other protocols.

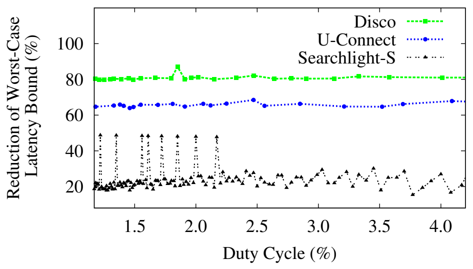


Fig. 6. Reduction of Worst-Case Latency Bound *vs.* Duty Cycle

Fig. 6 shows the improvements of worst-case latency bound achieved by Diff-Codes compared with the other protocols. Compared with Searchlight-S, with the same symmetric duty cycle, Diff-Codes can lower the worst-case latency bound by more than 20% in most cases, and the maximum reduction is as high as 50%. The average worst-case latency bound reduction of Diff-Codes over Searchlight-S, U-Connect, and Disco are 23.9%, 65.7%, and 80.8%, respectively. The above numerical results verify the effectiveness of Diff-Codes.

B. Discovery Latencies in Symmetric Case

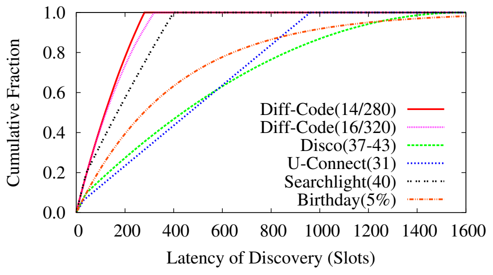


Fig. 7. CDF of Discovery Latencies for Symmetric Duty Cycle 5%

In this set of simulations, we set the duty cycle at 5%, and compare the performance of two different Diff-Codes with existing protocols. We set the cycle lengths of the two Diff-Codes at 280 and 320, the pair of primes in Disco at $(37, 43)$, the prime of U-Connect at 31, the probing period of Searchlight-S at 40 slots, and the active probability of Birthday protocol at 5%. From the cumulative distribution of discovery latencies (Fig. 7), we can see that Diff-Codes perform the best in both the median case and worst-case. Specifically, both of the two evaluated Diff-Codes realize a median gain of 30% over Searchlight-S; the minimum worst-case latency of Diff-Codes is 280 slots, which is also 30% less than that of Searchlight-S.

C. Discovery Latencies in Asymmetric Case

In the set of simulations for the asymmetric case, we consider two different scenarios. In one of the scenarios, the asymmetric duty cycles are set at 10% and 1%, while in the other scenario, the asymmetric duty cycle are set at 5% and 1%. We compare multiple setups of ADiff-Codes with the existing asymmetric neighbor discovery protocols. The parameters of the evaluated protocols are shown in Fig. 8 and Fig. 9.

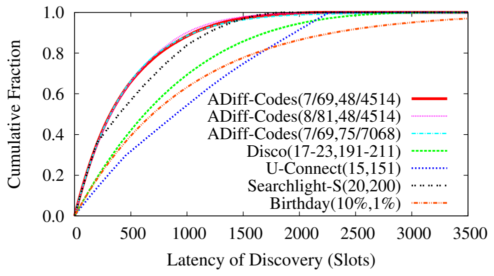


Fig. 8. CDF of Discovery Latencies for Asymmetric Duty Cycles 10%-1%

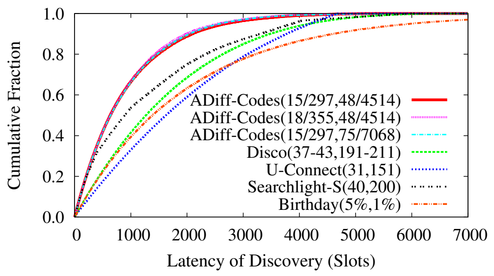


Fig. 9. CDF of Discovery Latencies for Asymmetric Duty Cycles 5%-1%

Fig. 8 shows the evaluation results for the first scenario, in which the asymmetric duty cycles are set to 10% and 1%. All the three simulated ADiff-Codes realize an improvement of about 20% in the median case compared with Searchlight-S, and outperform the other protocols before the 95-th percentile. Furthermore, Birthday protocol shows a long tail of discovery latencies. Fig. 9 shows the evaluation results for the second scenario, where the evaluated asymmetric duty cycles are 5% and 1%. In the figure, all the three ADiff-Codes reduce the median case latency by around 30% over Searchlight-S, and accomplish the discovery faster than other protocols in more than 99% of times.

Furthermore, by comparing the performance of the ADiff-Codes with different setups, we can observe that ADiff-Codes can achieve a relatively stable discovery latency, despite of the combination of duty cycles.

D. Experiment Results

To examine the performance of (A)Diff-Codes in practice, we have prototyped our designs as well as other existing protocols using Ettus USRP-N210 testbed. The discovery latencies are measured by a pre-specified node in each discovering pair. For symmetric neighbor discovery, the duty cycle is set at 5%, while for the asymmetric case, the duty cycles are set at 5% and 1%. We note that as in [1], we implemented stripped U-Connect.

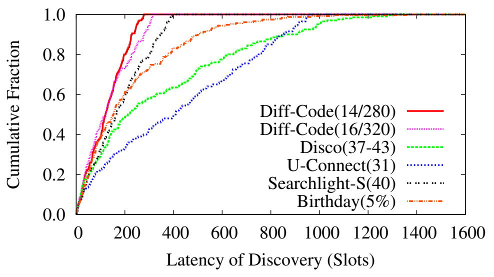


Fig. 10. Implementation: CDF of Discovery Latencies for Symmetric Duty Cycle 5%

Fig. 10 shows the experiment results for the symmetric case. The setups of all the protocols are the same as those in the simulations. The overall trends of CDFs in the figure turn out to cord with the simulation results (Fig. 7). The two Diff-Codes both perform apparently better than the other protocols. The improvement of latency in the median case is 27.7%, which is approximate to the simulation result of 30%.

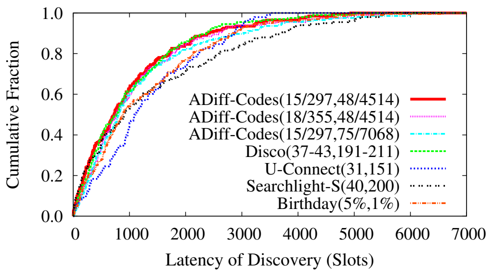


Fig. 11. Implementation: CDF of Discovery Latencies for Asymmetric Duty Cycles 5%-1%

With asymmetric duty cycles set at 5% and 1%, ADiff-Codes perform the best in a majority of cases. As illustrated in Fig. 11, the maximum gain in the median case reaches 20% compared with Searchlight-S, the worst ADiff-Codes are superior to U-Connect before the 88-th percentile. Fig. 11 also shows that Disco's performance is closed to ADiff-Codes. In contrary, in the simulations, when the asymmetric duty cycles are set at 5% and 1%, Disco has the largest latency in both the median case and worst-case. This is because the worst-case bound of Disco appears only when slot boundaries are aligned. However, the asynchronous USRP-N210 nodes always lead to slot non-alignment in experiments. Therefore, Disco performs much better in the experiments than in the simulations. Such a phenomenon was also observed and discussed in [1].

VIII. Conclusion

In this paper, we have presented a systematic study of designing highly effective and energy-efficient neighbor discovery protocols in mobile wireless networks. We have designed Diff-Codes for the case of symmetric duty cycle, and extended it to ADiff-Codes to deal with the asymmetric case. We have derived a tighter lower bound for the worst-case latency, by exploiting active slot non-alignment. Both of our simulation, and experiment results have shown that (A)Diff-Codes can achieve significantly better performance compared with state-of-art neighbor discovery protocols. Specifically, Diff-Codes can reduce the worst-case latency by up to 50%, and achieve a median gain of 30%; while ADiff-Codes are also 30% better in the median case, and outperform existing neighbor discovery protocols in more than 95% simulations and experiments.

References

[1] M. Bakht, M. Trower, and R. H. Kravets, "Searchlight: won't you be my neighbor?" in *MOBICOM*, 2012.
[2] L. D. Baumert, *Cyclic difference sets*. Springer-Verlag New York, 1971.
[3] S. Bitan and T. Etzion, "Constructions for optimal constant weight cyclically permutable codes and difference families," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 77–87, 1995.
[4] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[5] F. R. K. Chung, J. A. Salehi, and V. K. Wei, "Optical orthogonal codes: Design, analysis, and applications," *IEEE Transactions on Information Theory*, vol. 35, no. 3, pp. 595–604, 1989.
[6] P. Dutta and D. E. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *SenSys*, 2008.
[7] T. Evans and H. Mann, "On simple difference sets," *Sankhyā: The Indian Journal of Statistics*, vol. 11, pp. 357–364, 1951.
[8] J.-R. Jiang, Y.-C. Tseng, C.-S. Hsu, and T.-H. Lai, "Quorum-based asynchronous power-saving protocols for ieee 802.11 ad hoc networks," *Mobile Networks and Applications*, vol. 10, no. 1-2, pp. 169–181, 2005.
[9] A. Kandhalu, K. Lakshmanan, and R. Rajkumar, "U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol," in *IPSN*, 2010.
[10] A. Keshavarzian and E. Uysal-Biyikoglu, "Energy-efficient link assessment in wireless sensor networks," in *INFOCOM*, 2004.
[11] R. Khalili, D. Goeckel, D. F. Towsley, and A. Swami, "Neighbor discovery with reception status feedback to transmitters," in *INFOCOM*, 2010.
[12] S. Lai, B. Ravindran, and H. Cho, "Heterogenous quorum-based wake-up scheduling in wireless sensor networks," *IEEE Transactions on Computers*, vol. 59, no. 11, pp. 1562–1575, 2010.
[13] M. J. McGlynn and S. A. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *MobiHoc*, 2001.
[14] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *MobiSys*, 2010.
[15] J. Singer, "A theorem in finite projective geometry and some applications to number theory," *Transactions of the American Mathematical Society*, vol. 43, no. 3, pp. 377–385, 1938.
[16] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks," in *INFOCOM*, 2002.
[17] S. Vasudevan, J. F. Kurose, and D. F. Towsley, "On neighbor discovery in wireless networks with directional antennas," in *INFOCOM*, 2005.
[18] S. Vasudevan, D. F. Towsley, D. Goeckel, and R. Khalili, "Neighbor discovery in wireless networks and the coupon collector's problem," in *MOBICOM*, 2009.
[19] W. Zeng, S. Vasudevan, X. Chen, B. Wang, A. Russell, and W. Wei, "Neighbor discovery in wireless networks with multipacket reception," in *MobiHoc*, 2011.
[20] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *MobiHoc*, 2003.