Code-Based Neighbor Discovery Protocols In Mobile Wireless Networks

Tong Meng, *Student Member, IEEE,* Fan Wu, *Member, IEEE,* and Guihai Chen, *Member, IEEE*

Abstract—In mobile wireless networks, the emerging proximity-based applications have led to the need 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 the 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%.

Index Terms—Neighbor Discovery, Mobile Wireless Network, Protocol Design

1 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 (*e.g.*, Sony's Vita [1]). 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 due 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.

T. Meng, F. Wu, and G. Chen are with the Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, China.
E-mails: mengtong@sjtu.edu.cn, {fwu, gchen}@cs.sjtu.edu.cn.
F. Wu is the corresponding author.

- The first one is energy efficiency. It it known that it takes the mobile devices almost similar amount of energy to transmit and to listen to the wireless media [2], [32]. 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 [21]) 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 [23]. 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 [8], U-Connect [14]) cannot realize the optimal worst-case latency provided in [33]. Furthermore, although Searchlight [3] is approximate to the optimum as in [33] with symmetric 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 both 0-1 code and set theory.
- 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 in one-to-one and clique scenarios, 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 99% situations.

The rest of the paper is organized as below. In Section 2, we briefly introduce the related works. In Section 3, we explain the system model. The feasibility conditions for symmetric neighbor discovery are presented in Section 4. Then in Section 5, we propose the construction of Diff-Codes. In Section 6, we extend to the asymmetric case, and design ADiff-Codes. In Section 7, we provide the results of simulations and testbed experiments. Finally, the paper is concluded in Section 9.

2 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.

2.1 Probabilistic Protocols

McGlynn *et al.* [21] 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 [21], Keshavarzian and Uysal-Biyikoglu [16] proposed a random protocol for link assessment. Vasudevan *et al.* [29] reduced the probabilistic algorithm for neighbor discovery to the Coupon Collector's Problem. In [17], Khalili *et al.* further realized the mechanism of channel status detection, and designed algorithms providing feedback of reception. Later, Vasudevan *et al.* [27] further analyzed the problem of neighbor discovery in a general multi-hop setting. Additionally, probabilistic neighbor discovery using directional antennas was discussed in [10], [28]. Zeng *et al.* [30] extended the solution to multipacket reception networks. Karowski *et al.* [15] dealt with multi-channel neighbor discovery.

Birthday protocols support both symmetric and asymmetric cases, and have satisfying median case performance due to Birthday Paradox [22]. 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.

2.2 Deterministic Protocols

A deterministic protocol establishes a pattern scheduling the periodical operations of each node. A code-based protocol is presented in [16] utilizing constant-weight codes [5], [7], but it assumes synchronization among nodes. Moreover, Zheng *et al.* [33] 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 [33] can be further lowered. Besides, our designs fit for both symmetric and asymmetric cases with low complexity.

In a class of quorum-based protocols [18], [26], 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 [18] 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 [8] and U-Connect [14]). These neighbor discovery protocols are based on the Chinese Remainder Theorem [12]. 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.* [3] leveraged the regular relationship between the patterns of two nodes, and designed Searchlight. Compared with previous protocols, Searchlight [3] 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.

Additionally, Zhang *et al.* [31] proposed a scheme for full-duplex neighbor discovery, which implemented compressive sensing on basis of Reed-Muller Codes. Li *et al.* [19] designed a localized discovery scheme using recursive binary time partition. However, different from our work, they require nodes to be synchronized.

3 System Model

We focus on deterministic asynchronous neighbor discovery for mobile wireless networks. Similar as existing works (*e.g.*, [3], [8], [14]), 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 a sleeping slot, a node does not send or receive, and consumes negligible energy. In contrast, in an active slot, a node transmits beacons at the beginning and the end, respectively, and listens for other nodes' transmissions in between. Each beacon contains the MAC address of its sender. A node discovers its neighbors by decoding the received beacons and extracting the contained MAC addresses. Thus, in general, two neighboring nodes can discover each other when their active slots overlap. 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, *i.e.*, the active-sleep pattern defines a periodic cycle of the state transformation of a node. 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.

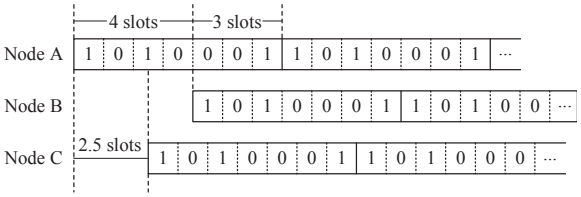


Fig. 1. Example: the slot offset between two nodes

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. 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 $\frac{n}{2}$. Thus, for the example in Fig. 1, given symmetric pattern code "1010001", the slot offset between node A and node B is regarded as $d_{AB} = 3$ slots instead of 4 slots. Besides, the value of d_{AB} is not necessarily an integer. For instance, referring to Fig. 1, d_{AC} equals 2.5.

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.

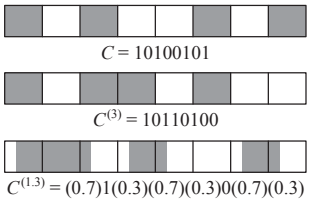


Fig. 2. Examples on Cyclic Shifts of a Pattern Code

Additionally, we relax the value of $c_i^{(j)}$ to a closed interval $[0, 1]$, to address the case of partial shift. Thus,

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

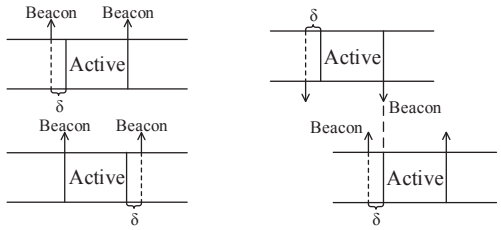
$c_i^{(j)}$ denotes the active proportion of the node in slot i regarding the shifted code $C^{(j)}$. After the cyclic shift by a non-integer j, the value of $c_i^{(j)}$ is determined by both $c_{i-\lceil j \rceil}^{(0)}$ and $c_{i-\lfloor j \rfloor}^{(0)}$. For instance, in Fig. 2, the result of cyclic 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).

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

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

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

In this work, our designs exploit the power of non-alignment of active slot boundaries. Like [3], we implement overflowed active slots, to avoid the rare case when the slot boundaries of neighboring nodes are perfectly aligned, which makes two adjacent active slots at two nodes non-overlapping. 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. We note that owing to the time durations for beacon transmission and listening, only when the active slots of two nodes overlap by a long enough period, they can discover each other. Specifically, for a 30-byte packet and a bit rate of 6 Mbps, the transmission time of each beacon is 40 μs. We can set the value of δ to be larger than 100 μs. That is a long enough period, such that adjacent active slots of two nodes can overlap and guarantee successful discovery even in the case of perfect slot alignment (as in Fig. 3(b)). Besides, the slot offset between two neighboring nodes is irrelevant to the overflowed active slots. We only consider the original slot boundaries when calculating the slot offset.



(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.

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

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

4 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. In addition, we look at the feasibility conditions from a set theoretic perspective in order to direct algorithm design.

4.1 Feasibility Conditions

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. There comes 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,*

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

where $c_i(d)$ is the value of the d^{th} bit in code $C(d)$.

Proof: Whether d is an integer or not, bit $c_i(0)$ of one node and bit $c_i(d)$ of the other always correspond to the same period of time in each cycle. Because the value of $c_i(0)$ and $c_i(d)$ are no larger than 1, the existence of such an i satisfying (6) implies the overlapping of active slots by at least δ. Therefore, the discovery is guaranteed to be successful. The sufficiency is proved.

Otherwise, if (6) does not hold, then the two nodes have no overlapping active slots, and hence cannot discover each other. Thus condition (6) is necessary. This completes the proof. □

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

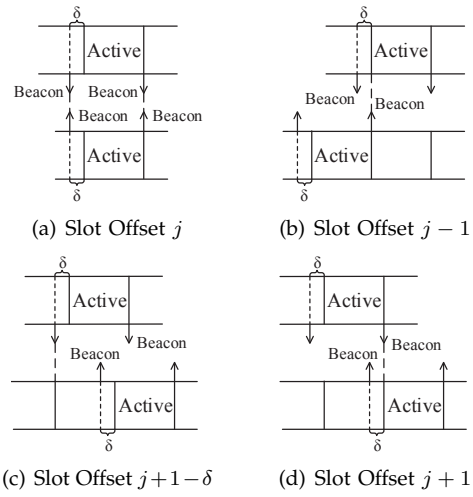
Corollary 1. *A pattern code C of length n schedules the operations of two neighboring nodes. If there exists an integer i and an integer $j \leq \lfloor \frac{n}{2} \rfloor$ such that $c_i^{(0)} = c_i^{(j)} = 1$, the two nodes can discover each other as long as their slot offset d takes its value from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

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

The above situation is also demonstrated in Fig. 4. What's more, according to Fig. 4(c), the feasibility of pattern code is not influenced by the corner case when the overflowed active slot leads to aligned active slot beginning boundary and slot ending boundary. By Lemma 1, the pattern is feasible for any $d \in [j-1, j+1]$. □

Furthermore, when random slot offset is considered, works on quorum-systems (*e.g.*, [13]) 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.



(a) Slot Offset j (b) Slot Offset $j-1$

(c) Slot Offset $j+1-\delta$ (d) Slot Offset $j+1$

Fig. 4. An Integer Slot Offset Cover a Range of Slot Offsets

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

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

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

Proof: We assume that for a pattern code C, there exists an integer slot offset d dissatisfying (7), *i.e.*,

$$\forall i, c_i^{(0)} \cdot c_i^{(d)} = c_i^{(0)} \cdot c_i^{(d+1)} = 0. \quad (8)$$

According to Lemma 1, if the pattern code C is feasible for symmetric neighbor discovery, then under the slot offset of $d + \frac{1}{2}$, there should be

$$\exists i \in N, \quad c_i(0) + c_i(d + \frac{1}{2}) \geq 1 + \delta.$$

To guarantee that the inequality can hold, there are the following possibilities of the value of $c_i(0)$ and $c_i(d+\frac{1}{2})$.

- $c_i(0) = 1$ and $c_i(d + \frac{1}{2}) = \delta$: This case is impossible, because the value of $c_i(d+\frac{1}{2})$ cannot equal δ for the non-integer slot offset $d + \frac{1}{2}$.
- $c_i(0) = 1$ and $c_i(d + \frac{1}{2}) = \frac{1}{2}$: In such case, there is $c_i^{(0)} = c_i^{(d)} = 1$, which contradicts with (8) on above. Moreover, we can draw the similar contradiction from the case of $c_i(0) = 1$ and $c_i(d + \frac{1}{2}) = \frac{1}{2} + \delta$, and the case of $c_i(0) = c_i(d + \frac{1}{2}) = 1$.
- $c_i(0) = \delta$ and $c_i(d+\frac{1}{2}) = 1$: This case leads to $c_i^{(1)} = c_i^{(d)} = c_i^{(d+1)} = 1$. Equivalently, there is $c_{i-1}^{(0)} = c_{i-1}^{(d)} = 1$ contradicting with (8) again.

Therefore, the pattern code C cannot guarantee successful neighbor discovery under the slot offset of $d+\frac{1}{2}$, and is infeasible. That proves that a feasible pattern code for symmetric neighbor discovery should satisfy the relation (7) under all the possible integer slot offsets. □

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 $O(n^2)$.

4.2 A Set Theoretic Perspective

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. Besides, we define the set of feasible integer slot offsets of a pattern code C as $\Delta(C) = \{i-j \mid i, j \in \mathcal{D}(C^{(0)}), i \neq j\}$.[2] Clearly, $0 \notin \Delta(C)$, and $d \in \Delta(C)$ implies that $n - d \in \Delta(C)$. Then we can interpret Corollary 1 and Theorem 1 from the perspective of set theory.

Corollary 2. *For a pattern code C, if $d \in \Delta(C)$, then C is feasible for symmetric neighbor discovery, as long as the slot offset between neighboring nodes is in the range of the closed interval $[min(d, n-d) - 1, min(d, n-d) + 1]$.*

Corollary 3. *A feasible pattern code C should satisfy that,*

$$j \in \{1, 2, \cdots, \left\lfloor \frac{n}{2} \right\rfloor\} \Rightarrow j \in \Delta(C) \ \vee \ j+1 \in \Delta(C). \quad (9)$$

5 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 [33]. 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 [4], [25], 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.

5.1 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:

$$Minimize \qquad w = \sum_{i=0}^{n-1} c_i, \quad (10)$$

Subject to:

$$\sum_{i=0}^{n-1} c_i(c_{i+j} + c_{i+j+1}) \geq 1, \quad \forall j \in \{0, 1, \cdots, \left\lfloor \frac{n}{2} \right\rfloor\}, \quad (11)$$

$$c_i \in \{0, 1\}, \quad \forall i \in \{0, 1, \cdots, n-1\}. \quad (12)$$

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

The lower bound of the above primal problem can be calculated via the Lagrange dual problem [6] by relaxing constraint (12). We first define the Lagrangian L,

$$\begin{aligned} L(\boldsymbol{c},\boldsymbol{\lambda},\boldsymbol{\omega},\boldsymbol{\nu}) &= \sum_{i=0}^{n-1} c_i + \sum_{i=0}^{n-1} \omega_i \cdot (c_i - 1) - \sum_{i=0}^{n-1} \nu_i \cdot c_i \\ &\quad + \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j \left[1 - \sum_{i=0}^{n-1} c_i \cdot (c_{i+j} + c_{i+j+1}) \right] \\ &= (\mathbf{1} + \boldsymbol{\omega} - \boldsymbol{\nu})^T \boldsymbol{c} + \mathbf{1}^T \boldsymbol{\lambda} - \mathbf{1}^T \boldsymbol{\omega} \\ &\quad - \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j \left[\boldsymbol{c}^T (A_j + A_{j+1}) \boldsymbol{c} \right], \end{aligned} \tag{13}$$

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

$$a_{kl}|_{A_j} = \begin{cases} 1, & \text{if } k = l + j \text{ or } k = l + j - n; \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Hence, the Lagrange dual problem takes the following form.
Objective:

$$\textit{Maximize} \quad g(\boldsymbol{\lambda},\boldsymbol{\omega},\boldsymbol{\nu}) = \inf_{\boldsymbol{c}} L(\boldsymbol{c},\boldsymbol{\lambda},\boldsymbol{\omega},\boldsymbol{\nu}),$$

Subject to:

$$\begin{aligned} \lambda_j &\geq 0, \qquad \forall j = 0, 1, \cdots, \left\lfloor \frac{n}{2} \right\rfloor, \\ \omega_i,\ \nu_i &\geq 0, \qquad \forall i = 0, 1, \cdots, n-1. \end{aligned}$$

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 [6] for the details).

Theorem 2. *An optimal solution to the relaxed problem of formulation* (10) *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 (11) and (12) 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 (11), which yields the following equation,

$$\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j^* \left[1 - \sum_{i=0}^{n-1} c_i^* \cdot (c_{i+j}^* + c_{i+j+1}^*) \right] = 0. \tag{15}$$

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

$$\begin{aligned} &\nabla_{\boldsymbol{c}} L(\boldsymbol{c}, \boldsymbol{\lambda}^*, \boldsymbol{\omega}^*, \boldsymbol{\nu}^*) \\ =\ & (\mathbf{1} + \boldsymbol{\omega}^* - \boldsymbol{\nu}^*) - \\ & \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \lambda_j^* (A_j + A_{n-j} + A_{j+1} + A_{n-j-1})^T \boldsymbol{c} \\ =\ & \mathbf{1} - (\frac{4 \sum \lambda_j}{\sqrt{2n}})_{n\times 1} = \mathbf{0}, \end{aligned} \tag{16}$$

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 (15) and (16) 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 that provided by Zheng *et al.* [33], because we exploit the power of active slot non-alignment in the asynchronous case. Consequently, compared with the active-sleep patterns in [33], which is identical with perfect difference sets, we achieve much better patterns.

5.2 Asymptotically Optimal Pattern via Perfect Difference Set

Referring to the set theoretic interpretation of pattern feasibility in Section 4.2, and the definition below, an $(n, w, 1)$-perfect difference set [4], [25] already corresponds to a feasible symmetric pattern code of length n and weight w.

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 (9) in Corollary 3. For example, a pattern code $C = 10100010000000$ can be verified to be feasible, whereas $\mathcal{D}(C^{(0)})$ is not a perfect difference set since $\Delta(C) = \{2, 4, 6, 8, 10, 12\}$. 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. The following is an illustrating example.

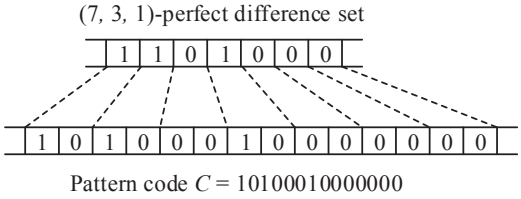


Fig. 5. Example: Extend Perfect Difference Set

According to the above definition, for a perfect difference set C', there is $\Delta(C') = \{1, 2, \cdots, n-1\}$. Therefore, if

we extend C' of length n and obtain pattern code C, we can calculate that $\Delta(C) = \{2, 4, \cdots, 2n-2\}$. Considering its length of $2n$, C is hence feasible according to Corollary 3. 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 any pair of active slots in $C^{(0)}$, say, slot i_1 and slot i_2 ($i_1 \neq i_2$), there is $i_1 - i_2 \in \Delta(C)$, as well as $i_2 - i_1 \in \Delta(C)$. In that sense, the total number of distinct active slot pairs is $\frac{1}{2}w(w-1)$. Note that (i_1, i_2) and (i_2, i_1) represent the same pair. By Corollary 3, for any pattern code of length $2n$, the weight w should satisfy the inequality,

$$\frac{1}{2} \cdot w(w-1) \geq \left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2} - \frac{1}{2}. \tag{17}$$

Considering that the length and weight of C lead to equality in (17), its optimality is verified. □

Nevertheless, a perfect difference set requires specific value of its length and weight [9]. 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 $w = p^s + 1$.

5.3 Diff-Code Construction

Although doubling the length of a perfect difference set can generate the optimal schedule, it's only suitable for specific 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.[3] 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 $i_2 - i_1 \leq \lfloor \frac{n_1}{2} \rfloor$, 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 $i_2 - i_1 \leq \lfloor \frac{n_1}{2} \rfloor$.

3. The shortest optimal Diff-Code has the length of 14, and the duty cycle of 21.4%. In practice, owing to limited battery power, we tend to need a pattern code with smaller duty cycle and larger length. Thus, the existence of optimal Diff-Codes shorter than the target length can be guaranteed.

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** $i_2 - i_1 \leq \lfloor \frac{n_1}{2} \rfloor$ **then**
4 $\mathcal{D}(C^{(0)}) \leftarrow \mathcal{D}(C^{(0)}) \cup \{i_1, i_2\}$;
5 **end**
6 **end**
7 $Q \leftarrow \{i \mid 1 \leq i \leq \lfloor \frac{n}{2} \rfloor,\ i \in N^+\}$;
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** $Q \neq \emptyset$ **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 (9) in Corollary 3 is not satisfied (lines 7-11). Then, Algorithm 1 iteratively checks each of the remaining sleeping slots, and calculates the number of newly satisfied integer slot offsets if the slot is active (lines 15-20). In the end of each iteration, the slot that brings the largest increment in the number of satisfied integer slot offsets is assigned to be active (line 25), and the set Q is updated accordingly (line 26).

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. If considering that the code weight is also smaller than n, Algorithm 1 seems to be in $O(n^3)$. Then, referring to the lower bound of code weight (see Section 5.4), it actually induces a time complexity of $O(n^2)$.

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 such integer p that is relatively prime to 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.

5.4 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 $2\left[w(w-1)+1\right] = n$. 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 1
Worst-Case Bounds Comparisons

In Table 1, we compare Diff-Codes with existing protocols, *e.g.*, Disco [8], U-Connect [14] and Searchlight [3], where Searchlight-S is the stripped version of Searchlight in [3]. 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 7.1.

5.5 Diff-Code Seeking with Fixed Duty Cycle

The construction of Diff-Codes discussed till now focuses on minimizing the code weight while the code length is fixed. However, in practice, a user may prefer selecting the appropriate pattern with whatever duty cycle according to the remaining battery of his/her mobile device. Thus it's necessary to support Diff-Codes construction that minimizes the worst-case latency with a fixed duty cycle. We finish this section by a heuristic algorithm accomplishing such a task.

Briefly speaking, given a duty cycle c, an interval $[n_{min}, n_{max}]$ is determined, which is expected to cover the cycle lengths of the Diff-Codes whose duty cycle are around c. Then, we utilize dichotomy to construct Diff-Codes, until a pattern code whose duty cycle is close enough to c has been obtained. Specifically, for interval $[n_{min}, n_{max}]$, we construct three Diff-Codes with lengths n_{min}, n_{max}, and $n_{mid} = (n_{min} + n_{max})/2$, respectively. For convenience, we assume the duty cycles of such three Diff-Codes are c_{min}, c_{max}, and c_{mid}. If c_{mid} is closer to c_{min}, we then turn to the interval $[n_{min}, n_{mid}]$. Otherwise, we turn to $[n_{mid}, n_{max}]$.

Obviously, the performance of the above algorithm is determined by the initial estimations of n_{min} and n_{max}. According to Table 1, we set n_{min} and n_{max} to be the optimal worst-cast latency bounds under the duty cycles of c and $c - \sigma$, respectively. Besides, the width of the interval can be adjusted according to accuracy requirements, *i.e.*, a larger σ is used for higher accuracy. For example, the above heuristic algorithm achieves good performance when $\sigma \in [0.5\%, 1.0\%]$. Because Algorithm 1 is in $O(n^2)$, this heuristic algorithm has the complexity of $O(n^2 \log n)$.

6 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.

6.1 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,*

$$\exists i,\ (c_{1i}^{(0)} + c_{2i}^{(j)} = 2) \vee (c_{1i}^{(0)} + c_{2i}^{(j+1)} = 2) = TRUE, \quad (18)$$

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

6.2 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 Theorem 5 and Theorem 6 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.*

Proof: For convenience, assume slot 0 with regard to $C_1^{(0)}$ is aligned with slot 0 of $C_2^{(0)}$. In such case, the indices of $C_2^{(0)}$'s slots, which are aligned with slot 0 in $C_1^{(0)}$, form the set S represented as follow,

$$S = \{(k \cdot n_1) \bmod n_2 \mid 0 \leq k < n_2, k \in N\}.$$

What's more, there exists the following fact: 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.

Therefore, $S = n_1 Z_{n_2} = Z_{n_2}$, which indicates that slot 0 in $C_1^{(0)}$ witnesses all the slots of code $C_2^{(0)}$. Similarly, every slot of $C_1^{(0)}$ can overlap with all the slots of $C_2^{(0)}$. It means that any two integer slot offsets are equivalent, and thus, the assumption of aligned slot 0 for the two pattern codes do not affect the correctness of the proof. As a result, condition (18) holds for any value of j. Thus Diff-Codes C_1 and C_2 are feasible in asymmetric case. □

Theorem 6. *Two distinct Diff-Codes, say, C_1 with length n_1 and C_2 with length n_2 $(n_1 < n_2)$, are feasible for asymmetric neighbor discovery, as long as $\frac{n_1}{2}$ and $\frac{n_2}{2}$ are relatively prime.*

Proof: Because $\frac{n_1}{2}$ is relatively prime to $\frac{n_2}{2}$, we have the following condition,

$$\left\{(k \cdot \frac{n_1}{2}) \bmod \frac{n_2}{2} \mid 0 \leq k < \frac{n_2}{2}, k \in N\right\} = \frac{n_1}{2} Z_{\frac{n_2}{2}} = Z_{\frac{n_2}{2}}.$$

Again we assume that slot 0's in $C_1^{(0)}$ and $C_2^{(0)}$ are aligned. Then the slot indices of $C_2^{(0)}$ that are aligned with slot 0 with regard to $C_1^{(0)}$ form the set S as below,

$$\begin{aligned} S &= \{(k \cdot n_1) \bmod n_2 \mid 0 \leq k < n_2, k \in N\} \\ &= 2Z_{\frac{n_2}{2}} = \{0, 2, 4, \cdots, n_2 - 2\}. \end{aligned}$$

In such a case, slot i in $C_1^{(0)}$ is aligned with the slots of $C_2^{(0)}$, whose indices are in the following set,

$$\{i, i+2, i+4, \cdots, i+n_2-2\}.$$

Similar to the proof of Theorem 5, we have the conclusion that the slot offset of j is equivalent to any slot offset of $j + 2k$ $(k \in Z)$. Therefore, condition (18) holds. This completes the proof. □

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.*, $n_1, n_2, \cdots$), such that any two of them, or the half of any two, are relatively prime. Then the ADiff-Codes series will contain all the Diff-Codes with corresponding lengths.

6.3 Worst-Case Bound of Discovery Latency

It can be drawed from Theorem 5 and 6 that, in the asymmetric case, the worst-case latency bound of an ADiff-Codes series is determined by its composing Diff-Codes. Thus there is not a stable relation between the worst-case bound and the lengths of the composing Diff-Codes. In the following, we provide an algorithm for the purpose of worst-case latency bound calculation.

Algorithm 2 takes the two Diff-Codes that form a series of ADiff-Codes as inputs. According to Theorem 5, any two integer slot offsets i and j are equivalent, provided that n_1 and n_2 are relatively prime. By Theorem 6, slot offset i is equivalent to $(i + 2k)$ $(k \in Z)$, if $\frac{n_1}{2}$ is relatively prime to $\frac{n_2}{2}$. Therefore, the algorithm only need to consider two slot offsets, which are 0 and 1 (line 2). With each slot offset, the algorithm traverses $N = lcm(n_1, n_2)$ consecutive slots, and calculates the largest interval between two adjacent instances of overlapping active slots (line 4-20). It is enough to consider N consecutive slots, because for two nodes that have the corresponding patterns of C_1 and C_2, their slot indices at the same instant change with a cycle of length N. Finally, the worst-case latency bound is determined by the largest interval in all cases. In Algorithm 2, the number of iterations for each of the two slot offsets is bounded by N, and it only involves several single-step operations in each iteration. Therefore, the complexity of Algorithm 2 is $O(N) = O(n_1 \cdot n_2)$.

Algorithm 2: Worst-Case Latency Calculation

Input: Two Diff-Codes in an ADiff-Codes series, which are C_1 of length n_1, and C_2 of length n_2.
Output: The worst-case bound of discovery latency, L, between C_1 and C_2.

$L \leftarrow 0$; $N \leftarrow lcm(n_1, n_2)$;
foreach $d \in \{0, 1\}$ **do**
  $l, l_0 \leftarrow -1$;
  foreach $s \in [0, N-1]$ **do**
    $index_1 \leftarrow s \pmod{n_1}$;
    $index_2 \leftarrow s + d \pmod{n_2}$;
    if $c_{1index_1} \cdot c_{2index_2} + c_{1index_1} \cdot c_{2(index_2+1)} = 0$ **then**
      continue ;
    end
    if $l_0 = -1$ **then**
      $l_0 \leftarrow s$;
    end
    if $l \neq -1$ **then**
      $L \leftarrow max(L, s - l)$;
    end
    $l \leftarrow s$;
  end
  $L \leftarrow max(L, N - l + l_0)$;
  if $L = N$ **then**
    break ;
  end
end
return L;

7 Evaluation

We not only conducted comprehensive simulations, but also prototyped our designs on a USRP-N210 testbed, to evaluate the discovery latencies with various specific symmetric and asymmetric pattern codes. For comparison, we used deterministic protocols, including Disco [8], U-Connect [14], and Searchlight-S [3], and a probabilistic protocol, Birthday [21]. In this section, we first present how the worst-case latency bound changes with the symmetric duty cycle for various deterministic neighbor discovery protocols. Then, we compare both symmetric and asymmetric discovery latencies of different neighbor discovery protocols in two scenarios: one-to-one neighbor discovery and clique neighbor discovery. We compare different neighbor discovery protocols using similar duty cycles as in previous works (*e.g.*, Disco [8], U-Connect [14], and Searchlight [3]).

7.1 Worst-Case Bound of Symmetric Discovery Latency

Fig. 6 demonstrates the worst-case latency bound of various neighbor discovery protocols restricted by symmetric duty cycle. Note 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.

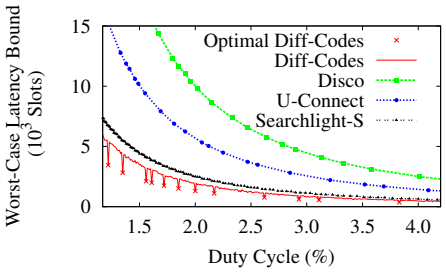


Fig. 6. Worst-Case Latency Bound *vs.* Duty Cycle

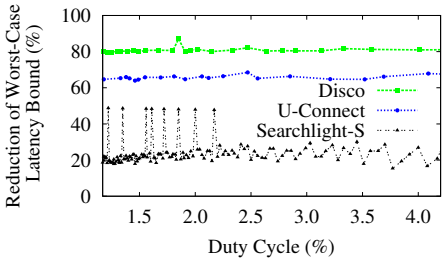


Fig. 7. Reduction of Worst-Case Latency Bound

Fig. 7 shows the improvements of worst-case latency bound achieved by Diff-Codes compared with the other protocols. We note that we compare Diff-Codes with other neighbor discovery protocols under exactly the same duty cycles. 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%. Specifically, those cases of maximum reduction in Fig. 7 correspond to optimal Diff-Codes, which are in correspondence to Table 1, as well. 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.

7.2 One-to-One Neighbor Discovery Latencies

In the one-to-one scenario, there are exactly two nodes conducting neighbor discovery, or equivalently, a node has only one neighbor in its proximity to discover. The discovery latency is the number of slots for two neighboring nodes to discover each other since they enter each other's transmission range. Suppose the two nodes are 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. Therefore, 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. Moreoever, to achieve the worst-case latency bound, we set the slot boundaries of different nodes to be perfectly aligned for Disco and U-Connect, whlie setting an interleaving of half the 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 duty cycle in [21].

7.2.1 Discovery Latencies in Symmetric Case

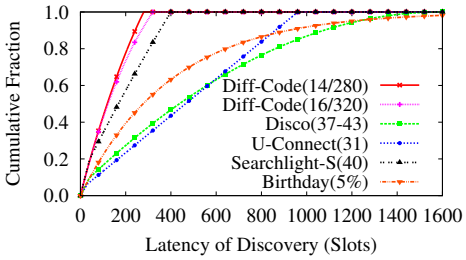


Fig. 8. CDF of One-to-One Discovery Latencies for Symmetric Duty Cycle 5%

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. 8), 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 nearly 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.

7.2.2 Discovery Latencies in Asymmetric Case

In the simulations for asymmetric one-to-one neighbor discovery, we consider the asymmetric duty cycles of 5% and 1%. The parameters of the evaluated protocols are shown in Fig. 9. In addition, we compare multiple setups of ADiff-Codes, which have the same set of asymmetric duty cycles, as well.

Fig. 9 shows the evaluation results for various neighbor discovery protocols. The simulated ADiff-Codes outperform Searchlight-S, Disco all along. Specifically, ADiff-Codes reduce the median discovery latency by 26.0% compared to Searchlight-S, and by 47.9% compared to Disco. The worst-case gains of ADiff-Codes are 15.9% and 28.2% over Searchlight-S and Disco, respectively. What's more, in comparison with U-Connect,

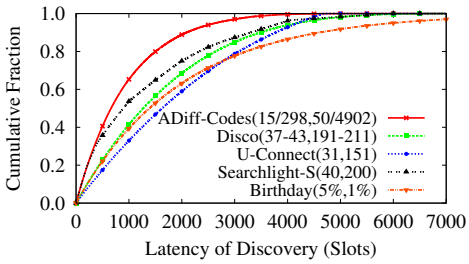


Fig. 9. CDF of One-to-One Discovery Latencies for Asymmetric Duty Cycles 5%-1%

ADiff-Codes reduce the median case discovery latency by as high as 59.3%, and achieve smaller latencies for more than 99.9% of times, while having a worst-case bound that is only 7.1% larger.

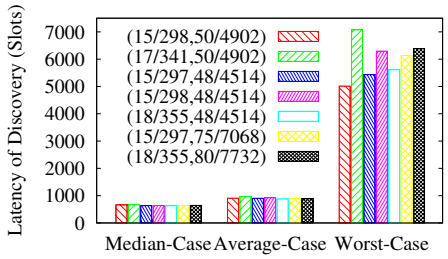


Fig. 10. One-to-One Discovery Latencies of Multiple ADiff-Codes Series with Asymmetric Duty Cycles 5%-1%

In Fig. 9, there is only one single series of ADiff-Codes. However, for the same set of asymmetric duty cycles, many ADiff-Codes can be constructed. Hence, in Fig. 10, we compare the discovery latencies of various ADiff-Codes setups conforming to the same asymmetric duty cycles of 5% and 1%. There are seven ADiff-Codes series presented in the figure. Even though their worst-case latencies show big differences, the latencies in the median and the average cases are fairly close. That indicates ADiff-Codes can achieve a relatively stable discovery latency, despite of the combination of duty cycles. Furthermore, from the perspective of practical implementation, it tends to be acceptable in mobile wireless networks as long as enough neighbors are found. For example, even though there may be more than 10 mobile device users in a coffee bar, a guest is satisfied to discover only 4 of them to start a card game. It is different from neighbor discovery in sensor networks, where failing to discover a neighbor can result in unreachable nodes in transmission. This means that if a neighbor discovery protocol has short latencies in most cases, it can satisfy user's demands well most of the times in practice. Considering the tremendous latency gain in the median case, and the superior performance in most of the times, *i.e.*, all along in symmetric case, and more than 99% according to the one-to-one asymmetric simulation results, our design of (A)Diff-Codes should have great advantage in reality.

7.3 Clique Neighbor Discovery Latencies

Besides the one-to-one discovery latencies, we also examine the performance of our design, when there are multiple neighbors within a nodes transmission range. The discovery latency in the scenario of clique neighbor discovery is the number of slots for a node to discover all its neighbors. What's more, as explained in Section 7.2, practical applications can be satisfied by discovering enough number of neighbors. Thus, we also compare different neighbor discovery protocols using 90%-latency, *i.e.*, the latency of discovering 90% of neighbors.

We assume that a node A has M neighbors, and the cycle lengths of A and its neighbors are n_A and $n_i (1 \leq i \leq M)$, respectively. Then, there are overall $n_A \cdot \prod_{i=1}^{M} n_i$ combinations of the initial slot indices for neighbor discovery. In such case, even for $M = 10$ and $n_A = n_i = 100$, there are as many as 10^{20} different combinations. Therefore, it is impractical to go through all those probabilities as in the evaluation of one-to-one neighbor discovery. Instead, we calculate the cumulative distribution of discovery latencies out of as many as 10^5 random initial combinations. Furthermore, with multiple neighbors, the beacons from different neighbors may collide at node A, leading to that node A cannot decode any one of them. For simulation, we set one of the M neighbors to have the identical slot interleaving as in the one-to-one case, which reflects the worst-case performance. In addition, the other neighbors have random slot interleaving compared with A. As explained in Section 8.2, each time slot is of length $20\ ms$, and the difference between slot interleaving of two neighboring nodes should exceed $50\ \mu s$ to avoid interference.

7.3.1 Discovery Latencies in Symmetric Case

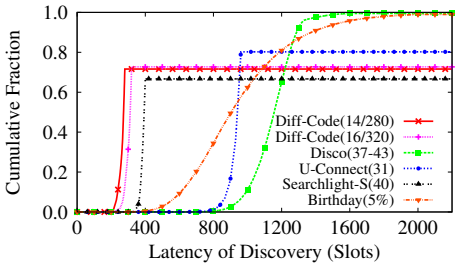
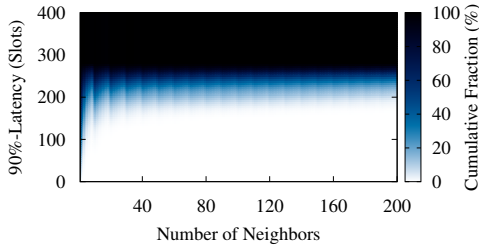
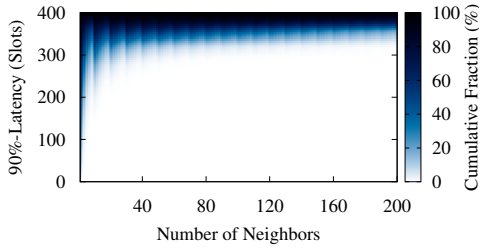


Fig. 11. CDF of Clique Discovery Latencies with 50 neighbors for Symmetric Duty Cycle 5%

For symmetric neighbor discovery with the existence of multiple neighbors, we set the duty cycle of all the nodes at 5%. Fig. 11 presents the CDF of discovery latencies when a node has 50 neighbors overall. The two simulated Diff-Codes outperform other neighbor discovery protocols significantly. Specifically, the two Diff-Codes achieve median gains of 30.7% and 20.8% compared with Searchlight-S, respectively. However, in the worst-case, the evaluated neighbor discovery protocols cannot always discover all the 50 neighbors. For example, the two Diff-Codes can discover all the neighbors in 71.6% and 72.7% simulations, respectively, and Searchlight-S only converges at 66.7%. This is because the interference of concurrent discovery signals cannot be ignored in the clique scenario. Therefore, as stated in Section 7.2, we focus on the 90%-latency in the following simulations of clique neighbor discovery.



(a) Diff-Code (14/280)



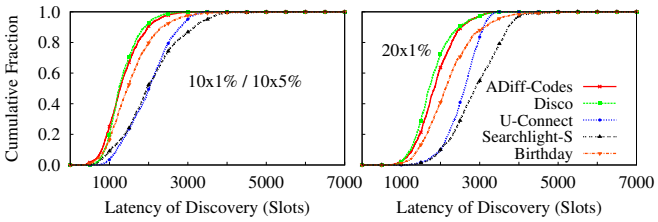
(b) Searchlight-S (40)

Fig. 12. CDF of 90%-Latency with up to 200 neighbors for Symmetric Duty Cycle 5%

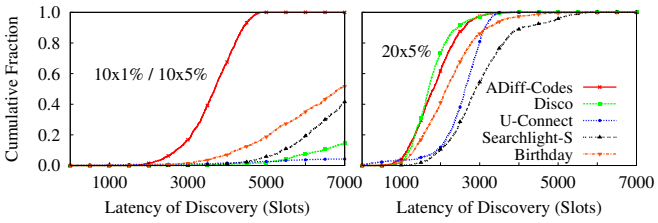
To obtain how the cumulative distributions of 90%-latencies change as the number of neighbors increases, we compare the CDFs of Diff-Code$(14/280)$ with Searchlight-S, which is the best among existing protocols. As shown in Fig. 12, Diff-Code has better performance than Searchlight-S in all cases. The figures also indicate that both neighbor discover protocols can discover at least 90% of all the neighbors, regardless of the number of neighbors. Thus, although there are interferences among nodes in clique neighbor discovery, user can still discover enough neighbors with high possibility.

7.3.2 Discovery Latencies in Asymmetric Case

Because the duty cycle of a mobile device is independently determined by its energy budget, there may exist various duty-cycled neighbors in a node's proximity. That is to say, the node may have both symmetric neighbors with the same duty cycle, and asymmetric neighbors that have different duty cycles. Hence, in our simulations, we set the duty cycle of node A to be 5% and 1%, respectively, and node A has 20 neighboring nodes. For each duty cycle of node A, we consider two cases: (1) all of the 20 nodes are asymmetric neighbors of node A, and (2) half of the neighbors have duty cycle of 1% and the other half have 5%.



(a) 5% Duty-Cycled Node A



(b) 1% Duty-Cycled Node A

Fig. 13. CDF of 90%-Latency for Clique Neighbor Discovery with 20 neighbors

Fig. 13 presents the CDFs of 90%-latency in different cases. We note that all the deterministic neighbor discovery protocols adopt the same active-sleep patterns as in one-to-one case, with duty cycles of 5% and 1%. The performance of Disco and U-Connect is better than that in one-to-one case. This is because in our setups for clique neighbor discovery, most neighbors have non-aligned time slots with node A. What's more, it is apparent and reasonable that the 90%-latencies in the worst-case are smaller than the worst-case bound of discovery latencies (as shown in Fig. 9). Besides, we can see from the figures that the ADiff-Codes series outperforms all the other protocols except Disco, which performs only slightly better than ADiff-Codes. In addition, both Fig. 13(a) and 13(b) show that the 90%-latency in the median case becomes smaller as the number of neighbors with 5% increases. Specifically, in Fig. 13(a), ADiff-Codes achieve the median case 90%-latencies of 1842 and 1273 slots, corresponding to the cases where there are 0 and 10 neighbors with duty cycle of 5%, respectively. That corresponds to our common sense that in clique neighbor discovery, the latencies to discover neighboring nodes tend to decrease when the neighboring nodes have higher duty cycles.

7.4 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. In our experiments, we set the length of each slot to be 20 ms, and use a 30-byte packet that contains the MAC address as beacon for neighbor discovery. The USRP node should decode the incoming packets to discover neighboring nodes. The discovery latencies are measured by a pre-specified node in each discovering pair/clique. As in simulations, we consider both one-to-one and clique neighbor discovery. In the one-to-one scenario, the duty cycle in the symmetric case is set at 5%, while the asymmetric duty cycles are set at 5% and 1%. For the clique neighbor discovery, the pre-specified node has the duty cycle of 5%, with 3 neighboring nodes of 5% and 2 neighboring nodes of 1%.[4] In each set of experiments, we randomly generate the initial indices of the USRP-N210 nodes, and compute the CDF over 200 runs. We note that overflowed active slots are adopted in all the realization of neighbor discovery protoocls. Therefore, we strip the consecutive active slots, which may appear in Disco and U-Connect. In Birthday protocol, the USRP node generates random number in each slot, which is used to determine the state of the next slot. Additionally, in the clique scenario, we consider the latencies to discovery all the neighbors, because the number of neighbors is quite limited. Thus, in our experiments, interfering discovery signals are rare, and only have a very minor impact.

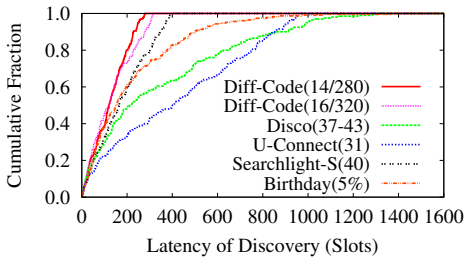


Fig. 14. Implementation: CDF of Discovery Latencies for Symmetric Duty Cycle 5%

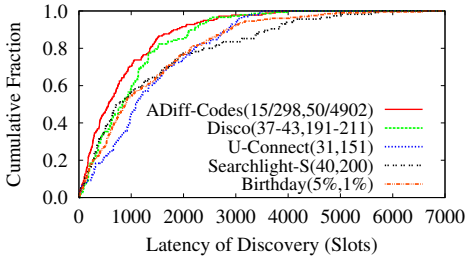


Fig. 15. Implementation: CDF of Discovery Latencies for Asymmetric Duty Cycles 5%-1%

Fig. 14 shows the experiment results for symmetric one-to-one neighbor discovery. The overall trends of CDFs in the figure cord with the simulation results (Fig. 8). The two Diff-Codes both perform apparently better than the other protocols. Compared with Searchlight-S, the reduction of discovery latency in the median case is 27.7%, which is approximate to the median gain of nearly 30% in simulations. Next, with the asymmetric duty cycles set at 5% and 1%, ADiff-Codes perform the

4. In our experiments, the link between two USRP nodes has good quality, and we do not manage to handle collisions. Thus, packet loss is possible with a small probability.

best in a majority of one-to-one cases. As illustrated in Fig. 15, the gain of discovery latency in the median case reaches 25.5%, compared with Searchlight-S, and the evaluated ADiff-Codes series is superior to U-Connect before the 98-th percentile.

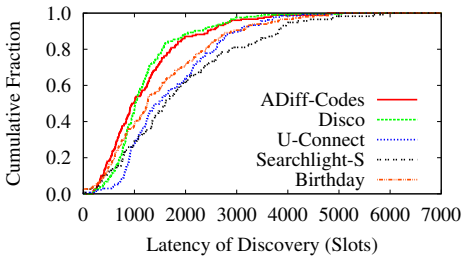


Fig. 16. Implementation: CDF of Clique Discovery Latencies

Fig. 16 presents the CDFs of discovery latencies in clique neighbor discovery experiments. The performance of ADiff-Codes is similar to that of Disco. The median case gain compared to Searchlight-S is as high as 38.6%. What's more, although according to simulation results (Fig. 9), the worst-case discovery latency of U-Connect is slightly smaller than ADiff-Codes, the experiment results in Fig. 16 turn out that ADiff-Codes even achieve slightly better worst-case latency than U-Connect.

Finally, we notice that in all the experiments, the performance of Disco turns out to be much better than that in the one-to-one simulations. For example, in Fig. 15 and 16, Disco outperforms Searchlight-S, and achieves similar CDFs as ADiff-Codes. This is because the asynchronous USRP-N210 nodes always lead to slot non-alignment in experiments. However, in simulations of one-to-one neighbor discovery, we set aligned slot boundaries so as to get the worst-case latencies of Disco. Owing to the similar reason, the performance of different protocols corresponds to the simulation results in the clique scenario. Such effects are observed in [3], due to the power of slot non-alignment, as well.

8 Discussion

In this section, we discuss several improtant issues on the implementation of Diff-Codes.

8.1 MAC/PHY Compatibility

Most deterministic neighbor discovery protocols, including (A)Diff-Codes in this work, design the active-sleep patterns that schedule the state transformation of nodes between active and sleeping. Previous works (*e.g.*, [8], [14], *etc.*) propose to implement active-sleep patterns in wireless sensor networks with existing low-power listening protocols, such as B-MAC [24]. For the application of (A)Diff-Codes, we can use a counter that counts from 0 to $n-1$ repeatedly, where n is the code length. The counter increases by one every time slot, and the node can turn active at corresponding slot indices. In practice, each node can store a whole ADiff-Codes series locally, and turn to a Diff-Code in the series according to its requirement on the duty cycle. The counter will be tuned accordingly.

In mobile wireless networks, neighbor discovery is conducted among wireless devices. Although according to [3], the slot duration on the order of seconds is needed on the application level, we can reduce the duration to the order of milliseconds by turning on/off the WiFi interface in the kernel level [11]. Hence, we have adopted time slots of 20 ms for evaluations in this work. In addition, as explained in Section 1, one of the objectives of neighbor discovery in mobile wireless networks is to find neighboring devices to enjoy various proximity-based applications. In that case, neighbor discovery tends to be an on-demand service, and does not need to operate all the time. Hence, it is acceptable to use a relatively high duty cycle, as evaluated in this work and most existing works.

Furthermore, neighbor discovery relies on the WiFi radios on wireless devices, which work on the $2.4/5$G frequency band. Consequently, it will not impact or interrupt most cellular services such as phone call and cellular data service (*e.g.*, GSM on 1.9G band, LTE on 700M band, *etc.*). However, most of the existing neighbor discovery protocols cannot co-exist with other WiFi transmissions. In practice, we can alleviate the problem of interference by choosing a less-busy channel in 2.4 GHz band, or turning to 5 GHz band, where there are not heavy WiFi transmissions. Whats more, the beacons for neighbor discovery adopted in existing protocols can be replaced with correlatable symbol sequences, which have been proved to be robust under the SNR of -6 dB [20].

8.2 Clique Neighbor Discovery

In the scenario of clique neighbor discovery, the possibility of interference between the beacons from different neighboring nodes increases with the number of nodes in the network. For a 30-byte packet, its transmission time is 40 μs under the bit rate of 6 Mbps, and even shorter under other bit rates. If we further take the hardware jitter into consideration, the interval between two beacons needs to be at least 50 μs, so that a node can receive both of them without interference. Equivalently, when a node A has two neighbors, the slot offset between those two neighbors should be no less than 50 μs. In the evaluation, we use the time slot of 20 ms. With such slot duration, the slot offset between any two nodes is below 50 μs in only 0.25% of time. In addition, when the slot offset between the two neighboring nodes is below 50 μs, they still cannot cause interference at node A if they do not turn active at the same time with node A. Considering that there are only limited active slots within each cycle under the relatively low duty cycle, the possibility that the two neighboring nodes may interfere with each other is actually even lower. What's

more, as claimed in Section 7.2, instead of discovering all the neighbors, it is acceptable in practical applications to discover a high enough proportion of them. Therefore, in practice, a node can discover enough neighbors in most cases.

At last, each node needs to select the duty cycle for neighbor discovery according to their remaining battery power. Our designs can guarantee that any two of such Diff-Codes are feasible for symmetric/asymmetric neighbor discovery. Therefore, (A)Diff-Codes can adapt to the complex scenario where each node in the network has different neighborhood as well as specific duty cycle. In this work, we consider the evaluation of some simple clique cases for clarity.

9 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 in both one-to-one and clique neighbor discovery, compared with state-of-art neighbor discovery protocols. Specifically, in the one-to-one scenario, Diff-Codes can reduce the worst-case latency by up to 50%, and achieve a median gain of around 30%; while ADiff-Codes are also 30% better in the median case, and outperform existing neighbor discovery protocols in more than 99% simulations and experiments. In the clique scenario, both Diff-Codes and ADiff-Codes have smaller latencies to discover 90% of all the neighbors.

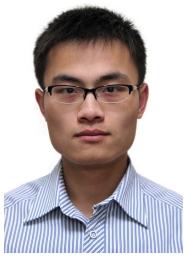
Acknowledgements

This work was supported in part by the State Key Development Program for Basic Research of China (973 project 2012CB316201), in part by China NSF grant 61422208, 61472252, 61272443 and 61133006, in part by CCF-Intel Young Faculty Researcher Program and CCF-Tencent Open Fund, in part by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, and in part by Jiangsu Future Network Research Project No. BY2013095-1-10. 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.

References

[1] "Sony ps vita - near," *http://us.playstation.com/psvita*.
[2] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta, "Wireless wakeups revisited: energy management for voip over wi-fi smartphones," in *MobiSys*, 2007.
[3] M. Bakht, M. Trower, and R. H. Kravets, "Searchlight: won't you be my neighbor?" in *MOBICOM*, 2012.
[4] L. D. Baumert, *Cyclic difference sets*. Springer-Verlag New York, 1971.
[5] 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.
[6] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[7] 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.
[8] P. Dutta and D. E. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *SenSys*, 2008.
[9] T. Evans and H. Mann, "On simple difference sets," *Sankhyā: The Indian Journal of Statistics*, vol. 11, pp. 357–364, 1951.
[10] E. Felemban, R. Murawski, E. Ekici, S. Park, K. Lee, J. Park, and Z. Hameed, "Sand: Sectored-antenna neighbor discovery protocol for wireless networks," in *SECON*, 2010.
[11] H. Han, Y. Liu, G. Shen, Y. Zhang, and Q. Li, "Dozyap: power-efficient wi-fi tethering," in *MobySys*, 2012.
[12] G. G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*. Oxford University Press, 1979.
[13] 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.
[14] A. Kandhalu, K. Lakshmanan, and R. Rajkumar, "U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol," in *IPSN*, 2010.
[15] N. Karowski, A. C. Viana, and A. Wolisz, "Optimized asynchronous multi-channel neighbor discovery," in *INFOCOM*, 2011.
[16] A. Keshavarzian and E. Uysal-Biyikoglu, "Energy-efficient link assessment in wireless sensor networks," in *INFOCOM*, 2004.
[17] R. Khalili, D. Goeckel, D. F. Towsley, and A. Swami, "Neighbor discovery with reception status feedback to transmitters," in *INFOCOM*, 2010.
[18] 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.
[19] D. Li and P. Sinha, "RBTP: Low-power mobile discovery protocol through recursive binary time partitioning," *IEEE Transactions on Mobile Computing*, vol. 13, no. 2, pp. 263–273, 2014.
[20] E. Magistretti, O. Gurewitz, and E. W. Knightly, "802.11 ec: collision avoidance without control messages," in *Mobicom*, 2012.
[21] 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.
[22] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.
[23] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *MobiSys*, 2010.
[24] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys*, 2004.
[25] 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.
[26] 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.
[27] S. Vasudevan, M. Adler, D. Goeckel, and D. Towsley, "Efficient algorithms for neighbor discovery in wireless networks," *IEEE Transactions on Networking*, vol. 21, no. 1, pp. 69–83, 2013.
[28] S. Vasudevan, J. F. Kurose, and D. F. Towsley, "On neighbor discovery in wireless networks with directional antennas," in *INFOCOM*, 2005.
[29] S. Vasudevan, D. F. Towsley, D. Goeckel, and R. Khalili, "Neighbor discovery in wireless networks and the coupon collector's problem," in *MOBICOM*, 2009.
[30] W. Zeng, S. Vasudevan, X. Chen, B. Wang, A. Russell, and W. Wei, "Neighbor discovery in wireless networks with multipacket reception," in *MobiHoc*, 2011.
[31] L. Zhang and D. Guo, "Neighbor discovery in wireless networks using compressed sensing with reed-muller codes," in *WiOpt*, 2011.

[32] X. Zhang and K. G. Shin, "E-mili: energy-minimizing idle listening in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 9, pp. 1441–1454, 2012.
[33] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *MobiHoc*, 2003.



Tong Meng is a graduate student from the Department of Computer Science and Engineering at Shanghai Jiao Tong University, P. R. China. He received his B.S. in Computer Science from Shanghai Jiao Tong University in 2013. His research interests encompass neighbor discovery, routing in wireless networks and mobile social networks. He is a student member of ACM and CCF.

Fan Wu is an associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He received his B.S. in Computer Science from Nanjing University in 2004, and Ph.D. in Computer Science and Engineering from the State University of New York at Buffalo in 2009. He has visited the University of Illinois at Urbana-Champaign (UIUC) as a Post Doc Research Associate. His research interests include wireless networking and mobile computing, algorithmic network economics, and privacy preservation. He has published more than 70 peer-reviewed papers in leading technical journals and conference proceedings. He is a receipt of China National Natural Science Fund for Outstanding Young Scientists, CCF-Intel Young Faculty Researcher Program Award, CCF-Tencent Rhinoceros bird Open Fund, and Pujiang Scholar. He has served as the chair of CCF YOCSEF Shanghai, on the editorial board of Elsevier Computer Communications, and as the member of technical program committees of more than 40 academic conferences. For more information, please visit http://www.cs.sjtu.edu.cn/~fwu/.

Guihai Chen obtained his B.S. degree from Nanjing University, M. Engineering from Southeast University, and Ph.D. from University of Hong Kong. He visited Kyushu Institute of Technology, Japan in 1998 as a research fellow, and University of Queensland, Australia in 2000 as a visiting Professor. During September 2001 to August 2003, he was a visiting Professor in Wayne State University. He is a Distinguished Professor and Deputy Chair with the Department of Computer Science, Shanghai Jiao Tong University. Prof. Chen has published more than 200 papers in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing and performance evaluation. He has also served on technical program committees of numerous international conferences. He is a member of the IEEE Computer Society.