Acc: Generic On-Demand Accelerations for Neighbor Discovery in Mobile Applications

Desheng Zhang, Tian He[*]
{zhang,tianhe}@cs.umn.edu

Yunhuai Liu[†]
yunhuai@trimps.ac.cn

Yu Gu[‡]
jasongu@sutd.edu.sg

Fan Ye[∓]
fanye@us.ibm.com

Raghu k. Ganti[∓]
rganti@us.ibm.com

Hui Lei[∓]
hlei@us.ibm.com

[*]Computer Science and Engineering, University of Minnesota, USA
[†]Third Research Institute of Ministry of Public Security, China
[‡]Singapore University of Technology and Design, Singapore
[∓]IBM T.J. Watson Research Center, USA

Abstract

As a supporting primitive of many mobile device applications, neighbor discovery identifies nearby devices so that they can exchange information and collaborate in a peer-to-peer manner. To date, discovery schemes trade a long latency for energy efficiency and require a *collaborative* duty cycle pattern, and thus they are not suitable for interactive mobile applications where a user is unable to configure others' devices. In this paper, we propose *Acc*, which serves as an on-demand generic discovery accelerating middleware for many existing neighbor discovery schemes. *Acc* leverages the discovery capabilities of neighbor devices, supporting both *direct* and *indirect* neighbor discoveries. Our evaluations show that *Acc*-assisted discovery schemes reduce latency by a maximum of 51.8%, compared with the schemes consuming the same amount of energy. We further present and evaluate a *Crowd-Alert* application where *Acc* can be employed by taxi drivers to accelerate selection of a direction with fewer competing taxis and more potential passengers, based on a 10 GB dataset of more than $15,000$ taxis in a metropolitan area.

Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design, Wireless communication

General Terms

Design, Experimentation

Keywords

Protocol, Neighbor Discovery, Mobile Applications

1 Introduction

Personal mobile devices, *e.g.*, smartphones and tablets, have become popular recently, enabling numerous applications [10] [6]. Such applications rely on sensor data collected in opportunistic fashion, which they process and share within a community to monitor large-scale phenomena, *e.g.*, urban environments [3] [5],user behaviors [21] [22],transportation [2] [18],and social networks [14].

Many of these applications require a fast discovery of neighbor devices in a nearby region [11] [20] [7] [12]. For example, fast discovery is critical for firefighters to exchange information during rescue operations [11] and for players to interact with each other in location-based games [20]. This quickly collected neighbor information will allow applications to effectively collaborate among participating devices.

Several state-of-the-art discovery protocols for wireless networks [19] [23] [4] [8] [17] have been proposed to achieve a bounded discovery latency and energy efficiency, since the devices are mostly powered by batteries. We found, however, that current schemes face two challenges when directly employed on personal devices.

First, typical applications of sensor networks are delay tolerant, but in many mobile applications, humans are involved in the loop, and a longer latency, even though bounded, will distract user's attention. One could argue that adjusting duty cycles of existing solutions [19] [23] [4] [8] [17] can reduce delay in discovery when so desired. These schemes, however, require coordinated changes of duty cycle patterns, a requirement only suitable for sensor networks where a user owns the whole network and can change all devices' duty cycles collaboratively. In personal device networks, a user may be unable to configure key system parameters (*e.g.*, duty cycles) of other users' devices, meaning that accelerated discovery can be achieved only by adjusting the duty cycle of a user's own device.

Second, many mobile applications (*e.g.*, geo-social networking) running on personal devices desire a fast discovery only when such a need arises, unlike the sensor network applications where continuous discovery is need to maintain network connectivity in mobile environments, and we argue that allocating duty cycles continuously in advance of user demands is wasteful.

Therefore in this paper, we advocate accelerated discovery by individual users in an *on-demand autonomous* manner. In particular, we consider a scenario in which an effective discovery protocol, *e.g.*, Disco [4], has already been deployed in networks running with a very low duty cycle. When a faster discovery is needed by a user, an additional energy budget (in term of additional active slots) is used to perform an *on-demand* acceleration.

Our *Acc*elerator is called *Acc*, which functions based on knowledge collected by an existing discovery scheme. We aim at a generic middleware design that can support a wide range of discovery protocols with an *arbitrary* duty cycle pattern. Technically, the key novelty of *Acc* is that it leverages knowledge in the neighbor tables of *known* neighbors to maximize the utility of additional energy (*i.e.*, effectiveness of additional active slots) in order to accelerate discovery of *unknown* neighbors, while also introducing no changes on any device except the discovering one.

Specifically, our contributions are as follows:

- We introduce a transparent accelerating scheme *Acc* that works with many existing discovery protocols to greatly accelerate the discovery process. To our best knowledge, this is the first work that provides an on-demand generic solution to accelerate a wide range of discovery protocols under different duty cycle patterns.
- We propose a concept of *spatial-temporal coverage* and formally define a model for quantifying the effectiveness of each active slot in the acceleration of neighbor discovery. The model is fully distributed and leverages only information in the neighbor tables of known neighbors. It does not make any assumptions regarding radio or location models.
- Based on the spatial-temporal coverage, we design an agile online scheduling algorithm to decide additional active slots under a given energy budget. Comparing our online scheduling algorithm to its theoretically optimal Oracle version, we theoretically prove that our online scheduling algorithm is *competitive* by obtaining its competitive ratio ρ, which indicates that our online scheduling algorithm has a bounded performance compared to its Oracle version.

The evaluation results show that *Acc*-assisted schemes reduce the discovery latency by a maximum of 51.8% when consuming the same energy. In addition, based on a 10 GB dataset comprising about 7 days of GPS traces of more than $15,000$ taxis in a large city, we further propose and evaluate a *Crowd-Alert* application to show how *Acc* can be employed to quickly select a direction with fewer competing taxis or more potential passengers, It demonstrates that a smart taxi driver can maximize the possibility of picking up a passenger based on an accelerated neighbor discovery.

The paper is organized as follows. Section 2 introduces related work. Section 3 describes background. Section 4 provides our design goal. Section 5 proposes the *Acc* design. Section 6 and Section 7 present our implementation and simulation. Section 8 demonstrates how *Acc* can be used in a taxi-dispatch system. Section 9 concludes the paper.

2 Related Work

Neighbor discovery in low-power wireless networks has recently been studied in the literature. In general, neighbor discovery schemes can be divided into three categories, *probabilistic*, *quorum-based*, and *deterministic*.

The probabilistic protocols, *e.g.*, Birthday protocol [13], assign different probabilities for sending, receiving, and sleeping in individual slots. Due to Birthday Paradox [15], such probabilistic schemes offer very good performance in the average discovery latency. But their major limitation is a unbounded worst-case discovery latency, which leads to a long tail on discovery probabilities over time. Moreover, Birthday research concludes that this discovery scheme aims for stationary networks, instead of mobile networks.

The quorum-based discovery protocols address the above unbounded latency issue by ensuring overlapping active durations between any pair of devices within a bounded time. In these schemes, time is divided into $m \times m$ continuous slots as a matrix, and each device selects one row and one column (called quorums) to become active. Therefore, regardless which row and column a device chooses to become active, it is guaranteed to have at least two common active slots with other devices. But a main drawback of quorum-based protocols is a global parameter of m, which forces all devices in the network to have the same duty cycle [19] [23]. Although some work has been proposed to support asymmetric duty cycle patterns, they can support only two different duty cycle patterns [9]. Again, the quorum-based discovery protocols are also primarily proposed for stationary networks where energy is the most pressing concern, not mobility.

The deterministic protocols are most closely related to our work [4] [8] [17]. They recently have been proposed to handle the global parameter problem by letting every device distributedly select one or multiple prime numbers for itself to represent its duty cycle. Based on the Chinese Remainder Theorem [16], the devices would have bounded discovery latencies. In Disco [4], each device selects two prime numbers and generates its period independently based on these numbers. To improve Disco's performance, U-Connect [8] proposes an activation pattern using one prime and has a shorter latency, especially in asynchronous symmetric networks. More recently, WiFlock [17] combines discovery and maintenance using a collaborative beaconing mechanism with time synchronization.

Our work presents a different design architecture than the aforementioned three categories. We utilize an existing discovery protocol (*e.g.*, Disco) to guarantee a bounded discovery latency by maintaining original active slots. Our design adds only new active slots in addition to the slots specified by the underlying discovery protocol. This unique design philosophy allows an on-demand acceleration without the need for additional coordination among mobile devices. Another key novelty of this work is that when we add new active slots, we quantify the effectiveness of each added active slot on both *direct* and *indirect* discovery, and the latter part has not been considered in previous discovery designs.

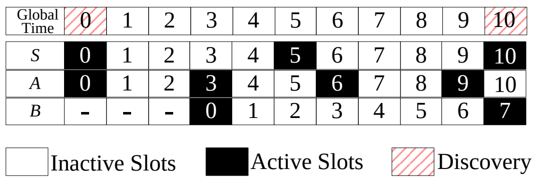
3 Preliminaries for Neighbor Discovery

In this section, we introduce some background information about in a distributed network how mobile devices can discover each other without any infrastructure support.

Many applications include devices with highly diverse configurations distributed in a wide geographic area, such as low-cost sensors in the wild. Therefore, it is very difficult to achieve global time synchronization at fine granularity. The devices usually decide their schedules based on a distributed yet coordinated duty cycle pattern.

To schedule its discovery, a device S divides time into continuous fixed-length time slots. Then, based on a specific protocol, S activates its radio and switches into a discovery mode during a specific set of slots. After that, S broadcasts one or multiple discovery messages for other devices to discover its existence. At the same time, S also listens to a wireless channel to receive similar messages from other devices. Essentially, when neighbor devices have overlapping slots in which they enter the discovery mode, they are able to discover each other [4] [8] [17].

Although our *Acc* design can work with a wide range of discovery protocols, for the sake of clarity in this paper we use Disco [4] as a representative example. In the evaluation, we will show how *Acc* works with WiFlock [17] and U-Connect [8] as well. Specifically, Disco employs the Chinese Remainder Theorem [16] to guarantee a discovery latency bound. Although in real implementation, Disco selects two different primes for a device to solve the issue of two devices having the same prime [4], for simplicity we choose only one prime to represent a duty cycle of a device to show the principle of Disco. For every chosen prime number of slots, the device will enter into its discovery mode for one slot. Consequently, the actual duty cycle is equal to the reciprocal of this chosen prime number. For example, to achieve an approximately 1% duty cycle, Disco would choose the prime number of 101. The maximal discovery latency between two devices, according to the Chinese Remainder Theorem, is equal to the product of two prime numbers chosen by these two devices. Figure 1 shows an example of asynchronous discoveries among three devices S, A, and B.



Global Time	▨ 0	1	2	3	4	5	6	7	8	9	▨ 10
S	■ 0	1	2	3	4	■ 5	6	7	8	9	■ 10
A	■ 0	1	2	■ 3	4	5	■ 6	7	8	■ 9	10
B	-	-	-	■ 0	1	2	3	4	5	6	■ 7

☐ Inactive Slots ■ Active Slots ▨ Discovery

Figure 1. Neighbor Discovery Process. Device S, A, and B start their local timers at global time 0, 0, and 3, respectively. According to Disco, S will discover device A and B at global slots 0 and 10, respectively, based on their duty cycles, *i.e.*, 20% ($\frac{1}{5}$), 33% ($\frac{1}{3}$), and 14% ($\frac{1}{7}$).

Note that existing asynchronous discovery protocols only assume that slots at individual devices have equal lengths [4] [8] [17]. By sending two messages at the beginning and end of an active slot, they do not require perfectly aligned slots and are robust to clock drift. The perfect alignment in Figure 1 is just for illustrations.

4 *Acc* Design Goals

Our work is motivated by the observation that current state-of-the-art neighbor discovery schemes suffer from long discovery latencies for energy efficiency. In many mobile applications, however, neighbor discovery has to be fast enough to enable crucial responsive user experiences. For traditional discovery schemes, its design goal is to discover neighbors with a more energy-efficient method, no matter how long it will take, as long as it is bounded. In contrast, for many mobile applications where energy is not the most pressing concern, when needed, an on-demand fast neighbor discovery has to be done in a very short period of time before users begin to lose their focus on the application.

These observations consequently lead to a new design philosophy for neighbor discovery: to perform an on-demand fast discovery within a given additional energy budget, a device should discover its neighbors as quickly as possible to make applications function smoothly. Therefore, our design goal is to more efficiently utilize the additional energy budget to accelerate the discovery process, compared to current designs with the same amount of energy.

To illustrate the design goal, we plot testbed results on the CDF of latency for both Disco [4] and its *Acc*-assisted version, *Acc*-Disco, in Figure 2. One would assume that discovery latency can be surely reduced if additional energy is used. To make the comparison fair, we run Disco at a 10% duty cycle and *Acc*-Disco with a 5% duty cycle allocated to Disco for bounded latency and another 5% duty cycle allocated to *Acc* for acceleration purpose. Therefore, Disco and *Acc*-Disco have the same total energy budget. The system details are given in Section 6.

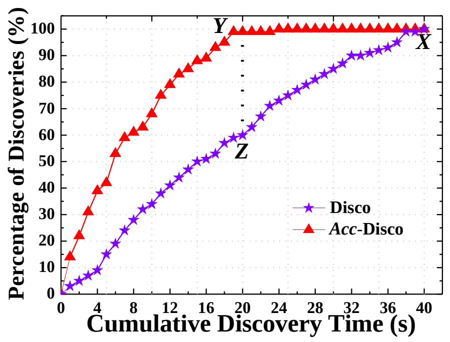


Figure 2. Illustration of Design Goal

In Figure 2, as shown by point X, Disco discovers more than 99% of neighbors after a latency 40s on average. In contrast, as shown by point Y, *Acc*-Disco completes this process within 20s. Under the same latency, Disco discovers about 60% of neighbors, as shown by point Z. Therefore, comparing point Y to X, to discover all neighbors, our *Acc* can assist Disco to accelerate its discovery process, by a maximum of 50%; comparing point Y to Z, under the same latency, our *Acc* can assist Disco to achieve more discoveries, by a maximum of 40%. Based on the above observations, our key goal is enabling *Acc* to optimally utilize the additional energy budget to reduce the discovery latency for the same number of neighbors rather than simply assigning this additional budget to the existing discovery protocols.

5 *Acc* Design

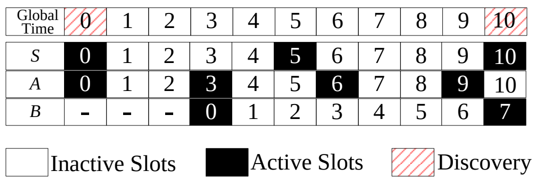
In this section, we introduce our detailed design for accelerations of neighbor discovery in mobile applications.

5.1 Main Idea

We assume an effective discovery protocol, *e.g.*, Disco, has already been installed in each device. Augmented further by *Acc*, a device runs in one of two discovery modes: *Energy Efficient Discovery Mode*, and *On-demand Accelerated Discovery Mode*. If a fast discovery is not required, a device S is in the first mode, *i.e.*, only running the underlying discovery protocol; otherwise, S enters the second mode, concurrently performing *Acc* and the underlying discovery protocol for both the acceleration and the bounded latency.

Energy Efficient Discovery Mode: In this mode, S performs the following two steps during its original active slots (as specified by the underlying discovery protocol), and turns off its radio in the rest of slots. (1) At the beginning and end of the original active slots, S sends a discovery message including its neighbor table, *i.e.*, its own duty cycle as well as IDs and duty cycles of its current known neighbors. (2) S may receive similar discovery messages from previously unknown or known neighbors if they also become active in the same slots with S. Therefore, S will collect some activation schedules about some known neighbors, *i.e.*, when the known neighbors will become active again in future slots. This information is very valuable, because when an on-demand accelerating discovery is required, it will help S to decide how to accelerate the discovery.

On-demand Accelerating Discovery Mode: When an on-demand fast discovery is required, S enters this mode to accelerate the discovery process with an additionally provided energy budget. In this mode, besides original active slots, S will also become active during several additional slots to receive discovery messages. These additional slots are optimal for discovering more potential neighbors in two ways: *direct neighbor discovery* by S itself, and *indirect neighbor discovery* by S's known neighboring devices. This indirect discovery is performed by receiving neighbor tables from other devices in active slots. Figure 3 gives an example of the indirect discovery.



Global Time	▨ 0	1	2	3	4	5	6	7	8	9	▨ 10
S	■ 0	1	2	3	4	■ 5	6	7	8	9	■ 10
A	■ 0	1	2	■ 3	4	5	■ 6	7	8	■ 9	10
B	-	-	-	■ 0	1	2	3	4	5	6	■ 7

☐ Inactive Slots ■ Active Slots ▨ Discovery

Figure 3. Indirect Discovery. After the discovery of a device A in the global time 0, if S can select one additional active slot between the global time 1 to 10, S would select slot 6 for possible *indirect* discoveries via A, since (1) S knows A will become active in slot 6 after the initial discovery, and (2) neighbors discovered by A in slot 3, *e.g.*, B, will be forwarded to S in Slot 6. So S accelerates the discovery process of B by 4 slots, *i.e.*, from slot 10 to 6.

A natural and key question comes up: how to select additional active slots that are most effective when the energy budget is given. Before answering this question, we first explain the operational difference between existing discovery protocols and *Acc*. In existing discovery schemes, a discovering device S discovers its neighbors only by S itself, without any direct collaborations with neighbors already known. Therefore, when characterizing a potential active slot in terms of discovery, existing schemes may consider only how many unknown neighbors whom S can *directly* discover by itself if S becomes active in this slot. These direct discoveries can accelerate the discovery process on a certain level, but not significantly. In contrast, our *Acc* characterizes a potential active slot based on how many unknown neighbors whom S's known neighbors will discover can be forwarded to S to achieve *indirect* discoveries. *This indirect discovery is one of the key features of Acc.* Compared to direct discoveries, these indirect discoveries significantly accelerate the discovery process. This is because direct discoveries increase only linearly, but indirect discoveries may increase geometrically.

We break down the question of how to select additional slots into two sub-questions: (1) how to evaluate the effectiveness of all potential active slots, and (2) among these potential active slots, how to select a subset of active slots to maximize the discovery probability and reduce discovery latency. A potential active slot t is evaluated by a metric of *spatial-temporal coverage*, which is considered as a slot gain to quantify discovery capabilities of all known neighbors becoming active at slot t. These known neighbors can discover common unknown neighbors for S during the slots that S is not active and then forward such information to S at slot t. Since the known neighbors of S will discover their neighbors anyway, *Acc* supports a transparent acceleration for S running at the on-demand accelerating discovery mode. This is because no additional effort (*e.g.*, additional activations) is needed for S's neighbors running at the energy efficient-discovery mode. We present the slot gain in the second subsection. Then we explain how to dynamically schedule a subset of active slots that maximize the total slot gains, given a fixed energy budget (*i.e.*, the number of active slots to be added). We present this online scheduling algorithm in the third subsection.

5.2 Characterization of Slot Gain

Before presenting the detailed characterization of slot gain, we first provide some intuition behind this concept. To discover more unknown neighbors, a discovering device S should become active at a future slot that has the largest number of potential *unknown* neighbors that are also becoming active. Therefore, intuitively, a future slot with more active unknown neighbors should be assigned to a larger gain.

But without making further assumptions, S cannot have this information about how many unknown neighbors will become active in a certain future slot. Alternatively, S indeed has information collected during previous discoveries about how many and which kinds of S's *known* neighbors will become active in a certain future slot. These known neighbors will passively forward their new collected neighbor information to S to achieve *indirect* discoveries by sending neighbor tables, if the known neighbors become active together with S in a future slot. Again intuitively, a future slot with more active known neighbors should be assigned to a larger gain.

But we observe that not all known neighbors at S are equally valuable for indirect neighbor discoveries. Specifically, S should favor those important known neighbors exhibiting both *temporal diversity* and *spatial similarity* to S. The temporal diversity indicates how many slots a known neighbor is active even though S is not, while the spatial similarity indicates how likely a neighbor of a known neighbor of S is also S's neighbor. Finally, a future slot with more active known neighbors exhibiting both higher temporal diversity and larger spatial similarity will be assigned to a larger gain.

5.2.1 Temporal Diversity

The temporal diversity between a pair of devices S and its known neighbor A is determined by the difference in active slot schedules between them. The more difference in active slots, the more likely that via A, S can *early indirectly* discover new neighbors whom S was supposed to *later directly* discover during S's original active slots. For example, Figure 4 shows an example of temporal diversity.

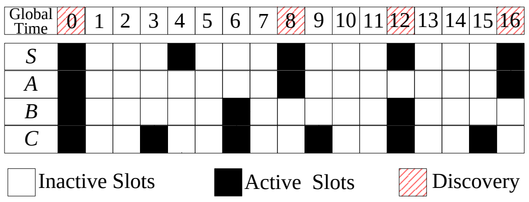


Figure 4. Example of Temporal Diversity

In Figure 4, whenever A becomes active, S also becomes active, so the temporal diversity between them is very limited. Because A can only discover neighbors in slots where S also does, there is limited information that A can learn but S cannot. However, a device C frequently becomes active in slots where S is inactive (*e.g.*, slot 3, 6, 9 and 15). Given a slot t, the more frequently C becomes active before t, the larger the possibility that C obtains more information on the potential neighbors not yet known to S. Therefore, to maximize the possibility that the known neighbors can forward more information about the unknown potential neighbors to S, *Acc* attempts to activate S at slots where more known neighbors with higher temporal diversities become active.

At current slot t_0, to calculate the temporal diversity between two device i and j at a future slot t, denoted as $\alpha_{t_0\to t}^{(i,j)}$, j utilizes the ratio between the number of non-overlapping active slots between i and j from the current slot t_0 to slot t, and the total number of slots until slot t. This ratio is given by the following formula.

$$\alpha_{t_0\to t}^{(i,j)} = \frac{|m_{t_0\to t}^{(i,i)}| - |m_{t_0\to t}^{(i,j)}|}{t - t_0}, \tag{1}$$

where $m_{t_0\to t}^{(i,j)}$ is the common active slot set of i and j from slot t_0 to slot t; clearly, if $j = i$, then $m_{t_0\to t}^{(i,i)}$ is the total active slot set of i from slot t_0 to slot t.

As in Figure 4, we show how to obtain $\alpha_{t_0\to t}^{(i,j)}$. Assuming device S, A, B and C first discover each other at slot 0. At t_0 = slot 1, the temporal diversity of slot 6 for A, B and C to S is $\alpha_{1\to 6}^{(A,S)} = \frac{0}{5}$, $\alpha_{1\to 6}^{(B,S)} = \frac{1}{5}$, and $\alpha_{1\to 6}^{(C,S)} = \frac{2}{5}$, respectively. Clearly, A has the least temporal diversity to S, while C has the most temporal diversity to S.

5.2.2 Spatial Similarity

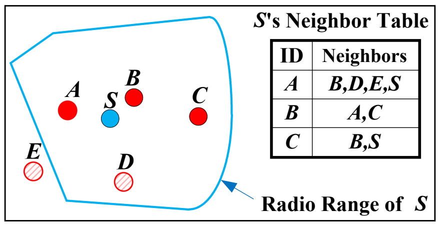
The spatial similarity between a pair of devices S and A is determined by the spatial closeness between them. In multi-hop networks, not all A's neighbors are S's neighbors. Intuitively, the closer A is to S, the larger the possibility that more common neighbors exist between them. So, to maximize the possibility that the potential unknown neighbors forwarded by the known neighbors to S are indeed S's neighbors, *Acc* attempts to activate S at slots where more known neighbors with larger spatial similarities become active.

At current slot t_0, to calculate the spatial similarity between device i and j, denoted as $\beta_{t_0}^{(i,j)}$, j utilizes the ratio between the number of common known neighbors of i and itself, and the total number of known neighbors to itself at slot t_0. This ratio is given by the following.

$$\beta_{t_0}^{(i,j)} = \frac{|n_{t_0}^{(i,j)}|}{|n_{t_0}^{(j,j)}|}, \tag{2}$$

where $n_{t_0}^{(i,j)}$ is the common known neighbor set of i and j at slot t_0; clearly if $i = j$, $n_{t_0}^{(j,j)}$ is j's neighbor table at slot t_0.

Figure 5 shows an example about how to obtain $\beta_{t_0}^{(i,j)}$.



S's Neighbor Table

ID	Neighbors
A	B,D,E,S
B	A,C
C	B,S

Figure 5. Example of Spatial Similarity

In Figure 5, at t_0 = slot 1, among 3 discovered neighbors (*i.e.*, A, B, and C), S shares 2, 3 and 2 neighbors with A, B and C, respectively, including a neighbor itself. So, for directly discovered neighbors A, B, and C, S can calculate $\beta_1^{(A,S)} = \frac{2}{3}$, $\beta_1^{(B,S)} = \frac{3}{3}$ and $\beta_1^{(C,S)} = \frac{2}{3}$. For indirectly discovered neighbors, *e.g.*, D, S can calculate $\beta_1^{(D,S)} = \frac{1}{3}$, since only 1 (device A) out of 3 known neighbors of S has D in its neighbor table.

5.2.3 Slot Gain Calculation

Based on the above observations, a discovering device S assigns larger gains to slots that have more active devices with higher *temporal diversity* and larger *spatial similarity*. At current slot t_0, based on Eq. 1 and 2, S calculates the *slot gain* of slot t, denoted as $\gamma_{t_0\to t}^{(S)}$, as follows.

$$\gamma_{t_0\to t}^{(S)} = \sum_{i\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(i,S)} \beta_{t_0}^{(i,S)} = \sum_{i\in n_{t_0}^{(S,S)}} \frac{(|m_{t_0\to t}^{(i,i)}| - |m_{t_0\to t}^{(i,S)}|) \times |n_{t_0}^{(i,S)}|}{(t - t_0) \times |n_{t_0}^{(S,S)}|}, \tag{3}$$

where $n_{t_0}^{(S,S)}$ is the neighbor table of S at slot t_0.

Ideally, if S is required to discover all its neighbors becoming active from slot t_0 to t but without being active all the time, then S should select a set of known neighbors who can cover the entire radio area (*i.e.*, spatial coverage) of S from slot t_0 to t (*i.e.*, temporal coverage). The temporal coverage is easy since we select a neighbor subset, if any, that has neighbors continuously becoming active from slot t_0 to t, but without further assumptions regarding a device's radio model, the spatial coverage is hard to perform. Essentially, S could use its complete neighbor set to represent its radio area, but S does not know its complete neighbor set either, but only a partial known neighbor set at a specific slot. Therefore, we employ S's partial known neighbor set (*i.e.*, $n_{t_0}^{(S,S)}$) to represent its radio area, *i.e.*, the spatial coverage for S is the coverage of S's known neighbor set. This strategy performs best in a situation where the partial known neighbor set is uniformly distributed in the complete neighbor set.

Consequently, the denominator $(t-t_0)\times|n_{t_0}^{(S,S)}|$ in the last term of Eq. 3 is the temporal-spatial coverage should be provided for S to discover all its neighbors becoming active from slot t_0 to t; whereas the numerator $(|m_{t_0\to t}^{(i,i)}|-|m_{t_0\to t}^{(i,S)}|)\times|n_{t_0}^{(i,S)}|$ is the temporal-spatial coverage that a known neighbor i can provide for S. Therefore, the fraction represents among the total temporal-spatial coverage of S, how much coverage can be provided by i who becomes active in slot t. This is the physical meaning of slot gains.

For example, with the schedule in Figure 4 and the neighbor table in Figure 5, assuming that t_0 is slot 1, S can calculate slot 6's slot gain as follows.

$$\gamma_{1\to6}^{(S)}=\alpha_{1\to6}^{(A,S)}\beta_1^{(A,S)}+\alpha_{1\to6}^{(B,S)}\beta_1^{(B,S)}+\alpha_{1\to6}^{(C,S)}\beta_1^{(C,S)}=\frac{0}{5}\frac{2}{3}+\frac{1}{5}\frac{3}{3}+\frac{2}{5}\frac{2}{3}=\frac{7}{15} \tag{4}$$

5.3 Online Activation Scheduling

In this subsection, we present our online scheduling algorithm, given an fixed duty cycle budget $\mathbb{B}$ and all slot gains. It outputs a slot sequence for additional activations and updates this sequence consistently based on the latest yet incomplete neighbor table. Then we analyze the online scheduling algorithm by comparing it to its optimal Oracle version.

5.3.1 Scheduling Algorithm

A device S can decide an additional active slot sequence $\mathbb{AS}$ which includes several additional active slots, according to three inputs as follows.

(1) Additional Energy Budget $\mathbb{B}$: Given $\mathbb{B}$ in terms of additional duty cycle, *e.g.*, $\frac{2}{11}$ beyond what has already been consumed by an underlying discovery scheme, S can perform discovery in some additional slots. $\mathbb{B}=\frac{2}{11}$ indicates that on average every 11 slots, S can additionally become active in 2 slots besides the original active slots.

(2) Neighbor Table $n_{t_0}^{(S,S)}$ in Current Slot t_0: After every active slot, $n_{t_0}^{(S,S)}$ will be updated based on latest neighbor information collected during this active slot. With this updated $n_{t_0}^{(S,S)}$, S continues to decide upon following additional active slots based on the updated slot gains we defined in Eq. 3.

(3) Next Original Active Slot t_N: Taking t_N into consideration is because S should not select additional active slots after t_N. This is because all slot gains may be changed after t_N, since S's neighbor table may be changed after an active slot. Therefore, selecting additional active slots after t_N will lead to a sub-optimal selection.

The above three inputs provide necessary information for S to decide $\mathbb{AS}$ with Algorithm 1 after every active slot.

Algorithm 1 *Acc* Activation Scheduling

```
Require: (1) $\mathbb{B}$; (2) $n_{t_0}^{(S,S)}$; (3) $t_N$;
Ensure: Additional active slot sequence $\mathbb{AS}$;
1: Calculating the number, denoted as $K$, of additional active slots that $S$ can have before $t_N$, based on $\mathbb{B}$;
2: Updating the slot gains for all remaining slots before $t_N$, based on $n_{t_0}^{(S,S)}$ and Eq. 3;
3: Selecting Top-$K$ slots from all remaining slots before $t_N$ to update $\mathbb{AS}$ as the additional active slots combined with original active slots;
```

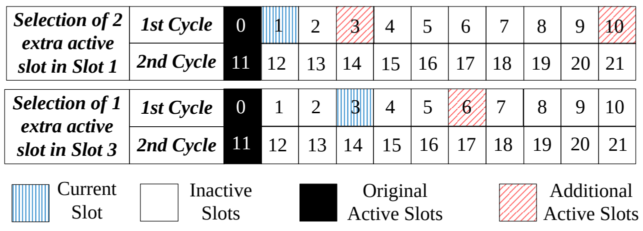


Figure 6. Example of Activation Scheduling

Figure 6 gives an example of this algorithm. Suppose that $t_0=1$, original duty cycle is $\frac{1}{11}$ and $\mathbb{B}$ is $\frac{2}{11}$, which means in every 11 slots S can activate approximately 2 additional active slots. Suppose that slots 3 and 10 have the top-2 largest gains among all slots before $t_N=$ slot 11. Therefore, in the first round, S selects slot 3 and 10, and puts them into $\mathbb{AS}$. After the activation in slot 3, S updates the slot gains of remaining slots via $n_3^{(S,S)}$. Suppose that now slot 6 has the largest slot gain, instead of slot 10, so in the second round, S would select slot 6 as the last additional slot to update $\mathbb{AS}$.

5.3.2 Competitive Analysis of Scheduling Algorithm

We analyze the performance of our online scheduling algorithm by comparing it to its optimal Oracle version. In our online scheduling, S's incomplete neighbor table in slot t_0, $n_{t_0}^{(S,S)}$, is processed piece-by-piece in a serial fashion to decide $\mathbb{AS}$, because it is consistently updated, whereas the Oracle version will have the complete neighbor table $N^{(S,S)}$, not $n_{t_0}^{(S,S)}$, to decide $\mathbb{AS}$. In the appendix, we prove that our online scheduling is competitive by showing that the performance ratio between it and its Oracle version, denoted as ρ, is bounded by a parameter R, which is the size ratio between $n_{t_0}^{(S,S)}$ and $N^{(S,S)}$. The rationale behind this analysis is that our online scheduling performance is proportional to the size of $n_{t_0}^{(S,S)}$. For example, if $R=1$, then our online algorithm is as effective as its Oracle version, since $R=1$ indicates that $n_{t_0}^{(S,S)}=N^{(S,S)}$.

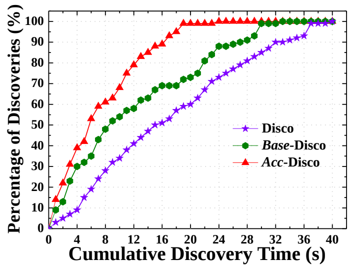


Figure 7. Disco CDF

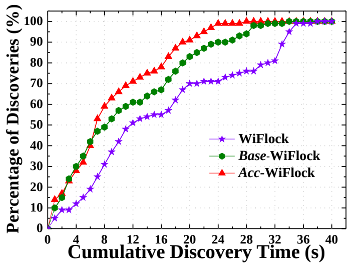


Figure 8. WiFlock CDF

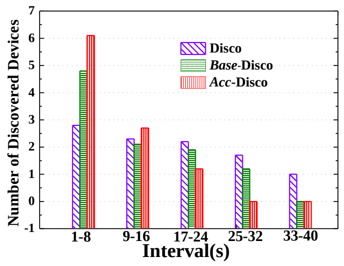


Figure 9. Disco Distribution

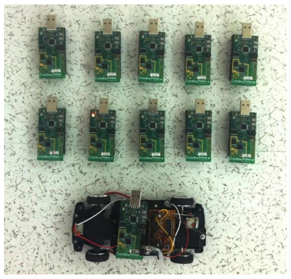


Figure 10. Testbed Setup

Energy / Name	Additional Duty Cycle	Original Duty Cycle
Disco	Used by **Disco**	Used by **Disco**
***Base* -Disco**	Used by **Baseline**	Used by **Disco**
***Acc* -Disco**	Used by ***Acc***	Used by **Disco**

Figure 11. Compared Schemes

6 Testbed Evaluation

To evaluate *Acc* in a real world setting, we integrate *Acc* with two state-of-the-art discovery protocols: Disco [4] and WiFlock [17]. We implement the above two schemes employing 11 TelosB sensor devices with a 10 KB RAM size on the TinyOS/Mote platform . During the testbed experiments, we deploy 10 TelosB sensor devices in a one-hop grid network and utilize a mobile toy car attached with another TelosB as a discovering device, with a mobility pattern of circling around the grid. The testbed is shown in Figure 10.

At individual devices, we set the time slot length to be $25ms$ for two reasons. (1) For direct discovery, a smaller slot will lead to a faster discovery, but a too-small slot ($< 5ms$) will lead to the jitters introduced by the TinyOS timer library [4]. (2) For indirect discovery, a bigger slot will reduce collisions of messages and enable more exchanges of neighbor tables. Based on the above two reasons, we make a tradeoff about time slot length on $25ms$. Note that WiFlock was implemented on modified hardware to support an extremely small time slot $80\mu s$ [17], but in our paper we implement WiFlock only on a standard hardware to examine the principle of its collaborative beaconing mechanism. The additional duty cycle budget $\mathbb{B}$ for the acceleration is set to be 5%, the same as the original duty cycle of 5% at every device. The 5% duty cycle is extensively studied in Disco [4].

To evaluate the effectiveness of the slot gains we proposed, we also implemented a ***Baseline*** design. This design shares the same scheduling scheme as *Acc*, but it uses the number of active devices in a slot t as the slot gain, not considering any temporal diversity or spatial similarity. So, we implement three versions as shown in Figure 11. In all three versions, the original duty cycle is controlled by Disco, and the additional duty cycle is controlled by their own schemes. Similar versions are implemented for WiFlock.

We evaluate the above schemes by three metrics: (1) the percentage of discoveries with respect to cumulative discovery time, (2) the number of discovered devices in different time intervals, and (3) average discovery latency in different duty cycles. The first two metrics are to verify the effect of *Acc*'s assistance to existing schemes in the acceleration of discovery process. The third metric is to verify the effect of different duty cycles on the average discovery latency.

In an experiment, after every 40 slots, *i.e.*, about $1s$, the discovering device logs the number of neighbors it discovered so far. All experiments are repeated 20 times and the average results are reported.

6.1 Effectiveness in Acceleration of Discovery

Figure 7 plots the acceleration effect of Disco. In Figure 7, we observe that the curve of *Acc*-Disco is the above all other curves in every percentage of discoveries. For example, to discover 80% of neighbor devices, *Acc*-Disco, *Base*-Disco, and Disco spend around $13s$, $22s$, and $27s$, respectively. *Acc*-Disco finishes the discovery process faster than Disco by 51.8%, while both consume the same energy. This is because Disco does not consider using known neighbors to discover unknown neighbors, which leads to a longer discovery process in which a device has to find its neighbors one by one. In addition, we observe that *Base*-Disco outperforms the original scheme by a maximum of 18.9% when discovering more than 99% of neighbors on average. This is because *Base*-Disco selects active slots with more known neighbors becoming active, which proves the value of taking the known neighbors into consideration. But we also observe that *Acc*-Disco still outperforms *Base*-Disco by nearly 36.6% when discovering more than 99% of neighbors. This suggests that when selecting additional active slots, considering only the quantity, not the quality, of devices becoming active in slots is not enough to significantly accelerate discovery. This can also be shown by the fact that *Base*-Disco discovers half of devices' neighbors by $8s$, but finishes the whole discovery process at $32s$. The above results indicate that *Acc*-Disco exhibits a significant acceleration, when compared to other versions.

In Figure 8, we observe the similar results as in Figure 7. Among the three versions, *Acc*-WiFlock achieves the highest performance in the percentage of discovered devices in the most instances of the discovery process. But we also observe that the performance gain between *Acc*-WiFlock and other versions of WiFlock is less than that between *Acc*-Disco and

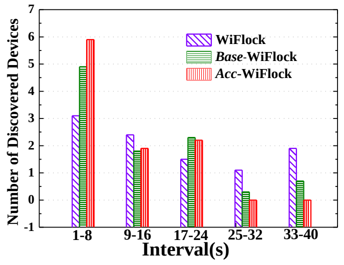


Figure 12. WiFlock Distribution

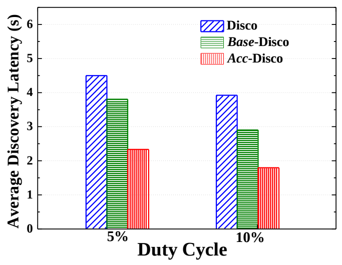


Figure 13. Disco Latency

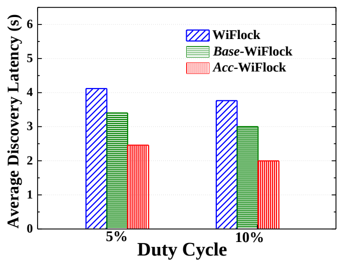


Figure 14. WiFlock Latency

other versions of Disco. This is because in the collaborative beaconing mechanism of WiFlock, WiFlock has already taken neighbor tables into consideration. Different than *Acc*-WiFlock and *Base*-WiFlock, however, the neighbor tables in WiFlock are intended to maintain the membership of a device group to achieve synchronized listening. In Figure 8, we observe that *Base*-WiFlock outperforms WiFlock as well. This demonstrates the effectiveness of considering known neighbors for unknown neighbor discovery. But the fact that *Acc*-WiFlock outperforms *Base*-WiFlock indicates that considering temporal-spatial coverage, instead of only the number of neighbors, will achieve further improvement. This is because by simply measuring the slot gain as the number of active neighbors, *Base*-WiFlock can increase performance in a certain level but cannot make a device become active at the most effective slots, as *Acc*-WiFlock does.

Figures 9 and 12 plot the number of neighbors discovered in every 8s time window under the versions of Disco and WiFlock. These two figures provide the distribution of discovered neighbor numbers in different phases of the discovery process. From Figures 9 and 12, we observe that both *Acc*-Disco and *Acc*-WiFlock discover the largest number of neighbor devices during the first 8s. In contrast, the other versions discover relatively uniform numbers of devices over time. The reason for Disco's uniform discoveries is obvious, since Disco performs a pair-wise discovery where discovering more neighbors is not helpful for the discovery of the next neighbor. But WiFlock does indeed consider a group-based strategy. One explanation for WiFlock's uniform discoveries is that WiFlock's synchronized listening and one-way discovery mechanism are efficient only for an existing group of devices to discover a new device, not for a new device to discover all its neighbors.

From the above four figures, we can conclude that when an additional energy budget is given for an acceleration of the discovery process, considering the number of devices active in a slot (Baseline) can assist current discovery schemes by a certain level, but there still is room to improve. By taking different qualities of known neighbors into consideration, *i.e.*, the temporal diversity or spatial similarity of neighbors, *Acc* can further accelerate the discovery process.

6.2 Impact of Duty Cycle

Figures 13 and 14 plot the impact of two different original duty cycles on average discovery latencies in both Disco and WiFlock. The average discovery latency is defined as the time a device takes to discover all its neighbors divided by the number of its neighbors. In these two sets of experiments, we observe that the versions with *Acc* outperform the versions with Baseline and original schemes by a maximum of 42.1% and 53.8%, respectively. We also observe that the performance gain between the *Acc* assisted versions and original versions increases as the duty cycle increases. In Disco, this gain increases from 47.7% to 53.8%, while in WiFlock it increases from 39% to 47.3%. This indicates that as devices become active more frequently, a discovering device can obtain more information from its known neighbors by considering the temporal diversity or spatial similarity of neighbors. Again, the performance gain between the *Acc*-assisted version and the original version in WiFlock is smaller than that in Disco, which is also because of WiFlock's collaboration beaconing scheme. We observe different trends in the performance gain between *Acc*- and Baseline-assisted versions in different protocols. The gain between *Acc*-Disco and *Base*-Disco decreases from 42.1% to 37.9%, while that between *Acc*-WiFlock and *Base*-WiFlock increases from 29.4% to 33.3%. This indicates that the slot gains utilized by Baseline and *Acc* have different effects in different protocols. It also shows that the performance gain between *Acc*-WiFlock and *Base*-WiFlock, *i.e.*, 29.4%, is smaller than the gain in Disco related comparisons, *i.e.*, 42.1%. This result is consistent with the observation that the performance gain between the *Acc*-assisted version and the original version in WiFlock, *i.e.*, 39%, is smaller than that in Disco, *i.e.*, 47.7%.

From Figures 13 and 14, we conclude that when devices become more active, *Acc* can more effectively assist the discovering device to accelerate the discovery process by leveraging the known neighbors to discover unknown neighbors.

7 Simulation Evaluation

To evaluate the performance of *Acc* serving as an accelerating middleware to support different protocols in larger-scale networks, we simulate *Acc* with three discovery protocols, Disco [4], U-Connect [8] and WiFlock [17]. In our simulation, 100 mobile devices are uniformly deployed in a square area of $200m \times 200m$. The transmission ranges of devices are set from 20m to 110m, which lead to average device densities from 3.6 to 55.36. We use a random waypoint model as a mobility model [1], with an average velocity $1m/s$.

Note that in a mobile multi-hop network, neighboring relations are consistently changing, and it is extremely costly

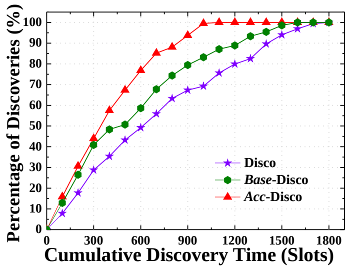


Figure 15. Disco CDF

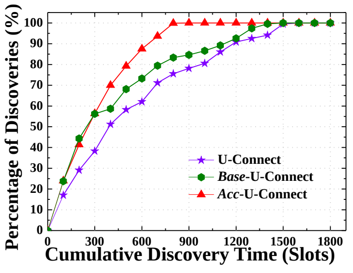


Figure 16. U-Connect CDF

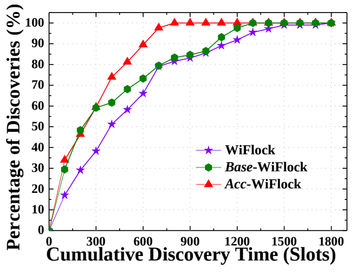


Figure 17. WiFlock CDF

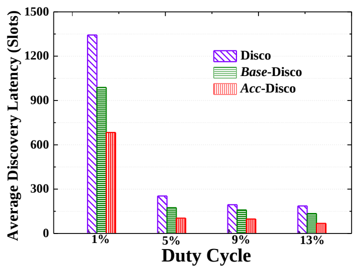


Figure 18. Disco Latency

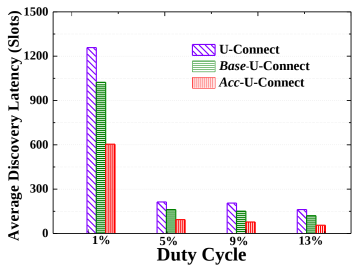


Figure 19. U-Connect Latency

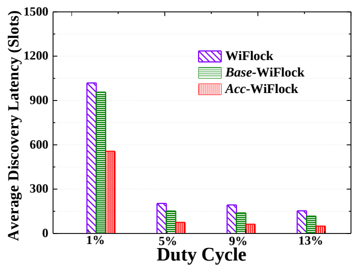


Figure 20. WiFlock Latency

in terms of energy to keep neighbor tables up to date, *i.e.*, immediately discovering a node when it is in one device's communication range. In the evaluation, we define a neighbor of a device A as a node who was continuously in the communication range of A at least a time period p, which is the discovery latency bound of a underlying neighbor discovery scheme. Two nodes just transitorily were in communication ranges of each other cannot be seen as neighbors, since they cannot be discovered by each other. That is, a neighbor will be discovered by *Acc* in advance or by a underlying scheme eventually.

7.1 Effectiveness in Acceleration of Discovery

In Figure 15, we plot the percentages of discoveries in terms of cumulative discovery time. From Figure 15, we observe that with the increase of cumulative discovery time, the percentage of discoveries also increases for all versions of Disco. Nevertheless, *Acc*-Disco is able to discover neighbors faster than other versions under the same duty cycle. For example, to discover more than 99% of neighbors, it takes *Acc*-Disco, *Base*-Disco, and Disco around 1000, 1600, and 1700 slots, respectively. If each slot is about 10ms, then *Acc*-Disco takes a device about 10s to discover more than 99% of neighbors. These results show a nearly 41.1% performance gain between *Acc*-Disco and Disco, which proves the value of taking known neighbors into consideration to discover unknown neighbors. Via a 37.5% performance gain between *Acc*-Disco and *Base*-Disco, we can verify the effectiveness of temporal diversity and spatial similarity as a slot gain.

Similarly, in Figures 16 and 17, we plot the same sets of curves for U-Connect and WiFlock. We also observe similar performance trends as in Figure 15. For example, in Figure 16, to discover more than 99% of neighbors, the cumulative discovery time for *Acc*-U-Connect, *Base*-U-Connect, and U-Connect is around 850, 1300, and 1500 slots, respectively. In Figure 17, we still observe that a performance gain between *Acc*-WiFlock and other versions of WiFlock, although the performance gain of *Acc* in WiFlock is smaller than those in Disco and U-Connect. This is also because in WiFlock's collaboration beaconing scheme, WiFlock already employs the neighbor table to log the neighbors information (*e.g.*, waking up slots, duty cycle patterns, *etc*) for further group maintenance.

From results in Figures 15, 16, and 17, we suggest that *Acc* can serve as an accelerating middleware for various schemes to accelerate the discovery process.

7.2 Impact of Duty Cycle

In this subsection, we investigate the impact of a device's original duty cycle on the average discovery latency in Figures 18, 19, and 20. We observe that with the increase of the duty cycle, the average latencies of all versions for all schemes decrease. But at each duty cycle, the versions with *Acc* in all three different schemes achieve the smallest latency. For example, when the duty cycle is set to 5%, the average discovery latencies for *Acc*-Disco, *Base*-Disco, and Disco are around 140, 200, and 380 slots, respectively. Thus, in Disco with *Acc*'s assistance, the average latency to discover one neighbor drops from 3.8s to 1.4s (at 10ms slot), a difference of 63.1%. From Figures 18, 19, and 20, we also observe that in general, as the duty cycle increases, the performance gain between versions with *Acc* and original versions increases. For example, in Figure 18, at 1% duty cycle, the performance gain between *Acc*-Disco and Disco is 50.1%, while it increases to 63.1% when the duty cycle is 5%. This is because with a higher duty cycle, the devices in

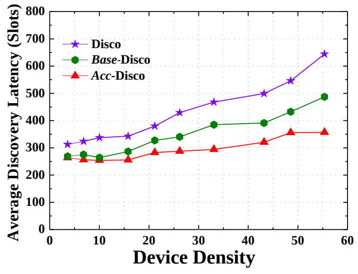


Figure 21. Disco Latency

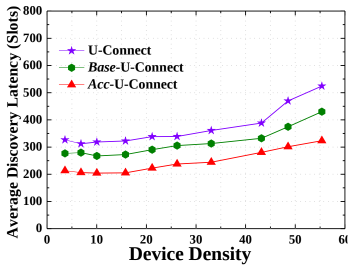


Figure 22. U-Connect Latency

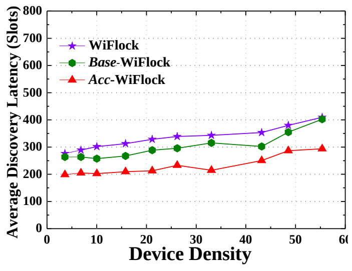


Figure 23. WiFlock Latency

the network become active more frequently, leading to more neighborhood information sharing.

7.3 Impact of Device Density

The impact of device density on the average discovery latency is shown in Figures 21, 22, and 23. From all three figures, we can see that as the device density increases, the average discovery latency increases for all schemes. This is because at higher densities, the devices have more neighbors, leading to more collisions and thus more time to find them. When the average number of neighbors increases from 3.6 to 55.36, the performance gain between original versions and the versions assisted with *Acc* also increases from 22.3% to 52.4% in Disco. This is because more known neighbor devices are able to share neighborhood information with discovering devices, thus accelerating the neighbor discovery process. Note that even though the a bigger network density can increase the collision among devices, a bigger network density also achieves a more diverse neighborhood information sharing among already known devices.

8 Crowd-Alert Application

In this section, we propose and evaluate a *Crowd-Alert* application with which taxi drivers can quickly navigate optimal directions to travel to maximize the possibility of picking up passengers, after they drop off passengers, *i.e.,* a faster neighbor discovery is demanded.

A trivial solution to this problem is to install a centralized system that collects the location and occupancy of the taxis and suggests areas of lesser competition, *i.e.*, fewer other taxis, and more passengers. But such a system requires that each taxi is equipped with cellular data connectivity (the current centralized solution with time bounds can be provided only by a cellular data channel), which can be a potential hindrance in terms of cost. We are thus faced with the challenge of obtaining passenger demand without the use of a centralized solution in a timely fashion, *i.e.*, no extra hardware installation on the taxis. We envision that individual taxi owners are equipped with smart phones that can communicate with each other using a peer-to-peer communication interface, *e.g.*, *ad hoc* WiFi, or *ad hoc* WiMax, and can install an application on their mobile devices that can obtain the *crowd levels*, both in terms of number of taxis and number of passengers in a given area. Taxi drivers who install this application can form groups of common interest to optimize their profits. Individual drivers using this application can quickly navigate to areas with a low density of taxis (and presumably a high passenger density) to maximize pickups (and thus profits).

Our proposed protocol, *Acc*, provides a mechanism for distributed discovery of neighbors in an accelerated manner, which we will adapt to the application installed on the taxi driver's mobile device. We will describe our application in further detail and evaluate the efficiency of this scheme in discovering neighbors in a timely fashion.

8.1 Application Background

In our application, every taxi broadcasts its own status record (*i.e.*, date and time, availability, direction, GPS coordinates, *etc*) to its neighboring taxis. The broadcast is performed based on a concrete discovery scheme, *e.g.*, Disco. According to the information collected, when a taxi becomes available and the driver wants to quickly pick up a passenger (*i.e.*, an on-demand acceleration is required), the taxi driver can navigate to the optimal directions, as determined by the number of *nearby competing taxis* and *nearby potential passengers*. These two strategies can maximize the probability of picking up the next nearby passengers.

Generally, the fewer competing taxis, the higher the probability of picking up passengers. With distributedly collected status records about neighboring taxis, our *Crowd-Alert* can compute the location distribution of other competing taxis that also aim to pick up new passengers.

Similarly, the more potential passengers, the higher the probability of picking up. Without the active participation of passengers, however, it is unrealistic to expect to obtain such a distribution based on taxi status records alone. But we can obtain a cumulative location distribution of passengers that have just entered or exited taxis, *i.e.*, *served* passengers, by observing the change of the Availability Bit (from 0 to 1 or from 1 to 0) in two consecutive status records about the same taxi. Further, we assume that a location distribution of served passengers is an indication of that of potential passengers, but how to effectively obtain a distribution based on the indication is outside the scope of this paper. To focus on system level, we simply utilize the location distribution of served passengers as that of potential passengers.

Based on the distributions of competing taxis and served passengers, *Crowd-Alert* can maximize the possibility of picking up passengers by guiding a taxi to a direction with fewer competing taxis or more served passengers. A faster discovery achieved by our *Acc* can assist a navigating scheme to make a timely decision.

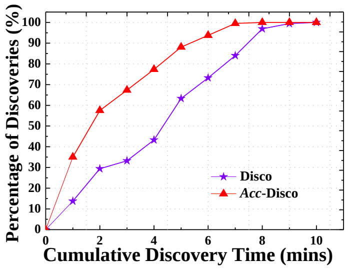


Figure 24. Disco Latency

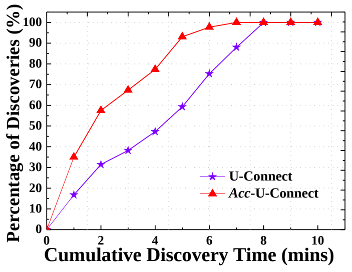


Figure 25. U-Connect Latency

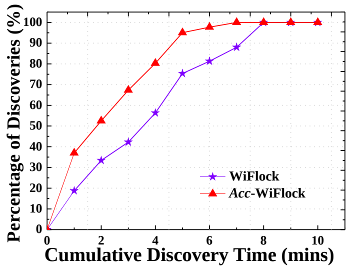


Figure 26. WiFlock Latency

8.2 Application Evaluation

We evaluate the *Crowd-Alert* application using a real world dataset of 10GB, collected from taxis in a large city.

8.2.1 Dataset

The dataset consists of 7 days of GPS traces from more than $15,000$ taxis. The data is used by the government for urban transportation pattern search. Each taxi uploads records on an average of every 30 seconds, with each record consisting of the following parameters: (1) Plate Number; (2) Date and Time; (3) GPS Coordinates; and (4) Availability: whether or not a passenger is in this taxi when the record is uploaded. Based on the above GPS trace records, we can obtain an location distribution of competing taxis or served passengers, as shown in Figures 27 and 28.

8.2.2 Reduction of Discovery Latency

Before we investigate the effects of *Acc*'s accelerations on the navigation of taxis, we first perform a trace-driven simulation on the dataset to verify how *Acc* accelerates neighbor discovery in this taxi network. With a total duty cycle $\frac{4}{30}$ (equal to the uploading speed of GPS record), we compare three schemes Disco, U-connect, and WiFlock with and without the assistance of *Acc*.

From Figure 24, we observe that *Acc*-Disco is able to discover neighboring taxis faster than Disco under the same duty cycle. For example, to discover all neighboring taxis, it takes *Acc*-Disco and Disco around 7 minutes and 9 minutes, respectively. These results show a 22% performance gain between *Acc*-Disco and Disco. We also observe that the performance gain achieves the maximum in the first half of the discovery process where a taxi can detect more than half of its neighboring taxis within 2 minutes. This suggests that *Acc* can enable *Acc*-Disco to quickly find the most neighbor taxis in a very short period of time, which can assist a driver to more quickly drive to the optimal directions. Similarly, in Figures 25 and 26, we plot the same sets of curves for U-Connect and WiFlock. In both Figures 25 and 26, we also observe similar performance trends between *Acc*-assisted versions and original versions. For example, in Figure 25, to discover all neighboring taxis, the cumulative discovery time for U-Connect and *Acc*-U-Connect is around 6 minutes and 8 minutes, respectively, achieving a 25% performance gain. In Figure 26, we still observe a performance gain between *Acc*-WiFlock and WiFlock.

From results in Figures 24, 25, and 26, we conclude that *Acc* can accelerate various discovery schemes in this taxi net-

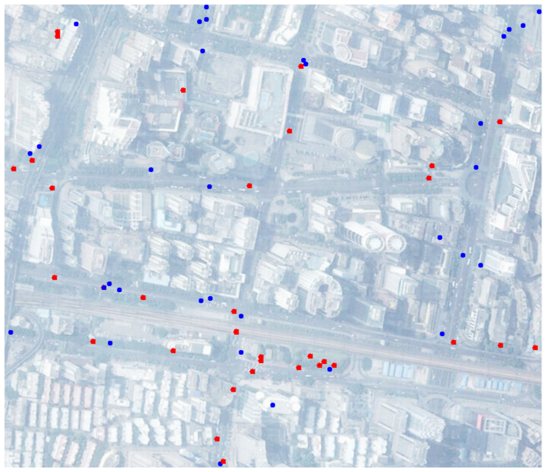


Figure 27. Distribution of Competing Taxis. Above figure shows a taxis distribution of an area about 1 square kilometer (GPS Coordinates $XXX.538$-$XXX.547 \times XXX.108$-$XXX.117$) based on a $10s$ uploading time window in a rush hour of one day, *i.e.*, 5PM. Red points indicate the taxis with passengers, while blue points indicate the taxis without passengers.

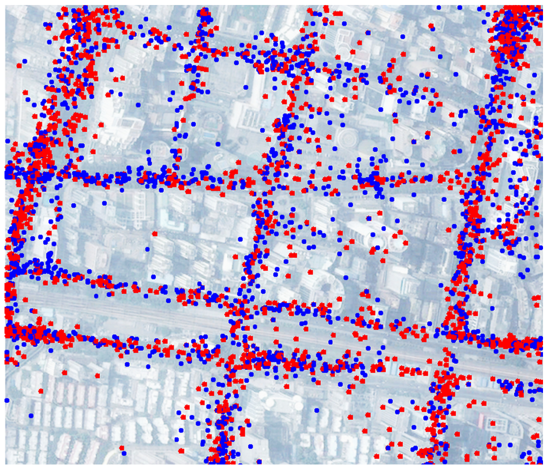


Figure 28. Distribution of Served Passengers. Above figure shows a served passenger distribution in the same area as in Figure 27 in a two hours uploading time window in one day, *i.e.*, from 4PM to 6PM. Red points indicate the locations of passengers entering taxis, and blue points indicate the locations of passengers exiting taxis.

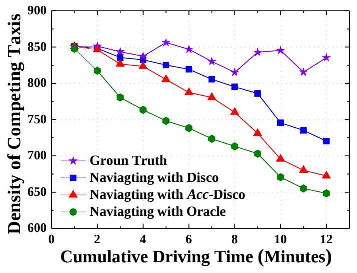


Figure 29. Density in One Smart Taxi

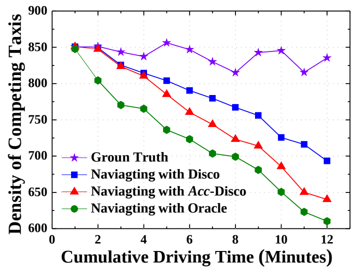


Figure 30. Density in 10% Smart Taxis

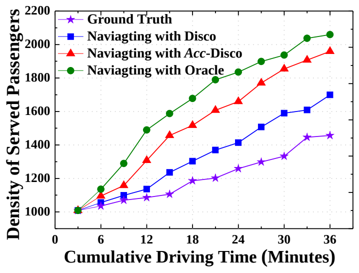


Figure 31. Density of Passengers

work, and may serve as an augmenting layer to accelerate discovery to quickly navigate taxis to optimal directions.

8.2.3 Acceleration of Navigation

In this section, we evaluate the performance of *Acc* in accelerating the navigation for taxis in *Crowd-Alert*. With a total duty cycle $\frac{4}{30}$, we compare 3 navigating results based on different discovery results of discovery schemes. **(1) Navigating with Disco**; **(2) Navigating with *Acc*-Disco**; **(3) Navigating with Oracle:** navigating taxis with discovery results of an Oracle discovery scheme where a taxi can *instantly* knows these two distributions without delay. Under all navigation, a taxi has the same preferable directions for fewer competing taxis or more served passengers. But since the employed discovery schemes are different, a navigation with a faster discovery scheme may achieve better performance. The performance is characterized by two metrics: competing taxis density and served passengers density of taxis' neighborhoods. A faster discovery may assist a navigation scheme to quickly navigating taxis to the area with fewer competing taxis or more served passengers.

To show the difference with or without our application, we also compare the above 3 schemes with **Ground Truth without Navigation**, where the density is computed based on original taxi traces without altering the routes of any taxis. Note that given the density of competing taxis or served passengers, how to select the optimal route to achieve the optimal density is outside the scope of this paper. We simply let taxis greedily select 1 out of 4 directions in an intersection according to densities in every direction and then compute densities of competing taxis or served passengers in its neighborhood every minute. We compare the performance of *Acc* under two conditions, *only one smart taxi* using navigation strategies and 10% *of total taxis* using them.

(a) Density of Competing Taxis

We investigate the densities of competing taxis in three different navigating strategies, and compare them to Ground Truth. We report the results of navigating only one taxi or 10% of total taxis to select a direction with a lower density of competing taxis, using a 3*km* communication radius, in Figure 29 and Figure 30, respectively. We assume that the mobile device is equipped with a radio (such as WiMax) that has a large communication radius. Recent phones such as the HTC Max 4G and the Sprint EVO 4G are equipped with WiMax interfaces. A smaller communication radius (*e.g.*, 100*m* in a WiFi interface) does not allow our system to fully exploit the quick discovery scheme in a taxi network. This is because an 100*m* communication radius is too small for vehicular networks where the length of a taxi is about 5*m* or 6*m*. Therefore, a navigation based on such small communication radius will lead to an extremely low density of taxis, and may not have any obvious performance difference under various discovery schemes.

Only one smart taxi: In Figure 29, we observe that as more driving time is allowed, there exists a jitter in the density of competing taxis of Ground Truth, which has no tendency toward consistent increase or decrease, while those of Disco, *Acc*-Disco and Oracle decrease. This is because the taxis with Disco, *Acc*-Disco, and Oracle navigate to a direction with fewer competing taxis, so after 3 minutes the density of competing taxis in its neighborhood drops. Compared to Ground Truth, after 12 minutes, under Disco the density decreases about 14%, while under *Acc*-Disco the density decreases about 20%, whereas Oracle outperforms Disco and *Acc*-Disco in all the cumulative driving times with a maximal performance gain of 7.5% and 12.6%, respectively. From the above results, Oracle does not significantly outperform Disco and *Acc*-Disco. One possible reason for this phenomenon is that the beginning time and location for this one smart taxi is rush hour in a downtown area. Therefore, even though Oracle provides the local optimal direction to reduce the density of competing taxis, the effect is limited.

10% smart taxis: Figure 30 plots results of 10% of total taxis using our application. We observe that compared to the results in Figure 29, all strategies have better performance, except for Ground Truth, which remains the same. The reason for the performance increase for this 10% smart taxis scenario is that the more taxis that use our applications, the more taxis that will select the direction with lower taxi density, which in turn will achieve a more uniform taxis distribution. Comparing Figure 29 to Figure 30, we note that the performance gain between navigating with *Acc*-Disco and Disco increases from 6.8% to 10.1%, indicating that *Acc*-Disco is more efficient when more taxis use our application.

(b) Density of Served Passengers

We show the effectiveness of *Acc* to assist navigating scheme to navigate taxis to a direction with more served passengers in Figure 31. Since the percentage of taxis using our application is not directly relevant to the density of already served passengers, we only show the results on the 10% of smart taxis scenario. As in Figure 27 and 28, the density of served passengers is denser than densities of competing taxis, so we use a 0.5*km* radius to compute the density.

Figure 31 plots comparisons of cumulative densities of served passengers. With an increase in cumulative driving time, the cumulative density of served passengers in a taxi's neighborhood also generally increases for all the schemes. The reason for the increases in navigation under Disco, *Acc*-Disco or Oracle is obvious, because that is the objective of our application. But the reason for the increase for Ground Truth is not so obvious. A possible explanation is that taxi drivers have rich experiences that help them select the area to maximize the probability of picking up, and that the location of already served passengers offers a strong indication to the location of potential passengers. Therefore, even without our application, experienced taxi drivers will still go to the area with more served passengers. But compared with Ground Truth, Disco can assist the taxi drivers in finding the optimal direction more quickly via discovery. Therefore, there is a performance gain between Disco and Ground Truth with a maximum of 14.3% after 36 minutes. In contrast, the navigation with *Acc*-Disco is able to discover neighbors even faster than that with Disco. For example, it outperforms those under Ground Truth and Disco with maximal gains of 25.6% and 13.2%, respectively, but has a worse performance than Oracle. Therefore, we conclude that with *Acc*-Disco, a navigation scheme can more quickly guide the taxi to a direction with a high density of served passengers.

9 Conclusion

In this paper, we introduce *Acc*, an augmenting layer for the acceleration of neighbor discovery in existing discovery schemes. The key insight into accelerating the discovery process is that known neighbors can help a device learn unknown neighbors indirectly. Thus, based on a characterization of active slots and an online scheduling algorithm that we proposed, *Acc* enables a discovering device to become active during a few optimal additional slots beyond their original schedules, leveraging both direct and indirect discovery. We have integrated our *Acc* design with three protocols, and extensively evaluated its performance in both small-scale testbed experiments and large-scale simulations. The evaluation results show that *Acc* is able to effectively accelerate the discovery process of 3 different discovery schemes by a maximum of 51.8%. To demonstrate the applications of *Acc* in the real world, we propose and evaluate a *Crowd Alert* application with a 10 GB dataset about taxi GPS traces.

10 Acknowledgements

This research was supported in part by the US National Science Foundation (NSF) grants CNS-0845994, CNS-0917097, NSFC 61170247, IBM OCR Fund, K. C. Wong Education Foundation of Hong Kong and Singapore-MIT IDC IDD61000102a.

11 References

[1] D. Alparslan and K. Sohraby. Two-dimensional modeling and analysis of generalized random mobility models for wireless adhoc networks. *IEEE/ACM Transactions on Networking*, 15(3):616–629, 2007.

[2] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson. Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In *9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*, 2011.

[3] P. Dutta, P. M. Aoki, N. Kumar, A. Mainwaring, C. Myers, W. Willett, and A. Woodruff. Common sense: participatory urban sensing using a network of handheld air quality monitors. In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, 2009.

[4] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *6th ACM Conference on Embedded Networked Sensor Systems (SenSys '08)*, 2008.

[5] P. Dutta and L. Subramanian. Human-enabled microscopic environmental mobile sensing and feedback. In *Proceedings of AAAI Artificial Intelligence for Development (AI-D '10)*, 2010.

[6] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[7] J.-H. Huang, S. Amjad, and S. Mishra. Cenwits: a sensor-based loosely coupled search and rescue system using witnesses. In *3rd ACM Conference on Embedded Networked Sensor Systems (SenSys '05)*, 2005.

[8] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *10th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN '10)*, 2010.

[9] S. Lai, B. Ravindran, and H. Cho. Heterogenous quorum-based wake-up scheduling in wireless sensor networks. *IEEE Transactions on Computers*, 59(11):1562–1575, 2010.

[10] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, 2010.

[11] H. Liu, J. Li, Z. Xie, S. Lin, K. Whitehouse, J. A. Stankovic, and D. Siu. Automatic and robust breadcrumb system deployment for indoor firefighter applications. In *8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, 2010.

[12] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet. In *2nd International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*, 2004.

[13] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, 2001.

[14] E. Miluzzo, M. Papandrea, N. D. Lane, A. M. Sarroff, S. Giordano, and A. T. Campbell. Tapping into the vibe of the city using vibn, a continuous sensing application for smartphones. In *1st international symposium on From digital footprints to social and community intelligence (SCI '11)*, 2011.

[15] M. Mitzenmacher and U. Upfal. Probabilitty and computing. 2007.

[16] H. L. I. Niven and H. S. Zuckerman. An introduction to the theory of numbers. 1991.

[17] A. Purohit, N. Priyantha, and J. Liu. Wiflock: Collaborative group discovery and maintenance in mobile sensor networks. In *10th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN '11)*, 2011.

[18] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In *8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*, 2010.

[19] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks. In *The Conference on Computer Communications (INFOCOM '02)*, 2002.

[20] Wikipedia. Location based game. *http://en.wikipedia.org/wiki/Location-based_game*.

[21] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, 2010.

[22] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner. mcrowd: a platform for mobile crowdsourcing. In *7th International Conference on Mobile Systems, Applications, and Services (MobiSys '09)*, 2009.

[23] R. Zheng, J. C. Hou, and L. Sha. Asynchronous wakeup for ad hoc networks. In *4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '03)*, 2003.

A Proof of Competitive Ratio

We analyze the performance of our scheduling by comparing it to its Oracle version having a complete neighbor table $N^{(S,S)}$. In our scheduling, a device S's neighbor table of in slot t_0, denoted as $n_{t_0}^{(S,S)}$, is processed piece-by-piece. This is a classic nature of *online* algorithm, which processes its incomplete input piece-by-piece from the start. Because of this incomplete input, an online algorithm is forced to make sub-optimal decisions. To study this sub-optimality, a competitive analysis is proposed to compare the relative performance of an online algorithm to its Oracle version that has a complete input. An online algorithm is *competitive* if its competitive ratio ρ, an performance ratio between it and its Oracle version, is bounded. To obtain ρ, we utilize qualities of selected active slots to represent algorithms' performances, indicating how much new neighbor information can be collected in these slots. The qualities of these slots can be represented by slot gains. Therefore, we can analyze ρ, by comparing the slot gains under our online scheduling and its Oracle version, employing different neighbor tables. In the following, we prove that ρ is bounded by a parameter R, which is the size ratio between $n_{t_0}^{(S,S)}$ and $N^{(S,S)}$.

Assumptions are as follows. (1) In slot t_0, a device S has already discovered a portion of its neighbors in $n_{t_0}^{(S,S)}$. (2) A parameter $R = \frac{|n_{t_0}^{(S,S)}|}{|N^{(S,S)}|} < 1$ is given, which is the ratio between the number of neighbors in $n_{t_0}^{(S,S)}$ and $N^{(S,S)}$. (3) All the discovered neighbors are uniformly distributed in $N^{(S,S)}$. (4) To minimize the effect of duty cycles, duty cycle pattern for different devices are the same.

Via Eq. 3 and assumption (1), ρ is given by

$$\frac{1}{\rho} = \frac{\gamma_{t_0\to t}^{(S)}(Oracle)}{\gamma_{t_0\to t}^{(S)}(Online)} = \frac{\sum_{i\in N^{(S,S)}} \alpha_{t_0\to t}^{(i,S)}\beta_{t_0}^{(i,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}}, \tag{5}$$

where $\alpha_{t_0\to t}^{(i,S)}$ and $\alpha_{t_0\to t}^{(j,S)}$ is temporal diversity for device $i \in N^{(S,S)}$ and $j \in n_{t_0}^{(S,S)}$, respectively; $\beta_{t_0}^{(i,S)}$ and $\bar{\beta}_{t_0}^{(j,S)}$ is spatial similarity for device $i \in N^{(S,S)}$ and $j \in n_{t_0}^{(S,S)}$, respectively. Eq. 5 can be reorganized as follows.

$$\frac{1}{\rho} = \frac{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\beta_{t_0}^{(j,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}} + \frac{\sum_{i\in \overline{n_{t_0}^{(S,S)}}} \alpha_{t_0\to t}^{(i,S)}\beta_{t_0}^{(i,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}}. \tag{6}$$

where $\overline{n_{t_0}^{(S,S)}}$ is the complement of $n_{t_0}^{(S,S)}$, given $N^{(S,S)}$.

The following is to analyze the first term in Eq. 6. According to Eq. 2 and assumption (2), we have

$$\frac{\beta_{t_0}^{(j,S)}}{\bar{\beta}_{t_0}^{(j,S)}} = \frac{|N_{t_0}^{(j,S)}|/|N_{t_0}^{(S,S)}|}{|n_{t_0}^{(j,S)}|/|n_{t_0}^{(S,S)}|} = \frac{|N_{t_0}^{(j,S)}|}{|n_{t_0}^{(j,S)}|}\frac{|n_{t_0}^{(S,S)}|}{|N_{t_0}^{(S,S)}|} = \frac{R}{R'}. \tag{7}$$

where $R' = \frac{|n_{t_0}^{(j,S)}|}{|N_{t_0}^{(j,S)}|} < 1$. Therefore, the first term in Eq. 6 can be represented as follows.

$$\frac{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\beta_{t_0}^{(j,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}} = \frac{\frac{R}{R'}\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}} = \frac{R}{R'} > 1. \tag{8}$$

where $R > R'$ is because due to assumption (3), not all $i \in \overline{n_{t_0}^{(j,S)}}$ are neighbors of j.

The following is to analyze the second term of Eq. 6. Due to assumption (4), $\forall i,\ j \in N^{(S,S)}$, $\alpha_{t_0\to t}^{(i,S)} = \alpha_{t_0\to t}^{(j,S)}$. Therefore, the second term in Eq. 6 can be reorganized as follows.

$$\frac{\sum\limits_{i\in \overline{n_{t_0}^{(S,S)}}} \alpha_{t_0\to t}^{(i,S)}\beta_{t_0}^{(i,S)}}{\sum\limits_{j\in n_{t_0}^{(S,S)}} \alpha_{t_0\to t}^{(j,S)}\bar{\beta}_{t_0}^{(j,S)}} = \frac{\alpha_{t_0\to t}^{(i,S)}\sum\limits_{i\in \overline{n_{t_0}^{(S,S)}}} \beta_{t_0}^{(i,S)}}{\alpha_{t_0\to t}^{(j,S)}\sum\limits_{j\in n_{t_0}^{(S,S)}} \bar{\beta}_{t_0}^{(j,S)}} = \frac{\sum\limits_{i\in \overline{n_{t_0}^{(S,S)}}} \beta_{t_0}^{(i,S)}}{\sum\limits_{j\in n_{t_0}^{(S,S)}} \bar{\beta}_{t_0}^{(j,S)}} \tag{9}$$

Based on the assumption (3) that $\forall j \in n_{t_0}^{(S,S)}$ and $\forall i \in \overline{n_{t_0}^{(S,S)}}$ are randomly and uniformly distributed in $N^{(S,S)}$. Therefore, $\forall i,\ j \in N^{(S,S)}$, $\beta_{t_0}^{(i,S)} = \beta_{t_0}^{(j,S)}$; $\forall i,\ j \in n_{t_0}^{(S,S)}$, $\bar{\beta}_{t_0}^{(i,S)} = \bar{\beta}_{t_0}^{(j,S)}$. So,

$$\frac{\sum_{i\in \overline{n_{t_0}^{(S,S)}}} \beta_{t_0}^{(i,S)}}{\sum_{j\in n_{t_0}^{(S,S)}} \bar{\beta}_{t_0}^{(j,S)}} = \frac{|\overline{n_{t_0}^{(S,S)}}|\beta_{t_0}^{(i,S)}}{|n_{t_0}^{(S,S)}|\bar{\beta}_{t_0}^{(j,S)}}. \tag{10}$$

Because $\overline{n_{t_0}^{(S,S)}} \cup n_{t_0}^{(S,S)} = N^{(S,S)}$ and $\frac{|n_{t_0}^{(S,S)}|}{|N^{(S,S)}|} = R$, we have $|\overline{n_{t_0}^{(S,S)}}| = \frac{1-R}{R}|n_{t_0}^{(S,S)}|$. Therefore Eq. 10 can be rewritten as follows.

$$\frac{|\overline{n_{t_0}^{(S,S)}}|\beta_{t_0}^{(i,S)}}{|n_{t_0}^{(S,S)}|\bar{\beta}_{t_0}^{(j,S)}} = \frac{\frac{1-R}{R}|n_{t_0}^{(S,S)}|\beta_{t_0}^{(i,S)}}{|n_{t_0}^{(S,S)}|\bar{\beta}_{t_0}^{(j,S)}} = \frac{1-R}{R}\frac{\beta_{t_0}^{(i,S)}}{\bar{\beta}_{t_0}^{(j,S)}}. \tag{11}$$

Since i and j are two arbitrary devices in the networks, so based on the analysis of Eq. 7, $\frac{\beta_{t_0}^{(i,S)}}{\bar{\beta}_{t_0}^{(j,S)}} > 1$. So, we have

$$\frac{1-R}{R}\frac{\beta_{t_0}^{(i,S)}}{\bar{\beta}_{t_0}^{(j,S)}} > \frac{1-R}{R} = \frac{1}{R} - 1. \tag{12}$$

Based on Eq. 8 we have the first term in Eq. 6; based on Eq. 9, Eq. 10, Eq. 11 and Eq. 12 we have the second term in Eq. 6. Therefore, Eq. 6 can be rewritten as follows.

$$\frac{1}{\rho} = \frac{R}{R'} + \frac{1-R}{R}\frac{\beta_{t_0}^{(i,S)}}{\bar{\beta}_{t_0}^{(j,S)}} > 1 + \frac{1}{R} - 1 = \frac{1}{R}. \tag{13}$$

Finally, we have the competitive ratio ρ.

$$\rho = \frac{\gamma_{t_0\to t}^{(S)}(Online)}{\gamma_{t_0\to t}^{(S)}(Oracle)} < R. \tag{14}$$

According to the above analysis, we have obtained the competitive ratio ρ of our online scheduling algorithm.□