BlindDate: A Neighbor Discovery Protocol

Keyu Wang, *Student Member, IEEE*, Xufei Mao, *Member, IEEE*, and Yunhao Liu, *Senior Member, IEEE*

Abstract—Many wireless applications urgently demand an efficient neighbor discovery protocol to build up bridges connecting user themselves or to some service providers. However, due to intrinsic constraints of wireless devices, e.g., limited energy and error of clock synchronization, there is still absence of effective and efficient neighbor discovery protocols in the literature. In this work, we propose neighbor discovery protocols for the following two problems. First, we study *Asynchronous Symmetry Neighbor Discovery* problem, in which potential neighbor devices with asynchronous time clocks but the same duty cycle aim to find each other. Second, we propose an efficient protocol (utilizing *Bouncing* strategy) named *BlindDate*with guaranteed worst-case performance $\frac{9}{10}(1+\delta)^2x^2$ where δ is a small fraction of the length of a time slot unit and $\frac{1}{x}$ is the duty cycle. Third, we extend this strategy to address *Asynchronous Asymmetry Neighbor Discovery* problem, in which both the time clock and the duty cycles of potential neighbors are considered to be heterogeneous. We conduct extensive experiments and simulations to examine the feasibility and efficiency of the proposed protocols, and results show that *BlindDate* greatly outperforms existing approaches in average-case. Compared with known protocols, *BlindDate* also achieves a better worst-case discovery latency bound (e.g., 10 percent performance gain comparing with Searchlight [1]).

Index Terms—Neighbor discovery, energy efficiency, latency

1 Introduction

Recently, social network applications gradually move from the Internet to mobile devices and many mobile applications (e.g., Sony's Near [2] and Nintendo's StreetPass [3]) have gained increasing attention. One common properties for these kind of mobile applications is that they run locally without support of a central server as long as they are closely enough. For instance, shop owners can distribute their booklets to potential customers when the latter walk into the store or pass by. Considering convenience, owners would favor this: discovering neighbors and giving out their E-booklets automatically. For the other side, customers have controls on receiving them, i.e., turning off this service when they are not willing to be disturbed. With no doubt, this requires that devices of shop owners can find devices of customers efficiently since nobody knows how long customers would stay, which is considered to be a typical neighbor discovery problem. Indeed, the discovery protocols are necessary for many proximate-based applications, such as some portable games, social community apps, and etc.

Let us take another real application scenario as an example. In our ongoing project, CitySee [4], more than $1{,}000$ Telosb sensor nodes have been deployed in the urban area to monitor the environment. In purpose of network diagnosis, network administrators sometime have to carry a handset device to collect packets from deployed sensor nodes directly. Since all deployed wireless sensor nodes are powered by batteries and the duty-cycle in CitySee is around 4 percent, which means nodes fall into sleep most of time, clearly, an efficient neighbor discovery is necessarily needed between deployed wireless sensor nodes and the mobile handset. In addition, another work [5] proposed by Liu et al., points out that a well-designed topology control approach is determined by efficient neighbor discovery strategies to some extent since letting wireless nodes find each other effectively is the precondition of building connections.

Based on aforementioned scenarios, it is not hard to conclude that among all factors which restrict the efficiency of neighbor discovery strategies, energy constraint and low latency requirements are two particularly dominant and contradictive aspects. On the one hand, wireless communication components, like Wi-Fi and 3G, are considered to be the most energy-consuming modules in a mobile device such that most mobile devices only periodically turn their radios on when needed [23], which may increase the latency of neighbor discovery strategies to some extent. On the other hand, due to the mobility of mobile devices, due to the mobility of devices, a high latency usually causes a fail handshaking between users.

So far, most known neighbor discovery strategies known in the literature aim at solving either energy efficiency, or latency efficiency, or both problems, which basically fall into two categories: probabilistic-based protocols and deterministic-based protocols. For instance, "Birthday" [6] is a typical probabilistic protocol, which determines whether turning the radio on or shutting it down based on probabilities defined *in priori*. Usually, probabilistic protocols perform well in a static network with large numbers of devices. However, they have a relatively bad performance, especially when the number of potential neighbors is small since they fail to guarantee latency in the worst-case. Compared with probabilistic ones, deterministic protocols usually have better performance in worst-case scenario. For

- *K. Wang is with the Department of Computer Science & Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR. E-mail: kwangad@cse.ust.hk.*
- *X. Mao and Y. Liu are with the School of Software, TNLIST, Tsinghua University, Room 215, East Main Building, Tsinghua University, Beijing 100084, China. E-mail: {xufei, yunhao}@greenorbs.com.*

Manuscript received 5 Nov. 2013; revised 20 Jan. 2014; accepted 21 Feb. 2014. Date of publication 7 Apr. 2014; date of current version 6 Mar. 2015.
Recommended for acceptance by J. Chen.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2316159

instance, Quorum-based protocols[7], [8], Disco [9] and U-Connect [10] gradually approach the worst-case energy-latency product with 1.5-approximation ratio [11] of the optimal [10]. The common idea is to utilize intrinsic properties of prime numbers to guarantee the existence of the overlap between active slots of potential neighbors, in which devices turn on their radios and are ready to handshake with others. Different from those protocols, Searchlight proposed in work [1] brings us a new view of deterministic protocols, which introduces *anchor-probe* active slots scheme. Basically, two devices can find each other with a shorter latency through letting a moving *probe* slot of one device periodically and dynamically scans the *anchor* slot of the other device. Through their analysis, Searchlight further approaches the worst-case latency by nearly 50 percent by adapting a strip strategy while improving the average performance as well.

Enlightened by SearchLight, we propose a deterministic neighbor discovery protocol named *BlindDate*, which maintains multiple *probe* slots with different moving directions such as to increase the chances of the overlaps between two *probe* slots, hence, further shorten the discovery latency. Compared with other known protocols, *BlindDate* protocol shortens worst-case discovery latency bound and achieves a much better average performance. The major contributions of this study are summarized as follows:

- We propose a general model for deterministic asynchronous neighbor discovery strategies, and further prove its feasibility with detailed analysis.
- Based on the presented model, we design and implement the *BlindDate* protocol, which further achieves a performance gain by 10 percent (proved).
- We test all the cases by both simulations and real implementations for both proposed protocols and other well known neighbor discovery protocols, and the results show that *BlindDate* performs well with a $30 \sim 40$ percent performance gain in average.

The rest of this paper is organized as follows. In Section 3, we present the design model of protocols, following which we prove its feasibility mainly through illustrating the strategies of major parameters selections in Section 4. We give the details of the *BlindDate* protocol and show the worst-case latency bound in comparison with other existing neighbor discovery protocols in Section 5. We introduce the implementation of both the prototypes and simulations and show extensive evaluation results in Section 6. Some important future work is discussed in Section 7. Section 8 reviews some state-of-art related work. Finally, we conclude our work in Section 9.

2 Preliminary of Neighbor Discovery

Our work is based on a general model [1], [10] in the literature. Time is chopped into well-defined slots with an equal length. In each time slot, a wireless node (device) can choose either turning on its radio (called "active" state) for data exchanging or shutting down the radio (named "sleep" state) in order to save energy. Devices broadcast beacon messages at the beginning and the ending of each active slot separately, while keeps listening others' beacon messages for the remaining time of this active slot. This setting simplifies the discovery process so that neighbor discovery occurs as long as active time slots of two potential neighbors have overlap. Since time clocks of different devices are usually not synchronous (e.g., due to software synchronization precision or hardware clock drifts, or both) which influences the efficiency of neighbor discovery strategies directly, we focus on solving slotted asynchronous neighbor discovery protocols.

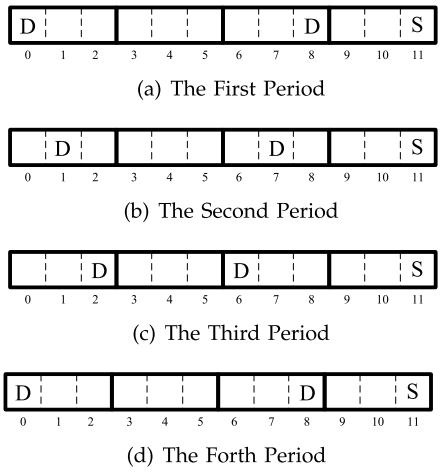


Fig. 1. Example of Model with $m = 4$, $k = 2$ and $s = 3$. D (resp. S) denotes the Dynamic slot (resp. Static slot).

Definition 1 (Blind Date). *For any two potential neighbor devices A and B, the sufficient condition for them to discover each other iff one device A (resp. B) which is in the active state hears at least one beacon message (or say probe message) from the other device B (resp. A). A successfully neighbor discovery progress between two devices as a successful* Blind Date.

In this work, we aims to solve the following two problems.

1. *Asynchronous symmetric problem*, in which all wireless devices are assigned with the same discovery schedule regardless of asynchronous time clocks and different starting time while keeping both discovery latency and energy consumption as optimization objectives.
2. *Asynchronous asymmetric problem*, in which all wireless devices can use individual discovery schedules, i.e., different duty cycles and asynchronous time clocks, etc.

3 Design Overview

3.1 Model

In order to describe the protocol model clearly, we first use the case shown in Fig. 1 as an example to illustrate our main idea.

Let t denote one schedule period of a wireless device, we further divided t into m blocks, each of which consists of s time slots, i.e., $t = m \cdot s$. In the example shown in Fig. 1, we have $m = 4$ and $s = 3$ such that each period contains $m \cdot s = 12$ time slots, whose sequence numbers are defined from 0 to 11 with step of 1. Here, we use bold boxes to represent the blocks and use dotted lines to split the block into time slots (see Fig. 1a for details).

TABLE 1
Summary of Major Symbols

Symbol	Description
t	Length of a period
T	Hyper-period
m	Number of blocks in a period
s	Number of time slots in a block
L	Worst-case discovery latency
S	Static active time slot
D	Dynamic active time slot
$P(x)$	Position of the time slot x

Next, we design two types of active slots. The first type is called *Static* active slot (abbreviated to *Static* slot and denoted by letter S in uppercase) which has fixed position of the *Static* slot at the last time slot (i.e., the position 11 in the example) of each period. The second type is called *Dynamic* active slot (abbreviated to *Dynamic* slot and denoted by letter D in uppercase). Different from the *Static* slot, the *Dynamic* slot could have different positions for different periods, In BlindDate, the position of Dynamic slot, continues to move from one side to the other inside within one block. The main parameters used in this work are summarized in Table 1.

Moving patterns of dynamic slots. According to the sufficient condition for two potential neighbors to find each other successfully, we have to make sure that there is overlap between the active slots of two potential neighbors. In order to achieve this, we propose to use two types of Dynamic slots with opposite moving directions. One Dynamic slot continuously change its position at each time period, i.e., moving from left to right with step of 1. For instance, it first occupies the time slot 0 in the first period (see Fig. 1a), and in the second period, the *Dynamic* slot moves to occupy the right adjacent time slot 1 (see Fig. 1b). It continues this moving until reaching the right-most time slot of the block (shown in Fig. 1c). Then it restarts from the left-most time slot and repeats this pattern (please refer to Fig. 1d). Hence, if we use $P(D^i)$ to denote the position of the *Dynamic* slot for the ith period, the *Dynamic* slot locating in the jth block can be computed as

$$P(D^{i+1}) = ((P(D^i) + 1)\ mod\ s) + (j-1) \cdot s. \quad (1)$$

Similarly, the position of the other Dynamic slot with opposite moving direction (the third block's *Dynamic* slot in the example), is as follows.

$$P(D^{i+1}) = ((P(D^i) - 1)\ mod\ s) + (j-1) \cdot s. \quad (2)$$

Since the number of Dynamic slots in one time period is directly related to the energy consumption of a device, in our design, we restrict that every block could possess at most one *Dynamic* slot and *Dynamic* slots in different blocks can use opposite moving directions, i.e., either from left to right or from right to left. We further call the block owning a *Dynamic* slot as a *Dynamic* block.

Protocol model. First, we choose k block(s) out of total m blocks as *Dynamic* blocks to insert a *Dynamic* slot. In the example shown in Fig. 1, k equals to 2 as we insert a *Dynamic* slot into the first and the third block respectively. Next, counting with *Static* slot together, a wireless device has $k+1$ active slots totally for each time period t including one *Static* slot and k *Dynamic* slot(s). Hence, for any two devices A and B, as long as one of the following cases happens, *Static-Static* overlap, *Dynamic-Static* overlap and *Dynamic-Dynamic* overlap, we consider that A and B could successfully find each other.

Model feasibility. According to the Equations (1) and (2), the schedule repeats every s periods following the pattern designed, here s is the number of time slots in a block. In this example shown in Fig. 1, s equals to 3 such that a wireless device in the status shown in Fig. 1d goes into the status shown in Fig. 1a. This period is called as the *hyper-period* T, which equals to the product of period t and slots number s of each block, i.e., $T = t \cdot s$.

As stated previously, the discovery occurs when three types of overlap appear: *Static-Static* overlap, *Dynamic-Static* overlap and *Dynamic-Dynamic* overlap. In other words, if the period offset between neighbors are exactly matched or shifts slightly, a *Static-Static* overlap appears. Along with the offset increases, the *Static* slot may locate within the *Dynamic* block of neighbors'. If so, within the *hyper-period* of scheduling, the *Dynamic* slot scans over the whole block. Hence, this *Dynamic* slot overlaps with neighboring device's *Static* slot at least once during the scanning, which is a *Dynamic-Static* overlap. Besides, neighboring devices' *Dynamic* slots could have an overlap when they move in different directions but under a precondition that the *Dynamic* blocks of these two slots have an overlap and both *Dynamic* slots appear in the overlap region at sometime. This precondition ensures the *Dynamic-Dynamic* overlap. To fulfill this precondition, the *Dynamic* blocks overlap must cover over 50 percent length of the block-before one *Dynamic* slot leaves this overlap region, so that the other *Dynamic* slot enters this region.

Unfortunately, if there exists a case of offset that none of these three overlap is promised, the model encounters a fail situation. Correspondingly, a feasible protocol under this model is that for all offset cases at least one kind overlap exists.

3.2 Model Discovery Latency

As we have mentioned before, the schedule patterns of a wireless device repeats for every *hyper-period* T such that the necessary condition for any two devices A and B (having the same *hyper-period* T) to find each other is that their active states have some overlap in T. Otherwise, they may miss each other. Hence, the upper bound of the worst-case discovery latency (assuming L) of this model should not exceed $T = s \cdot t = ms^2$ in order to work feasibly, i.e.,

$$L = ms^2.$$

With no doubt, utilizing more *Dynamic* blocks for a known m could reduce the discovery latency to some extent. However, we cannot neglect the fact that this

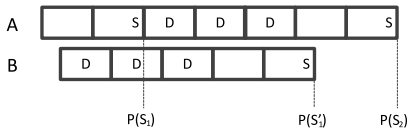


Fig. 2. Select first k blocks.

increases the energy consumption at the same time since there are more active time slots in each period. As a result, in order to have the same duty cycle, every block needs to contain more time slots (i.e., a larger s), which may lead to the increment of discovery latency. Thus, for the same duty cycle and a known m we first express the worst-case discovery latency as functions of x, where $\frac{1}{x}$ is the duty cycle of the protocol. Clearly, we have $\frac{k+1}{m \cdot s} = \frac{1}{x}$ such that

$$L = m \cdot s^2 = \frac{(k+1)^2}{m} \cdot x^2. \tag{3}$$

From the Equation (3), it is not hard to conclude that for the same duty cycle and a known m, utilizing more *Dynamic* blocks increases the worst-case discovery latency.

4 Major Parameter Selection

We first solve the symmetry case, i.e., all nodes have the same length of period time t. Later we will show how to extend this solution to the asymmetric case. Notice, the following analysis and proofs are based on the symmetry case.

Through the Equation (3), we know that the discovery latency L decreases along with the decreasing of k for the known m. Next, we show the relationship between k and m in order to guarantee the success of neighbor discovery. We analyze this case by case.

4.1 When $k = m$

When $k = m$, i.e., every block contains a *Dynamic* slot so that for any two devices, the *Static* slot of one device must have overlap with one *Dynamic* block of the other device no matter what offset difference they have. Clearly, in this case, two devices can find each other within s periods with a hundred percent guarantee.

4.2 When $\lfloor \frac{m}{2} \rfloor < k < m$

For the case $\lfloor \frac{m}{2} \rfloor < k < m$, we design an easy selection strategy ensuring the success of neighbor discovery (see Fig. 2 for illustration).

Basically, for two devices A and B, we select the first k blocks as their *Dynamic* blocks. Obviously, if the *Static* slot of node B locates in the first k blocks of node A, as the same reason for the first case, the *Dynamic* slot in that block definitely has an overlap time with B's *Static* slot within s periods; Otherwise, the *Static* slot of node B locates outside of the first k blocks, which means $P(S'_1) - P(S_1) > s \cdot k$. Since $P(S_i)$ denotes the position of S_i and $P(S_2) - P(S_1) = t = m \cdot s$, we have

$$\begin{aligned} P(S_2) - P(S'_1) &= m \cdot s - (P(S'_1) - P(S_1)) \\ &< (m-k) \cdot s. \end{aligned}$$

As we know that $m - k < k$ due to $k > \lfloor \frac{m}{2} \rfloor$, which indicates that A's *Static* slot locates in the first k blocks of B. Hence, they also can definitely discover each other within s periods.

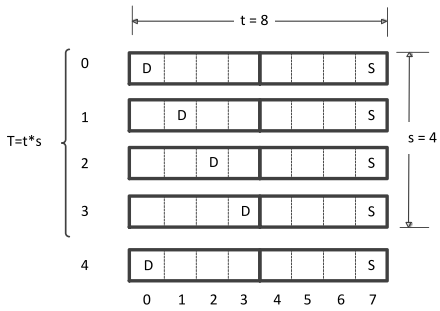


Fig. 3. Optimal even case.

4.3 When $k = \lfloor \frac{m}{2} \rfloor$

For simplicity of analysis, we consider m by the following two cases, an even number or an odd number, i.e., we analyze cases $m = 2p$ or $2p + 1$ separately. Substituting m with $2p$ and $2p + 1$ respectively, we have

$$L = \frac{(k+1)^2}{m} \cdot x^2 = \frac{(p+1)^2}{2p} \cdot x^2 \tag{4}$$

and

$$L = \frac{(k+1)^2}{m} \cdot x^2 = \frac{(p+1)^2}{2p+1} \cdot x^2. \tag{5}$$

Since both equations imply that L goes smaller when p gets smaller, we focus on finding a smaller p.

4.3.1 When m Is an Even Number

We select the first $k = p$ blocks as *Dynamic* blocks. Using the same analysis in case $k > \lfloor \frac{m}{2} \rfloor$, we have

$$\min\{P(S'_1) - P(S_1),\ P(S_2) - P(S'_1)\} \leq \frac{m \cdot s}{2},$$

which points out that at least one node's *Static* slot locates in the first half blocks ($k = p$ blocks) of its neighbor. Within s periods, the *Dynamic-Static* overlap happens. Referring to the Equation (4), when m is an even number, the worst-case discovery latency L achieves the minimal value when we assign $p = 1$ (see Fig. 3 for illustration).

Actually in this case, the worst-case discovery latency L equals to $2x^2$, which is optimal.

4.3.2 When m Is an Odd Number

Intuitively, the smallest value for Equation (5) is when $p = 1$, which means that select one *Dynamic* block out of total three blocks. Unluckily, we cannot find a feasible solution which can guarantee the success of *Blind Date* when $p = 1$. Rationally, we choose the value of p as 2.

Fortunately, after carefully arranging the position of *Dynamic* slots, the proposed *BlindDate* protocol can guarantee the success of neighbor discovery with latency bound by choosing 2 blocks out of five blocks (i.e., $k = 2$ and $m = 5$). One thing deserves mentioning that the worst-case discovery latency of this specific *BlindDate* protocol ($k = 2$ and $m = 5$) is better than all cases when m is even.

In the following Section 5, we provide design details of *BlindDate* protocol along with proved performance.

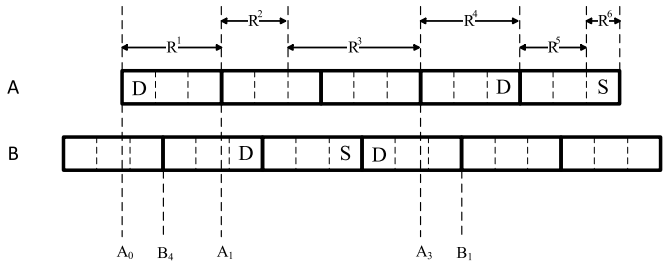


Fig. 4. Feasibility analysis.

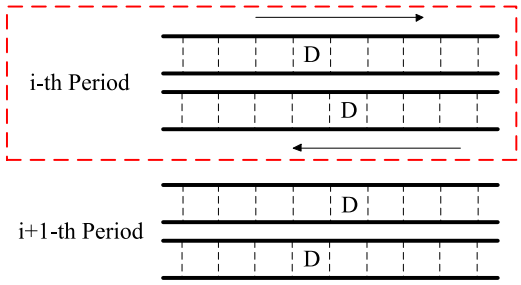


Fig. 5. *Dynamic* slots jump over.

5 BlindDate Protocol

BlindDate protocol (abbreviated to BD protocol) allocates *Dynamic* slots in the first and the forth blocks respectively when we put *Dynamic* slots inside of two out of five blocks. Specifically, the first block's *Dynamic* slot moves from left to right while the *Dynamic* slot in the fourth block moves in an opposite direction (see Fig. 4). Through the analysis in last section, we know that BD protocol guarantees the existence of active slots overlap for all offset cases. For the proof of the BD protocol's feasibility, please refer to the conference version [24] for details. Next, we will show that after adopting the bouncing *Dynamic* slot, BD protocol's performance is further improved. Compared with other known protocols for symmetric case, BD protocol achieves the smallest worst-case latency bound (proved theoretically in this section). Meanwhile, BD is also suitable for asymmetric case.

5.1 Feasibility

To illustrate the feasibility of BD protocol, we have to prove that two potential neighbor devices can find each other no matter what scheduling offset they have. Since the schedule pattern repeats every *hyper-period*, it is sufficient to consider the scheduling offsets within the *hyper-period*. Next, we use a time slot offset with a period offset to represent the scheduling offset. Here, the time slot offset indicates the offset between the start points of two potential neighbor nodes' *Static* slots'. Obviously, in symmetric case, the time slot offset remains the same for all periods. On the other hand, the period offset indicates the period difference between two potential neighbor devices. For instance, when one device A starts the protocol, its potential neighbor device B may have started the protocol for d periods already (d is a constant). Thus, the device A has its *Dynamic* slot in the first time slot of the block while the other device B has its *Dynamic* slot in the $d+1$ time slot (assuming the time slot number of each block is bigger than $d+1$). The period offset for this instance is d.

As Fig. 4 shows, we split the period t of device A into six regions for the convenience of proof. When we claim that a slot locates in region X we mean that the start point of this slot, especially the left edge of the time slot, locates in region X. Here, our partition is based on different time slot offsets. In the following proof, we show that BD protocol is feasible for all period offsets in these six regions one by one. In addition, since we have no need to take slot alignment as a condition of the proof, our proof applies to both slot alignment and non-alignment cases.

First of all, in Fig. 4, if the device B's *Static* slot locates in region R^1 or R^4, there exists a *Dynamic-Static* overlap because the device A's *Dynamic* slot cruises all slots in these regions during s periods. Second, if the device B's *Static* slot locates in region R^2 or R^5, the device A's *Static* slot locates in the fourth or the first block of device B, and we clearly have a *Dynamic-Static* overlap within s periods. Next, when the device B's *Static* slot locates in region R^6, the *Static* slots of A and B obviously have a *Static-Static* overlap since this region has only one time slot. Then we handle the most complex case when the device B's *Static* slot locates in region R^3. Fortunately, the opposite moving directions of *Dynamic* slots benefit us here. For ease of presentation, we first mark several positions, A_0, A_1, A_3, B_1 and B_4 (shown in Fig. 4) where the letter in upper case indicates which device it belongs to and the subscript represents the block it locates at the beginning of. With no doubt, the *Dynamic-Dynamic* overlap must happen inside those block overlap regions if the overlap exists, as the *Dynamic* slots go through these regions in the opposite directions. If both of two devices' *Dynamic* slots are in these regions at some time, the *Dynamic-Dynamic* overlap certainly happened already or will happen later. Thus, the remaining work is to prove that there exists a moment at which both devices' *Dynamic* slots are in the overlap region.

Since $(A_1 - B_4) + (B_1 - A_3) = s$, we have $\max\{A_1 - B_4, B_1 - A_3\} \geq \frac{s}{2}$. Without loss of generality, we assume that

$$A_1 - B_4 \geq \frac{s}{2}. \tag{6}$$

From Equation (6), we conclude that the device A's *Dynamic* slot spends no less than $\lceil \frac{s}{2} \rceil$ periods in the overlap region. Before the device A's *Dynamic* slot leaves the overlap region, the device B's *Dynamic* slot surely has entered into the overlap region or they are aligned at the different sides of position A_1. For the purpose of ensuring that those *Dynamic* slots have an overlap, we prolong the length of *Dynamic* slot to $(1+\delta)$ and shorten the adjacent time slot to $(1-\delta)$ so that we keep the period t unchanged. This modification also solves the problem of alignment case (see Fig. 5 for illustration), in which two *Dynamic* slots may jump over each other without an overlap. Later, we will move this prolonged length to other position and obtain an even better performance. From the above proof, we conclude that our protocol guarantees the performance for all cases.

5.2 Bouncing Dynamic Slot

When the slots are non-alignment, every active slot of a wireless device probably has overlaps with two sequential time slots of its potential neighbor's, the question is that "can we let the Dynamic slot skip one time slot for each movement and the results are still good?" The answer is

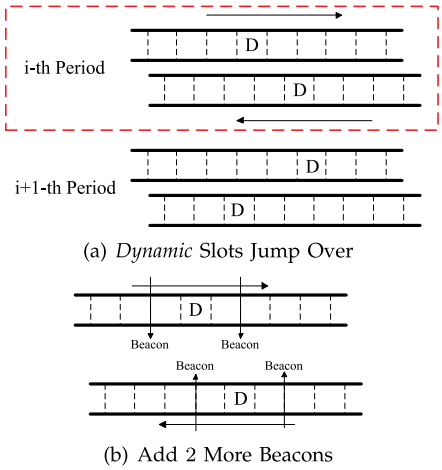


Fig. 6. Bouncing *Dynamic* Slot modification.

affirmative. In BD protocol, we use a bouncing *Dynamic* slot, which keeps moving with step of 2. In other words, if P_i represents the position of the *Dynamic* slot in the ith period, the *Dynamic* slot locates in the jth block and each block contains s slots, then

$$P(D^{i+1}) = ((P(D^i) + 2) \ mod \ s) + (j-1) \cdot s.$$

Similarly, for the *Dynamic* slot with the opposite moving direction, we have

$$P(D^{i+1}) = ((P(D^i) - 2) \ mod \ s) + (j-1) \cdot s.$$

It is not hard to see that if this improvement works, it reduces the *hyper-period* T by nearly 50 percent, hence decreases the worst-case discovery latency by nearly 50 percent. The new *hyper-period* T is as follows:

$$T = \left\lceil \frac{s}{2} \right\rceil \cdot t = \left\lceil \frac{s}{2} \right\rceil \cdot 5s.$$

However, the bouncing *Dynamic* slot brings us a big challenge at the same time, since two bouncing *Dynamic* slots of opposite directions may jump over each other without overlaps in some case (refer to Fig. 6a). To overcome this issue, we shift the prolonged length of time to the beginning of the previous slot and the end of next slot of current *Dynamic* slot (see Fig. 6b). From the system implementation's point view, typically, each prolonged time length should cover the time of sending a single beacon message. Fortunately, these beacon messages are on the purpose of informing the existence such that it takes much smaller time compared with the regular time slot. With a small duty cycle, the probability of conflicting between beacon messages stays low, which will not affect the success rate of discovery much.

TABLE 2
Latency Bound for Deterministic Protocols

Protocol	Parameter	Duty Cycle	Latency Bound
Disco [10]	p_1, p_2	$\frac{p_1+p_2}{p_1 \cdot p_2}$	$p_1 \cdot p_2$
U-Connect [11]	p	$\frac{3p+1}{2p^2}$	p^2
Searchlight [1]	t	$\frac{2 \cdot (1+\delta)}{t}$	$t \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil$
BlindDate	s	$\frac{3(1+\delta)}{5s}$	$5s \cdot \lceil \frac{s}{2} \rceil$

TABLE 3
Latency Bound under the Same Duty Cycle

Protocol	Latency Bound	Expressed by x
Disco	p^2	$4x^2$
U-Connect	$\frac{3p+1}{2p^2}$	$\frac{9x^2}{4}$
Searchlight	$t \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil$	$(1+\delta)^2 x^2$
BlindDate	$5s \cdot \lceil \frac{s}{2} \rceil$	$\frac{9}{10}(1+\delta)^2 x^2$

5.3 Latency Bound Comparison

In every period t, BD protocol has three active time slots, one *Static* slot and two *Dynamic* slots. If we use $(1+\delta)$ to represent the total length of a *Dynamic* slot. The duty cycle for our protocol approximatively equals to $3(1+\delta)/5s$. Next, we compare the worst-case discovery latency of our protocol with those of other existing deterministic protocols from theoretical point of view. Table 2 summarizes the worst-case latency bound for some known deterministic protocols.

Here, we assume all protocols use the same duty cycle as $\frac{1}{x}$ in order to compare them fairly. For ease of comparison, we make some assumptions that $p_1 = p_2 = p$ for Disco, such that

$$\frac{1}{x} = \frac{p_1 + p_2}{p_1 \cdot p_2} \approx \frac{2}{p}.$$

Clearly, when we express the parameter as a function of x, we have $p = 2x$. Hence, for U-Connect, we have the following result

$$\frac{1}{x} = \frac{3p+1}{2p^2} \Rightarrow p \approx \frac{3x}{2}.$$

And for Searchlight, the following equation is not hard to get.

$$\frac{1}{x} = \frac{2 \cdot (1+\delta)}{t} \Rightarrow t = 2x(1+\delta)$$

Finally, for our BD protocol, we have the following equation

$$\frac{1}{x} = \frac{3(1+\delta)}{5s} \Rightarrow s = \frac{3}{5}x(1+\delta)^1.$$

Then, the worst-case discovery latency for all compared protocols can be expressed by functions of x (see Table 3). Clearly, after Searchlight improved the worst-case latency by almost 50 percent, we further achieve a performance gain by 10 percent. BD protocol has the smallest worst-case bound among existing protocols.

5.4 Asymmetric Case

In a real application scenario, a mobile device may dynamically adjust it's duty cycle continuously, e.g., depending on

1. Here, the slot length increment ratios of Searchlight and BlindDate are same because they are for the same purpose, which covers the time of broadcasting beacon messages. In order to preventing the conflicts between beacon messages, the increment of Searchlight should be at least twice large as the size of beacon for the well-aligned case.

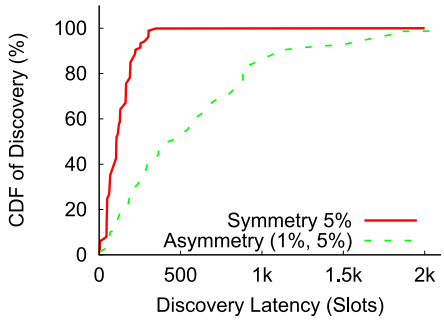


Fig. 7. Primary implementation.

energy conditions, so that two potential neighbor devices may use different duty cycles. Clearly, in this asymmetric case, offset between the period's starting time of two devices (to be precisely, the starting time of scheduling periods) changes along with the time. Without doubt, this brings challenges to guarantee the performance of BD protocol, and makes it harder to specify a worst-case latency bound. Next, we propose an approach to overcome this issue so that BD protocol can be extended to support the asymmetric case as well.

For a group of devices, we let the block of the device with highest duty cycle contain s time slots, and the blocks of other devices contain $s \cdot 5^t$ time slots, where t is a positive integer. Hence, the offset between devices keeps unchanged as the larger period is a multiple of the smaller one. This promises the success of our BD protocol. Here, we provide a loose worst-case bound. Assuming that the blocks of the device A contain s_A time slots and the blocks of the device B contain s_B time slots where $s_A = s_B \cdot 5^t$. Within $s_B \cdot 5$ periods, A's *Dynamic* slot will cover a whole period of B as the offset is unchanged. During this time, A's *Dynamic* slot and B's *Static* slot definitely have an overlap. So, we have,

$$L = s_B \cdot 5 \cdot t_A.$$

Combining with the bouncing *Dynamic* slot, the worst case discovery latency is shortened by nearly 50 percent. Then,

$$L = \left\lceil \frac{s_B \cdot 5}{2} \right\rceil \cdot t_A. \tag{7}$$

One thing deserves mentioning that other protocols using our design model can also adopt the same approach in order to solve asymmetric case by assigning the time slot number of blocks as $s_{base} \cdot m^t$ where t is a positive integer. The analysis is similar to that of BD protocol. Since we only give a loose worst-case bound of BD protocol for asymmetric case, we believe that the real performance of BD protocol is much better than this bound when the possibility of *Dynamic-Dynamic* overlap is considered. In fact, the simulation results in Section 6 also validate our conjecture.

6 Performance Evaluation

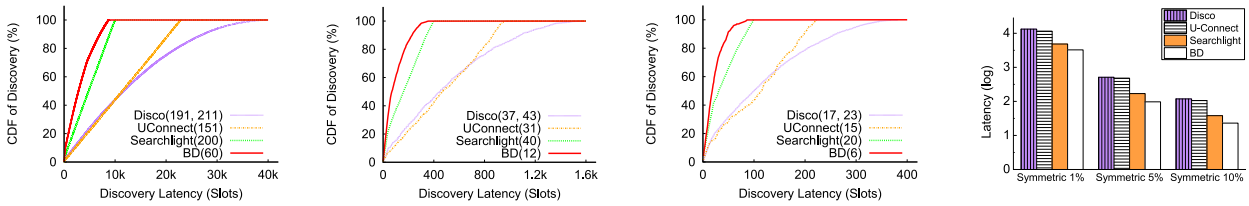
6.1 Primary Prototype Implementation

We first implement a prototype of BD protocol on a real testbed consisting of TelosB nodes. On one hand, experimental results verify the feasibility and efficiency of our protocol for both symmetric and asymmetric cases. On the other hand, they guide us to conduct extensive simulations in order to further examine the performance of BD protocol and to make a thoroughly comparison with other existing protocols.

BD protocol is implemented on TelosB [21] sensor nodes working on $802.15.4$ using nesC [22] programming language. The length of a single time slot is set to be 100 ms. Note that if the BD protocol is applied to devices using Wi-Fi, the length of a single time slot should be prolonged properly [1] because of the turning up delay constraints of the Wi-Fi interface. We deploy two sensor nodes within the transmission range representing two potential neighbors respectively, both of which have been programmed with BD protocol. Since BD is an asynchronous neighbor discovery protocol, we let two sensor nodes start BD protocol randomly at different time for each test case. In real application scenario, devices may have started BD protocol for a while before meeting a neighboring device. We record the duration from the moment that both nodes have run the BD protocol to the moment that one sensor node receives the beacon message from the other as the real discovery latency. Besides the discovery latency, we record the time that the wireless radio is on. Hence, based on those records we are able to compute the real energy consumption approximately.

For both symmetric and asymmetric cases, the experiments are repeated for over hundreds of times. A CDF graph shows the experiment results for both cases (see Fig. 7) and the average-case performance including the discovery latency and the real radio-on time are recorded. For the symmetric case, the duty cycle is set to be 5 percent so that the length of a block is $s = 12$. The BD protocol averagely takes $11{,}824$ ms, which is around 119 time slots, for one node to discover its neighbor. The total radio-on time is 628 ms averagely. Thus, the real duty cycle is slightly more than 5 percent and the increment δ is small. For the asymmetric case, the duty cycles are set to be 5 and 1 percent for two nodes. The length of a block for two nodes are 12 and 60 respectively. The experiment result shows that BD protocol takes averagely $60{,}037$ ms, which is around 600 time slots, for one node to find the other. And for the node with larger duty cycle, it turns radio on for $3{,}321$ ms averagely. Among all test cases, there is no failure situation. When both nodes work at 5 percent duty cycle, the largest discovery latency is less than $36{,}000$ ms. And when one works at 1 percent duty cycle with another working at 5 percent duty cycle, this latency prolongs. And the real implementation performs better than our theoretical analysis to some extent. We claim that BD protocol is suitable and feasible for both symmetric and asymmetric cases. Also, dividing the real radio-on time by the total time period, we find that the two additional beacons do not increase the duty cycle much (less than 0.5 percent). Hence, this solution is worthy in return for improving the performance.

The experiment results display a basic performance of BD protocol and light our directions to perform extensive simulations to further examine the performance of BD protocol. Next, a series of simulation is completed in order to test the performance of BD protocol. From the simulation results, a comprehensive view of the protocol displays.



(a) CDF for Duty Cycle 1% (b) CDF for Duty Cycle 5% (c) CDF for Duty Cycle 10% (d) Average Performance

Fig. 8. Discovery latency for symmetric case.

6.2 Simulation

BlindDate protocol is programmed using Java simulator. As we know, clock synchronization is not always true for mobile devices due to energy-constraint, neighbor devices may start at different time. Indeed, even for the symmetric case, the offset between nodes' start points exists. This offset has a uniform distribution in the real application scenario so that each unique offset repeats for equal times. The evaluation metric (discovery latency) represents the time passed from the potential neighbors entering the transmission range to the first discovery event (as we defined in Section 3). To simulate the scenario where a node A enters into the transmission range of the other node B, a random number is involved to represent the time that the later starting node has spent before entering the transmission range of the other node. In order to improve the accuracy, a time slot is chopped into 10 mini time slots equally. Only when two beacon messages occupy the same mini time slot, it makes the conflicts and results failure of discovery. In addition, both the alignment case and the non-alignment case occurs in the simulation at the same time. Since different platforms (protocols) may have different width of a time slot, for all results shown, the number of time slots collapsing represents the measurement index, rather than real running time.

To compare all deterministic protocols fairly, we simulate all protocols under the same duty cycle such that the discovery latency also indicates the total energy consumption. It is worth mentioning that, for Searchlight, we implement the version with Strip strategy, which achieves the best performance among all Searchlight families. All protocols we compared repeat their schedules every *hyper-period* T, which may be different due to the fact that they have the same duty cycle. Hence, the simulation offset is in the range of $[0, T-1]$. Since we focus on shortening the discovery latency for neighbor discovery protocols, the evaluation metric is the discovery latency in number of time slots.

6.2.1 Symmetric Case

For the symmetric case, we simulate all protocols for three different duty cycles, 1, 5 and 10 percent respectively. The results are shown in Fig. 8 and the parameters used for each protocol are also shown on the right-bottom of the CDF graphs. Clearly, for all three different duty cycle cases, BD protocol achieves the best performance among all protocols, and the worst-case discovery latency of BD protocol is around 10 percent smaller compared with Searchlight.

In addition, the average performance of BD protocol is $30 \sim 40$ percent better than that of Searchlight.

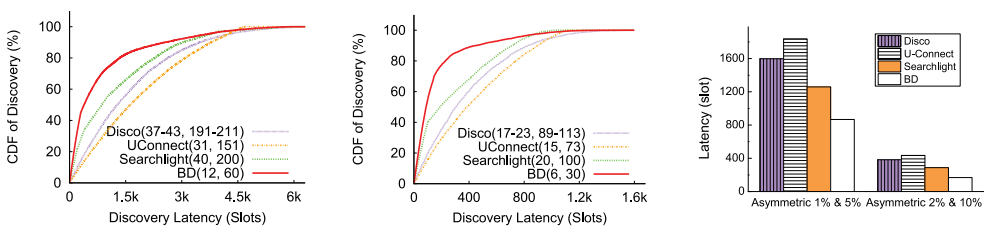
6.2.2 Asymmetric Case

For asymmetric case, the worst-case performance of BD protocol is smaller than the theoretical results as shown in Fig. 9. Especially, BD protocol shows a much better average-case performance: $30 \sim 40$ percent performance gain.

Notice that in Fig. 9, the lines indicating the performance of BD protocol and Searchlight cross with each other when the discovery ratio is over 95 percent. This is due to the fact that the length of period of Searchlight is smaller than that of BD protocol so the upper bound of the worst case of Searchlight is smaller than that of BD protocol for most of situations, BD protocol discovers neighbor faster than all other protocols.

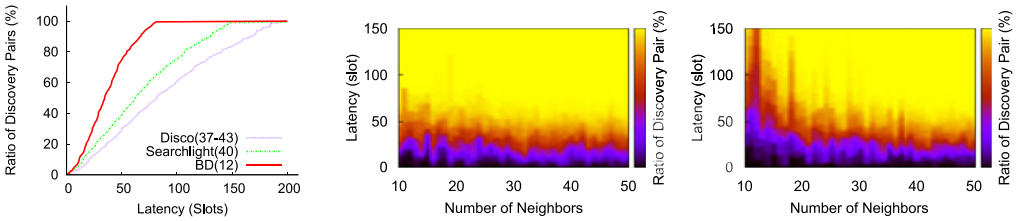
6.2.3 When Multiple Neighbors Exist

In a real application scenario, a group of users may co-exist in the same area and desire to discover each other. It is reasonable to assume that the neighbor information is deliverable among users such that one device could share its neighbor information with the new neighbor. Otherwise, without the memorable neighbor information, multiple neighbors case is same as the worst pair among these neighbors. In order to keep the smaller size of beacon message, we assume that the sharing of neighbor information is completed after the establishment of the connection instead of through transcieving beacon message. We record the total number of discovery pairs between neighbors along with the increasing of time slots. First, we use a CDF graph to summarize the performance for the case of 10 neighbors co-existing, who are all working at duty cycle of 5 percent. From the



(a) CDF for Duty Cycle 1%, 5% (b) CDF for Duty Cycle 2%, 10% (c) Average performance

Fig. 9. Discovery latency for asymmetry.



(a) Multiple Neighbors of 10 (b) BD with 10-50 Neighbors (c) Searchlight with 10-50 Neighbors

Fig. 10. Multiple nodes with duty cycle 5 percent.

Fig. 10a, we can see that BD protocol achieves the best performance for worst-case and average-case. For instance, when BD protocol assists all 10 neighbors in discovering others, other protocols discover around 60 percent of neighbor pairs. Compared with single pair performance, BD protocol achieves greater improvement. This result is mainly benefited by the additional beacon messages. Hence, with the cost of a small amount of energy for sending these beacon message, BD protocol has a significant performance gain.

We further vary the number of co-existing devices from 10 to 50 with step of 1, and the results are shown in Figs. 10b and 10c. Basically, BD protocol always outperforms over other protocols although this performance gain goes to smaller with the increment of the number of potential neighbors co-exist. But in real application scenario, the number of surrounding neighbors is usually small (e.g., 30). For such amount of neighbors, BD protocol normally shortens nearly 50 percent discovery latency in average. Within 80 time slots, all devices using BD protocol successfully find each other. It indicates that a few number of neighbors could benefit the discovery process greatly, which is more common in practice.

6.2.4 Beacon Message Loss

Since, packet loss due to unstable link quality is hard to avoid, the performance of BD protocol should be tested with the consideration of transmission failures. BD protocol and Searchlight are tested for two nodes discovery case considering different packet loss ratios such as 0, 50 and 70 percent. The results in Fig. 11 show that the lossy link does not affect the performance of BD protocol much. The two nodes using BD protocol quickly discover each other for over 80 percent cases. Even for the situation when the packet loss rate is 70 percent, BD protocol can still achieve over 80 percent ratios of discovery within around 300 time slots. Compared with Searchlight, BD protocol achieves a better performance for the average-case.

7 Discussion and Future Work

In our design, the energy consumption is mainly caused by the opening of the radio interface. Since the energy used in computation is incomparable with that of the radio interface. Fortunately, those additional beacon messages only use radio interface for sending without listening, therefore they occupy short time length. Compared with the length of time slots, they are insignificant especially when the slot number of each block is large. For the case of time drift, our protocol is slightly affected. First, it has multiple *Dynamic* slots. Within the time of worst-case bound, there exists multiple overlaps between neighbors. A clock drift can eliminate one of overlaps however it hardly erase all overlaps. Second, the additional beacon messages favor us here. The failure caused by clock drift are mainly because the overlap region disappears. By favor of those beacon message, even when there is no overlap region, the active slots are still within one time slot offset.

We would like to further reduce the discovery latency under our design model: for a given value of m, select as less *Dynamic* blocks as possible. Meanwhile, because our protocol has decreased the value of k to $\lfloor \frac{m}{2} \rfloor$ and m is odd, the total length of two neighbors' *Dynamic* blocks is less than that of a period. It seems that a smaller value of k is difficult to provide the feasibility. Thus, we would like to focus on providing a bound under this model with a solid theoretical proof in the future.

Last but not least, neighbors' information could be cached for further utilization and assistance, e.g., acc and WiFlock. Through a deep analysis of those cached information, we try to dig out some neighbor patterns, like spatial and temporal correlation. This helps us assign a better schedule to find some familiar neighbors, who are more likely to cooperate with the user.

8 Related Work

Neighbor discovery problem is relatively easy to solve when the nodes are deployed in a static and dense network. Wireless devices can communicate by long preamble packets [12], synchronizing their clock [13] or repeatedly sending packets until receiving ACK packet [14]. BMAC [12] and SMAC [15] are proposed for neighbor discovery protocols employing LPL. The key point is to let potential neighbors in active state concurrently. Generally speaking, current neighbor discovery protocol basically

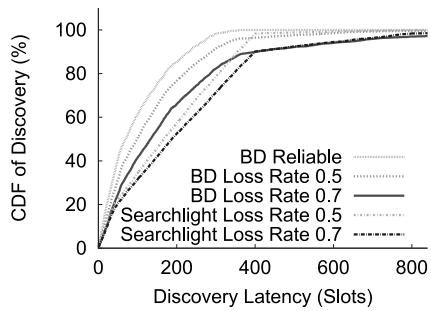


Fig. 11. Lossy link: Duty cycle 5 percent.

fall into two categories, probabilistic protocols and deterministic protocols.

"Birthday protocols" [6], is probabilistic approach for asynchronous neighbor discovery in static wireless networks. Basically, for every single node, *Birthday* sets different probabilities for sending packets, receiving packets and going to sleep. However, it cannot provide a worst-case bound, which is actually the common drawback for all probabilistic approaches.

In contrary, deterministic protocols like Quorum-based protocols [7], [8], can guarantee the worst-case bound. Quorum-based protocols consider time as a two-dimensional $m * m$ interval array and arbitrarily chooses one row and one column of intervals as active slots, which guarantees that any pair of two nodes have at least two active slots overlapped. Zheng et al. [16] use different settings to make optimal block designs for symmetric neighbor discovery with the help of the Multiplier Theorem [17]. However, their design did not consider the asymmetric case. Some other work [18] studied this problem from increasing the point view of reliable transmission services and prolonging the life time of wireless sensor networks.

Prime-based protocols one type of deterministic protocols one type of deterministic protocols, aim to solve neighbor discovery problem by providing a worst-case bound for both symmetric and asymmetric cases. For instance, in Disco [9], a node selects a pair of unequal prime numbers and comes into active state when the sequence number of a time slot is divisible by any of selected prime numbers. Kandhalu et al. propose U-Connect [10], that node selects only one prime number p. Except for those multiplies number of slots, there are $\frac{p+1}{2}$ more slots for active state.

Recently, Searchlight [20] is proposed for neighbor discovery problems, in which they further define two different types of slots for active slots, *anchor* slots and *probe* slots respectively. By keeping a *probe* slot changing its position (in the temporal dimension) periodically, Searchlight guarantees that a *probe* slot has an overlap with a potential neighbor's *anchor* slot within some time bound. Later, an improved version of Searchlight [1] utilizes a strip strategy greatly reducing the worst-case bound by nearly 50 percent. Motivated by Searchlight, we design a new deterministic protocol (named *BlindDate*) aiming at further shortening the worst-case latency by increasing the probability of overlaps.

9 Conclusion

In this work, we focus on designing an asynchronous neighbor discovery protocol named BlindDate, which guarantees a better worst-case latency bound for symmetric case and has better performance for average-case. It shortens the worst-case latency bound by exploiting *Static* and *Dynamic* slots utilizing a bouncing strategy. Analysis shows that *BlindDate* achieves a performance gain by 10 percent compared with other known work (e.g., SearchLight). We implement a prototype using real testbed consisting of wireless TelosB nodes and then perform extensive and comprehensive simulations. Both results show that *BlindDate* reduces the discovery latency by around 30 percent in average.

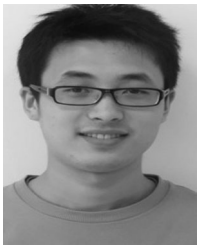
Acknowledgments

This work was supported by NSFC under grant 61272426, and 61379117, NSFC Major Program 61190110 and the NSFC Distinguished Young Scholars Program under grant 61125202.

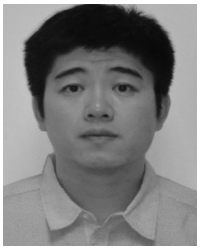
References

[1] M. Bakht, M. Trower, and R. H. Kravets, "Searchlight: Won't you be my neighbor?" in *Proc. ACM Int. Conf. Mobile Comput. Netw.*, 2012, pp. 185–196.
[2] Sony ps vita - near. (2013) [Online]. Available: http://us.playstation.com/psvita/apps/psvita-app-near.html.
[3] Nintendo 3ds - streetpass. (2013) [Online]. Available: http://www.nintendo.com/3ds/features.
[4] X. Mao, X. Miao, Y. He, X. LI, and Y. Liu, "Citysee: Urban co2 monitoring with sensors in," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2012, pp. 1611–1619.
[5] Y. Liu, Q. Zhang, and L. M. Ni, "Opportunity-based topology control in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 3, pp. 405–416, Mar. 2010.
[6] M. McGlynn and S. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. and Comput.*, 2001, pp. 137–145.
[7] S. Lai, "Heterogenous quorum-based wakeup scheduling for duty-cycled wireless sensor networks," Ph.D. dissertation, Ph.D. dissertation, Blacksburg, VA, USA, 2009.
[8] Y. Tseng, C. Hsu, and T. Hsieh, "Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks," *Elsevier Comput. Netw.*, vol. 43, no. 3, pp. 317–337, 2003.
[9] P. Dutta and D. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *Proc. ACM Conf. Embedded Netw. Sens. Syst.*, 2008, pp. 71–84.
[10] A. Kandhalu, K. Lakshmanan, and R. Rajkumar, "U-connect: A low-latency energy-efficient asynchronous neighbor discovery protocol," in *Proc. ACM Int. Conf. Inf. Process. Sens. Netw.*, 2010, pp. 350–361.
[11] T. Cormen, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.
[12] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. ACM Conf. Embedded Netw. Sens. Syst.*, 2004, pp. 95–107.
[13] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *Proc. ACM Conf. Embedded Netw. Sens. Syst.*, 2005, pp. 51–63.
[14] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-mac: A short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proc. ACM Conf. Embedded Netw. Sens. Syst.*, 2006, pp. 307–320.
[15] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2002, pp. 1567–1576.
[16] R. Zheng, J. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. and Comput.*, 2003, pp. 35–45.
[17] I. Anderson, *Combinatorial Designs and Tournaments*. Oxford, U.K.: Oxford Univ. Press, 1997.
[18] Y. Liu, Y. Zhu, L. M. Ni, and G. Xue, "A reliability-oriented transmission service in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 2100–2107, Dec. 2011.
[19] G. Hardy and E. Wright, *An Introduction to the Theory of Numbers*. Oxford, U.K.: Oxford Univ. Press, 1979.
[20] M. Bakht and R. Kravets, "Searchlight: Asynchronous neighbor discovery using systematic probing," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 14, no. 4, pp. 31–33, 2010.
[21] T. Crossbow. Telosb data sheet. (2013) [Online]. Available: http://www.xbow.com.
[22] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler, "The NESC language: A holistic approach to networked embedded systems," in *Proc. ACM SIGPLAN Conf. Program. Language Design Implementation*, 2003, pp. 1–11.

[23] M. N. Rahman and M. Matin, "Efficient algorithm for prolonging network lifetime of wireless sensor networks," *Tsinghua Science & Technology*, vol. 16, no. 6, pp. 561–568, 2011.
[24] K. Wang, X. Mao, and Y. Liu, "Blinddate: A neighbor discovery protocol," in *Proc. IEEE Int. Conf. Parallel Process.*, 2013.



Keyu Wang received the bachelor's degree from Computer Science Department at Zhejiang University, China, in 2008. He is currently working toward the PhD degree in the Department of Computer Science & Engineering at Hong Kong University of Science and Technology. His research interests span wireless ad hoc networks and mobile computing. He is a student member of the IEEE.



Xufei Mao (M'10) received the bachelor's degree in computer science from the Shenyang University of Technology and the MS degree in computer science from Northeastern University, in 1999 and 2003, respectively. He received the PhD degree in computer science from Illinois Institute of Technology, Chicago in 2010. He is currently an assistant research professor in the School of Software, Tsinghua University. His research interests span wireless ad hoc networks and pervasive computing. He is a member of the IEEE.

Yunhao Liu (SM'06) received the BS degree in automation from Tsinghua University, Beijing, China, in 1995, and the MS and PhD degrees in computer science and engineering from Michigan State University, East Lansing, in 2003 and 2004, respectively. He is a professor with the School of Software, Tsinghua National Lab for Information Science and Technology, and the director of the MOE Key Lab for Information Security, Tsinghua University. He is a senior member of the IEEE.