A

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

DESHENG ZHANG, University of Minnesota
TIAN HE, University of Minnesota
YUNHUAI LIU, Third Research Institute of Ministry of Public Security, China
YU GU, IBM Research at Austin
FAN YE, Stony Brook University
RAGHU K. GANTI, IBM T.J. Watson Research Center
HUI LEI, IBM T.J. Watson Research Center

As a supporting primitive of many mobile 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 deterministic neighbor discovery schemes. *Acc* leverages the discovery capabilities of neighbor devices, supporting both *direct* and *indirect* neighbor discoveries. Further, we present a proactive online rendezvous maintenance mechanism, which is used to reduce delays for the detection of leaving of neighbors. 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. More importantly, to prove the real-world value of *Acc*, we further present and evaluate a *Crowd-Alert* application where *Acc* is employed by taxi drivers to accelerate selection of a direction with fewer competing taxis and more potential passengers, based on a 280 GB dataset of more than $14,000$ taxis in Shenzhen, the most crowded city in China.

Categories and Subject Descriptors: C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design, Wireless Communication

General Terms: Design, Experimentation

Additional Key Words and Phrases: Protocol, Neighbor Discovery, Mobile Applications

ACM Reference Format:
Desheng Zhang, Tian He, Yunhuai Liu, Yu Gu, Fan Ye, Raghu K. Ganti, and Hui Lei 2014. Acc: Generic On-Demand Accelerations for Neighbor Discovery in Mobile Applications. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 35 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

This research was supported in part by the US National Science Foundation (NSF) grants CNS-0845994, CNS-0917097, CNS-1239226, NSFC 61170247, IBM OCR Fund, K. C. Wong Education Foundation of Hong Kong and Singapore-MIT IDC IDD61000102a. A preliminary work has been presented in ACM SenSys 2012 [Zhang et al. 2012].
Author's addresses: D. Zhang and T. He, Department of Computer Science and Engineering, University of Minnesota, USA; Y. Liu, Third Research Institute of Ministry of Public Security, China; Y. Gu, IBM Research, Austin, TX, USA; F. Ye, Department of Electrical and Computer Engineering, Stony Brook University, USA; R. K. Ganti and H. Lei, IBM T.J. Watson Research Center, USA.

DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Mobile devices, *e.g*., smartphones and tablets, have become popular recently, enabling numerous applications [Ganti et al. 2011] [Lane et al. 2010]. The early applications usually are built on the premise that users check into centralized servers to coordinates with peers [Google 2013] [Foursquare 2013] [Facebook 2013], so they typically result in excessive updating process, heavy control overhead, long communication delay, and the exposure of location information on centralized services. In contrast, new applications are proposed based on direct peer to peer communications [Synerge 2013] [Pietiläinen et al. 2009] [Softonic 2012]. Usually, they rely on data collected in an opportunistic fashion, which they process and share within a community to monitor large-scale phenomena, *e.g*., urban environments [Dutta et al. 2009] [Dutta and Subramanian 2010], user behaviors [Yan et al. 2010] [Yan et al. 2009], transportation [Biagioni et al. 2011] [Thiagarajan et al. 2010], and social networks [Miluzzo et al. 2011].

Many of these applications require a fast discovery of neighbor devices in a nearby region [Huang et al. 2005] [Liu et al. 2010] [Liu et al. 2004] [Wikipedia 2013]. For example, the fast discovery is critical for firefighters to exchange information during rescue operations [Liu et al. 2010], for players to interact with each other in location-based games [Wikipedia 2013] [Nintendo 2012] [Sony 2013], and for taxicabs to send status to other nearby taxicabs to enable a real-time distributed dispatching [Zhang and He 2012]. This quickly collected neighbor information allows applications to effectively collaborate among participating devices.

On the other hand, in the above applications, radios in the mobile devices are usually duty cycled between several modes to save energy or bandwidth. For example, sensor nodes have to alternate their radios between an inactive mode and an active mode to save energy due to the fact that most sensor nodes are powered by batteries. The duty cycling scheme prolongs the devices' lifetime, however, they pose a significant issue for the neighboring devices to find each other, since the neighboring devices may not enter the active mode at same time for a long time due to low duty cycles (*e.g.*, 1%), thus incapable of finding each other by communication in time.

To address this issue, several state-of-the-art discovery protocols for wireless sensor networks [Tseng et al. 2002] [Zheng et al. 2003] [Dutta and Culler 2008] [Kandhalu et al. 2010] [Purohit et al. 2011] [Bakht et al. 2012] have been proposed to achieve a bounded discovery latency. We found, however, that current protocols 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, distracts user's attention. One could argue that adjusting duty cycles of existing solutions [Tseng et al. 2002] [Zheng et al. 2003] [Dutta and Culler 2008] [Kandhalu et al. 2010] [Purohit et al. 2011] can reduce delay in a discovery when so desired. These schemes, however, require coordinated changes of duty cycle patterns, a requirement only suitable for the networks where a user owns the whole network and can change all devices' duty cycles collaboratively, *i.e.*, sensor networks. 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 has to 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. Thus, we argue that allocating duty cycles continuously in advance of user demands is wasteful.

To address the above two challenges, 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 [Dutta and Culler 2008], 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 supports 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 on-demand 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 deterministic 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 deterministic discovery protocols under different duty cycle patterns.

— We propose a concept of *spatial-temporal coverage* and define a model to quantify the effectiveness of each slot in the acceleration of discovery. The model is fully distributed and leverages only information in neighbor tables of known neighbors. It does not make any assumptions regarding radio or mobility models.

— Based on this coverage, we design an agile online scheduling algorithm to decide additional active slots under a given energy budget. Comparing our online scheduling to its theoretically optimal Oracle version, we prove that our online scheduling is *competitive* by obtaining its competitive ratio ρ, which indicates that our online scheduling algorithm has bounded performance compared to its Oracle version.

— We present a neighbor verification mechanism and a proactive rendezvous maintenance mechanism, which utilizes the online information about common neighbor reduction as a hint to infer the leaving of a neighbor, and then initiates a binary selection for additional active slots to proactively accelerate the detection process about the leaving of the neighbor.

— We test *Acc* at three scales of networks: (i) a small-scale testbed experiment with 11 TelosB devices, (ii) a middle-scale simulation with 100 mobile devices, and (iii) a large-scale trace driven evaluation with 14,000 vehicles. The results show that *Acc*-assisted schemes reduce the latency by 51.8% when consuming the same energy.

— To prove the real-world value of *Acc*, we propose a *Crowd-Alert* application to show how *Acc* can be employed by taxicab drivers to select a direction with fewer competing taxis or more potential passengers. We further evaluate *Crowd-Alert* based on a 280 GB dataset consisting of 6 months of GPS traces of more than 14,000 taxis in Shenzhen, which is the most crowded city in China with 17,150 people per KM^2 [Sasin 2012]. Our application demonstrates that a smart driver increases the possibility of picking up a passenger based on an accelerated discovery, which makes drivers to quickly learn the distributions of potential passengers and competing taxicabs.

The paper is organized as follows. Section 2 introduces related work. Section 3 shows background. Section 4 provides our motivation. Section 5 proposes *Acc* design. Sections 6 and 7 present our implementation and simulation. Section 8 demonstrates *Acc*'s application in a taxi-dispatch system. Section 9 concludes the paper.

2. RELATED WORK

The 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*.

Probabilistic. The probabilistic protocols, *e.g.*, Birthday protocol [McGlynn and Borbash 2001], assign different probabilities for sending, receiving, and sleeping in individual slots. Due to Birthday Paradox [Mitzenmacher and Upfal 2007], such probabilistic schemes offer very good performance in the average discovery latency. But their major limitation is an unbounded worst-case discovery latency, which leads to a long tail on discovery probabilities over time. Moreover, Birthday protocol concludes that this discovery scheme aims for the stationary networks, instead of the mobile networks.

Quorum-Based. 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 [Tseng et al. 2002] [Zheng et al. 2003]. Although some work has been proposed to support asymmetric duty cycle patterns, they can support only two different duty cycle patterns [Lai et al. 2010]. Again, the quorum-based discovery protocols are also primarily proposed for stationary networks where energy is the most pressing concern, not mobility.

Deterministic. The deterministic protocols are most closely related to our work [Dutta and Culler 2008] [Kandhalu et al. 2010] [Purohit et al. 2011] [Bakht et al. 2012]. 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 [Niven and Zuckerman 1991], the devices would have bounded discovery latencies. In Disco [Dutta and Culler 2008], each device selects two prime numbers and generates its period independently based on these numbers. To improve Disco's performance, U-Connect [Kandhalu et al. 2010] proposes an activation pattern using one prime and has a shorter latency, especially in asynchronous symmetric networks. Further, WiFlock [Purohit et al. 2011] combines discovery and maintenance using a collaborative beaconing mechanism with time synchronization. More recently, Searchlight [Bakht et al. 2012] is proposed to leverage the constant offset between periodic awake slots to design a simple probing-based approach to ensure discovery.

Summary: Our work presents a different design architecture than the aforementioned three categories and serves as a middleware for deterministic neighbor discovery schemes. We utilize an existing deterministic discovery protocol (*e.g.*, Disco) to guarantee a bounded discovery latency by maintaining original active slots. Built upon the utilized protocol, our design adds only new active slots in addition to the slots specified by the utilized 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 the previous discovery designs.

3. PRELIMINARIES FOR NEIGHBOR DISCOVERY

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

Note that 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 GPS based synchronization schemes are part of solutions [Liu et al. 2004] [Jun et al. 2006], but they are typically too energy expensive to be implemented on battery powered mobile sensors [Elson and Römer 2003] or smartphones [Paek et al. 2010]. Therefore, the devices usually decide their schedules based on a distributed yet coordinated duty cycle pattern. Specifically, 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 [Dutta and Culler 2008].

Although our *Acc* can work with a wide range of protocols, for the sake of clarity in this paper we use Disco [Dutta and Culler 2008] as a representative example. In the evaluation, we will show how *Acc* works with WiFlock [Purohit et al. 2011], U-Connect [Kandhalu et al. 2010] and Searchlight [Bakht et al. 2012] as well. Specifically, Disco employs the Chinese Remainder Theorem [Niven and Zuckerman 1991] to guarantee a discovery latency bound. Though in the real implementation, Disco selects two different primes for a device to solve the issue of two devices having the same prime, 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.

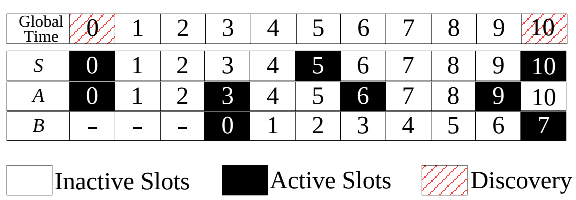

Fig. 1. Neighbor Discovery Process

In Figure 1, devices S, A, and B start their local timers at global time 0, 0, and 3, respectively. According to Disco, S discovers devices 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 discovery protocols only assume that the slots at individual devices have equal lengths [Dutta and Culler 2008]. By sending two messages at the beginning and end of an active slot, they do not require aligned slots and are robust to clock drift. The perfect alignment in Figure 1 is for illustrations.

The rationale behind the duty cycling based neighbor discovery is to ensure that the distributed asynchronous devices have their active slots quickly overlapped. Without further information, the neighbor discovery protocols have to be cautious about turning nodes' radios into active slots, which may waste the energy.

4. MOTIVATION: WHY WE NEED *ACC*?

Our work is motivated by the observation that current state-of-the-art neighbor discovery schemes suffer from long discovery latencies due to duty cycling for energy efficiency. In many mobile applications, however, neighbor discovery has to be fast enough to enable crucial responsive user experiences. Unfortunately, for traditional discovery schemes, its design objective is to discover neighbors with a more energy-efficient method, no matter how long it will take, as long as it is bounded.

We utilize a GPS dataset of 14,000 taxicabs to simulate a real-world mobile network (the detailed setting is given in Section 8) to investigate the performance of the neighbor discovery protocols. As in Figure 2, we plot results on the cumulative distribution function (*i.e.*, CDF) of latency for Disco [Dutta and Culler 2008]. As shown by point X, Disco discovers more than 30% of neighbors after a latency of 3 mins; as shown by point Y, Disco discovers more than 70% of neighbors after a latency of 6 mins.

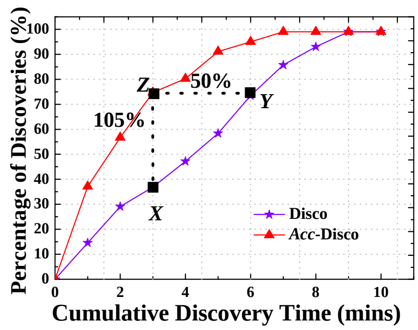

Fig. 2. Motivation

Based on the above evaluation, we find that though such a long discovery latency ensures the energy saving, it poses a significant challenge for interactive applications where energy is important but not the most pressing concern. Thus, in these applications, when needed, an on-demand fast neighbor discovery has to be performed 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 of *Acc* is to more efficiently utilize the additional energy budget to accelerate the discovery process, compared to the current designs with the same amount of energy.

In Figure 2, to visually show our design objective, we plot the curve of *Acc*-Disco where *Acc* works together with Disco to accelerate the discovery. To make the comparison fair, we run *Acc*-Disco at the same duty cycle with Disco. But in *Acc*-Disco, the half of the duty cycle is allocated to Disco for bounded latency and another half of the duty cycle is allocated to *Acc* for acceleration purpose. Therefore, Disco and *Acc*-Disco have the same total energy budget. The system details are given in Section 8. As shown by point Z, *Acc*-Disco discovers more than 70% of neighbors after a latency of 3 mins. Thus, comparing point Z to X, under the same latency, our *Acc* assists Disco to achieve more discoveries by a maximum of 105%; whereas comparing point Z to Y, to discover the same number of neighbors, our *Acc* assists Disco to accelerate its discovery process by a maximum of 50%.

Based on the above observations, our 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 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

As in Figure 3, based on the location of *Acc* in the whole networking architecture, we introduce the main idea of *Acc* as follows.

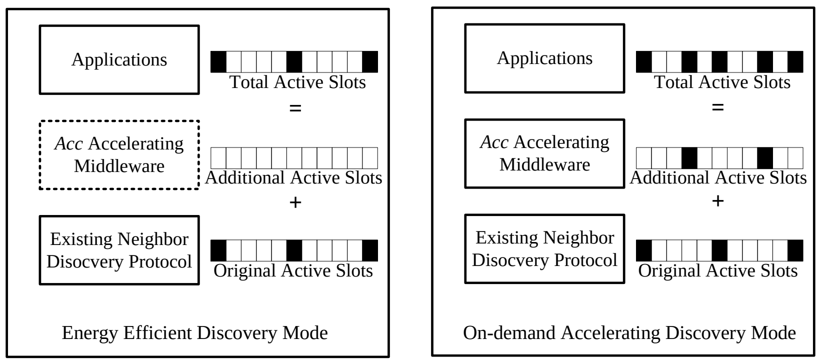

Fig. 3. Acc in the Architecture

In Figure 3, an effective existing discovery protocol, *e.g.*, Disco, has already been installed in each device. This existing protocol provides the neighbor information to the upper applications. Our *Acc* serves as a middleware between the existing neighbor discovery protocol and applications. 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, and *Acc* is completely transparent, *i.e.*, a device only turns on the radio at the active slots (*i.e.*, the black cells) indicated by the existing discovery protocol as in the left of Figure 3; otherwise, S enters the second mode, concurrently performing *Acc* and the underlying discovery protocol for both the acceleration and the bounded latency, *i.e.*, turning on the ratio at the active slots indicated by both the existing discovery protocol and *Acc*. The detailed operations of a device in these two modes are given as follows.

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. (i) 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. (ii) 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 with an additional energy budget. In this mode, besides original active slots, S also becomes 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 4 gives an example of the indirect discovery.

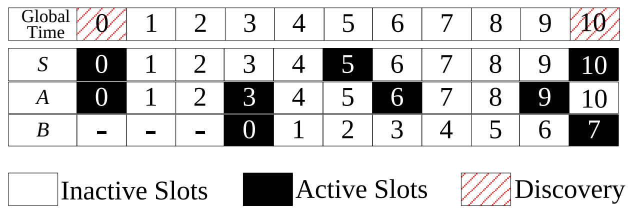

Fig. 4. Indirect Discovery

After the discovery of a device A in the global time 0, if S can select one additional active slot between the global time slot 1 to 10, S would select slot 6 for possible *indirect* discoveries via A, since (i) S knows that A will become active in slot 6 after the initial discovery, and (ii) 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 the 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: (i) how to evaluate the effectiveness of all potential active slots, and (ii) 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 marginal cost (*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 the 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 the 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.

Nevertheless, we observe that not all known neighbors at S are equally valuable for indirect discoveries. Specifically, S should favor those important known neighbors exhibiting both *temporal diversity* and *spatial similarity* to S. The temporal diversity indicates that in 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 is assigned to a larger gain.

Note that the temporal diversity and spatial similarity of the neighbors indicate their discovering capability for the discovering devices in terms of the *temporal-spatial* coverage. An example of temporal-spatial coverage is given in Figure 5.

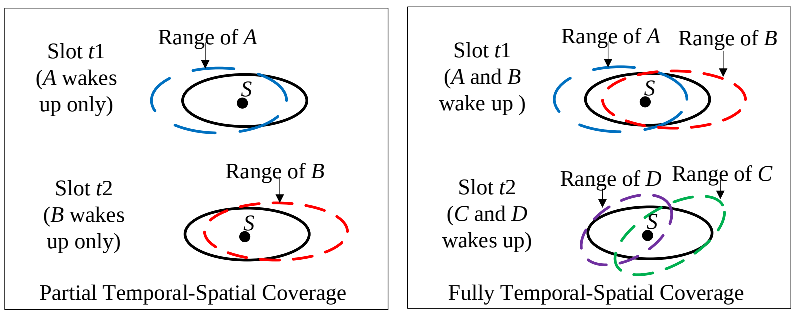

Fig. 5. Temporal Spatial Coverage

In the left of Figure 5, we show a partial temporal-spatial coverage where the discovering device S has two known neighbors A and B, and their radio ranges are shown in the figure. Based on their waking up schedules, S is inactive in both slots t_1 and t_2; A is active during slot t_1 only; B is active during slot t_2 only. Thus, A and B can only temporally and spatially *cover* partial neighborhood of S (*i.e.*, discovering S's neighbors), when S is inactive during slots t_1 and t_2. This is because during t_1, A cannot find S's active neighbors who are inside S's range, but outside A's range; similarly during t_2, B cannot find S's active neighbors who are inside S's range, but outside B's range. However, in the right of Figure 5, we find a full temporal-spatial coverage for S by known neighbors A, B, C and D in another setting. During S's inactive slots t_1 and t_2, any S's unknown active neighbor will be discovered by A, B, C or D. The discovery result will be forwarded to S, when S makes rendezvous with these neighbors later.

As follows, we introduce the details of how to use the temporal diversity and the spatial similarity to calculate the slot 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 the 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 6 shows an example of temporal diversity.

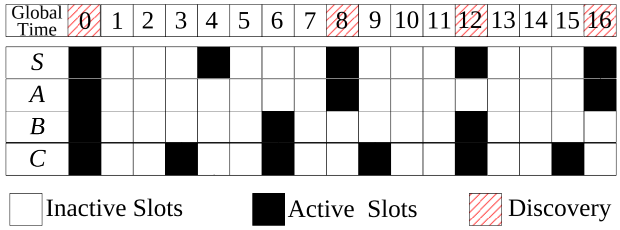

Fig. 6. Example of Temporal Diversity

In Figure 6, whenever A becomes active, S also becomes active, so the temporal diversity between them is limited. Since A can only discover neighbors in the slots where S also does, there is limited information that A can learn but S cannot. But a device C frequently becomes active in the slots where S is inactive (*e.g.*, slots 3, 6, 9 and 15). Given a slot t, the more frequently C becomes active before t, the larger the possibility that C has more information on the potential neighbors not yet known to S. Thus, 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 the 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 $\alpha^{(i,j)}_{t_0\to t}$, 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.

$$\alpha^{(i,j)}_{t_0\to t} = \frac{|m^{(i,i)}_{t_0\to t}| - |m^{(i,j)}_{t_0\to t}|}{t - t_0}, \tag{1}$$

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

As in Figure 6, we show how to obtain $\alpha^{(i,j)}_{t_0\to t}$. Assuming devices 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 $\alpha^{(A,S)}_{1\to 6} = \frac{0}{5}$, $\alpha^{(B,S)}_{1\to 6} = \frac{1}{5}$, and $\alpha^{(C,S)}_{1\to 6} = \frac{2}{5}$, 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 $\beta^{(i,j)}_{t_0}$, 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.

$$\beta^{(i,j)}_{t_0} = \frac{|n^{(i,j)}_{t_0}|}{|n^{(j,j)}_{t_0}|}, \tag{2}$$

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

Figure 7 shows an example about how to obtain $\beta_{t_0}^{(i,j)}$.

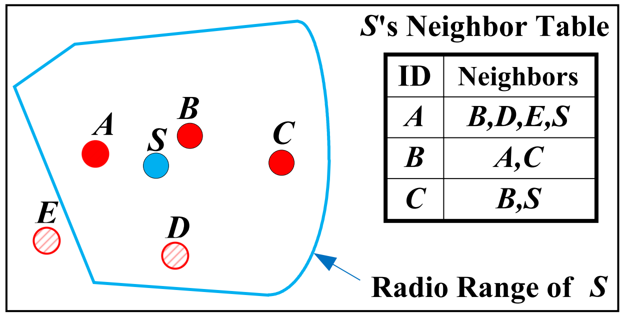

Fig. 7. Example of Spatial Similarity

In Figure 7, at t_0 = slot 1, among three discovered neighbors (*i.e.*, A, B, and C), S shares two, three and two neighbors with A, B and C, respectively, including a neighbor itself. So, for directly discovered neighbors A, B, and C, S calculates $\beta_1^{(A,S)} = \frac{2}{3}$, $\beta_1^{(B,S)} = \frac{3}{3}$ and $\beta_1^{(C,S)} = \frac{2}{3}$. For indirectly discovered neighbors, *e.g.*, D, S calculates $\beta_1^{(D,S)} = \frac{1}{3}$, since only one (*i.e.*, device A) out of three 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 the 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 range (*i.e.*, spatial coverage) of S from slot t_0 to t (*i.e.*, temporal coverage), *e.g.*, the fully temporal-spatial coverage in the right of Figure 5. 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 the 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 6 and the neighbor table in Figure 7, assuming that t_0 is slot 1, a discovering device S calculates slot 6's slot gain according to the follow formula.

$$\gamma_{1\to 6}^{(S)} = \alpha_{1\to 6}^{(A,S)} \beta_1^{(A,S)} + \alpha_{1\to 6}^{(B,S)} \beta_1^{(B,S)} + \alpha_{1\to 6}^{(C,S)} \beta_1^{(C,S)} = \frac{0}{5}\frac{2}{3} + \frac{1}{5}\frac{3}{3} + \frac{2}{5}\frac{2}{3} = \frac{7}{15} \tag{4}$$

5.3. Online Activation Scheduling

In the previous subsection, we present the method to calculate the slot gains for all the slots based on the neighbor table of a discovering device. According to the obtained slot gains, in this subsection, we first present our online scheduling algorithm, given an fixed duty cycle budget $\mathbb{B}$. This online algorithm outputs a slot sequence for additional activations and updates this sequence consistently based on the latest yet incomplete neighbor table. Then by comparing this online algorithm to its optimal Oracle version, we theoretically analyze the proposed algorithm to show its performance via a concept called competitive ratio.

5.3.1. Scheduling Algorithm. In our scheduling algorithm, a discovering device S decides an additional active slot sequence $\mathbb{AS}$ which includes several additional active slots, according to three inputs as follows.

(i) Additional Energy Budget $\mathbb{B}$. Given $\mathbb{B}$ in terms of additional duty cycles, *e.g.*, $\frac{2}{11}$ beyond what has already been consumed by an underlying discovery scheme, S performs discoveries 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.
(ii) 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.
(iii) 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: (i) $\mathbb{B}$; (ii) $n_{t_0}^{(S,S)}$; (iii) 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, according to S's current neighbor table $n_{t0}^{(S,S)}$ 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 8 gives an example of this algorithm.

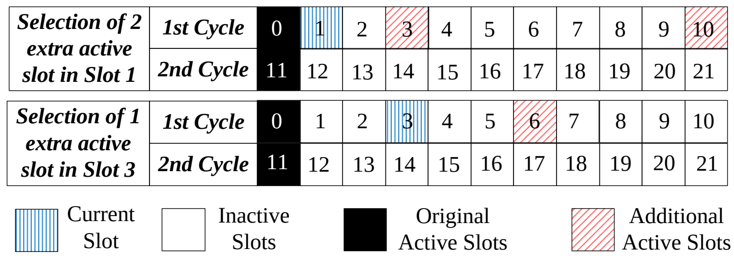

Fig. 8. Example of Activation Scheduling

Suppose that $t_0 = 1$, the 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 slots 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 $n_{t_0}^{(S,S)} = N^{(S,S)}$.

5.4. Neighbor Verification

In the previous subsection, we introduce how to use online activation scheduling to accelerate the process of neighbor discovery by indirect discovery. In the scenario of mobile multi-hop networks, for a discovering device, a neighbor's neighbor may not be its neighbor when the discovering device indirectly discovers it. This discovery would be a false positive. Therefore, we propose a passive neighbor verification technique to verify whether indirectly-discovered neighbors are actually one-hop neighbors.

In this paper, we define a neighbor of a device S as a device who is continuously in the communication range of S at least a time period p, which is the discovery latency bound of an underlying neighbor discovery scheme. Two devices just transitorily were in communication ranges of each other cannot be seen as neighbors, since they cannot be discovered by each other. Therefore, a neighbor will be discovered by *Acc* in advance or by an underlying protocol eventually. If two devices discover each other and then move out of communication rages of each other within a time period p, then they are not considered as neighbors (false position) and will be removed.

During a discovery process, since every device would broadcast its neighbor table to its neighbors during the discover process, a discovering device would have duty cycle patterns of indirectly-discovered neighbors whether they are one-hop neighbors or two-hop neighbors. Based on these duty cycle patterns, the discovering device would know when an indirectly-discovered neighbor will become active and broadcast messages to its neighbors. Therefore, in our passive neighbor verification, the discovering device would become active in the active slots of every indirectly-discovered neighbor and listen to the channel for its messages for a time period of p. If the discovering device can receive messages from this neighbor for a time period of p, then it indicates this indirectly-discovered neighbor is an actual one-hop neighbor. In contrast, if the discovering device cannot receive messages from this neighbors for a time period of p, then it indicates this indirectly-discovered neighbor is not an actual one-hop neighbor.

Figure 9 gives an example about the neighbor verification process for indirectly discovered neighbors. Assume we have a discovering device S. Based on its direct neigh-

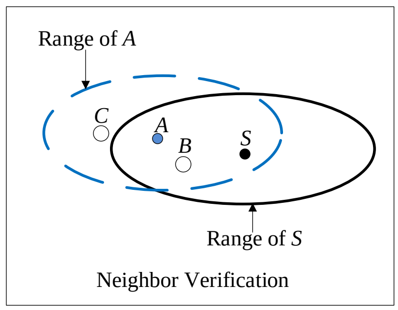

Fig. 9. Neighbor Verification

bor A, the discovering device S indirectly discovers two neighbors, *i.e.*, B and C. B is a one-hop neighbor of S and C is a two-hop neighbor of S. Based on the duty cycle patterns of B and C, S also becomes active during active slots of B and C, after the initial indirect discovery of them. During these active slots of B and C, S tries to receive their messages passively, in addition to its own active slots. Because B is within S's communication rage and C is out of S's communication rage, S can receive the message from B, but not from C. Therefore, S have verified B is a one-hop neighbor and C is a two-hop neighbor.

5.5. Proactive Online Rendezvous Maintenance

The neighbor discovery is a process to identify neighboring devices so that a device S can send messages to other devices in its neighborhood. Whereas, the rendezvous maintenance is a process where S contacts with its discovered neighbors regularly to verify and maintain the neighboring relationship by timely detecting that the neighbors are still in the neighborhood or not. But such a discovered neighborhood relationship among S and its neighbors is only temporal in mobile applications, because both the neighbors and S are moving around and will leave the radio ranges of each other after a period of time. The leaving of a known neighbor A is detected by S through a failure to receive the discovering message from A in the slot where S and A both become active, according to the schedule obtained when they first discover each other. After such a failure, S just drops off A from its neighbor table. The above scheme is the normal rendezvous for a device S to maintain its neighbor table up to date in existing protocols where the rendezvous is treated as a “rediscovery” during which a device and its known neighbors are both in the active slots again.

In this work, we argue that this passively rediscovery based rendezvous takes a long-time delay to detect the fact that a neighbor A has already leaved S's ratio range, *e.g.*, a detecting delay for two devices with 1% duty may take up to 101×101 slots. Such a long delay may be acceptable for delay tolerant applications, *e.g.*, sensor networks, but it is typically not acceptable in the interactive applications where the leaving of a neighbor should be proactively detected as soon as possible, instead of passively depending on the rediscovery. In our Acc, we proactively maintain the rendezvous to reduce the detecting delay in an on-demand method if it is required by users.

The rationale of our online rendezvous maintenance mechanism is as follows. We divide the rendezvous maintenance into two subobjectives: when to proactively maintain the rendezvous and how to proactively maintain the rendezvous. Since our Acc is designed as a transparent middleware, we do not change the schedule of neighboring devices. Thus, a naive yet safe method is that right after the discovering device S discovers a neighbor A, S becomes active in every slot where A becomes active to verify if A is still a neighbor of S or not. This naive method is the quickest method to transparently detect the leaving of A without the cooperation from A, but this method involves too much energy consumption for S, since S has to wake up at every active slot of A.

To address this issue, in Acc, S first uses the reduction of the common neighbors of S and A between two normal rendezvous as a hint to initiate a proactive rendezvous maintenance regarding to A. After the beginning of the maintenance, S utilizes a binary selection to find the some active slots of A for the additional wakeups to quickly detect the fact that A leaves the range of S or not. If A is still in S's range but the reduction of the common neighbors continues, S continues to select additional wakeup slots until A leaves the range of S or the reduction stops. As follows, we give the details about when and how to process the proactive rendezvous maintenance.

5.5.1. When to initiate proactive online rendezvous maintenance?. After the initial discovery of A, if the proactive online rendezvous maintenance is required by users, S compares the common neighbors between itself and A after every normal rediscovery about A. If S detects a reduction of the common neighbors, S initiates the proactive rendezvous maintenance regarding A. Figure 10 gives an example about the reduction of the common neighbors for a discovering device S and its neighbor A.

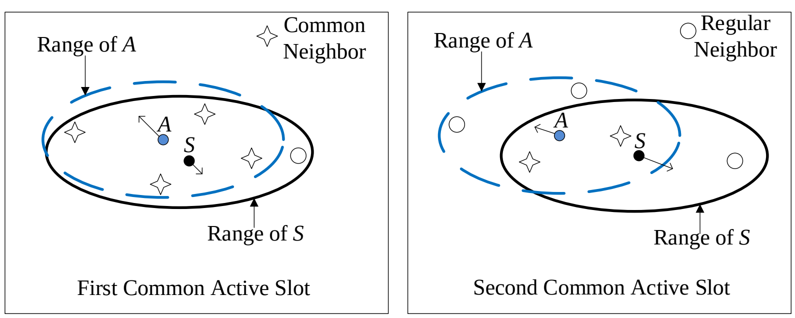

Fig. 10. Reduction of Common Neighbors

In Figure 10, in the first common active slot, S and A have four common neighbors, while at the second common active slot, S and A have only two common neighbors (due to the movements of S and A). The reduction can be obtained by their neighbor tables they broadcasted in the common active slot. Such a reduction of the common neighbors indicates that A is leaving the range of S. Thus, S initiates the proactive rendezvous maintenance regarding to A after the second common active slot. The rationale between method is the fewer the common neighbors, the farther the distance between S and A, the more likely A is leaving the range of S.

5.5.2. How to initiate proactive online rendezvous maintenance?. In the rendezvous maintenance regarding to A, S selects some active slots of A for the additional wakeup (before the next normal common active slot) to reduce the detecting delay for the fact that A leaves the radio range of S. We utilize a binary selection to choose these slots. Figure 11 gives an example about S selecting additional wakeup slots to detect the leaving of A.

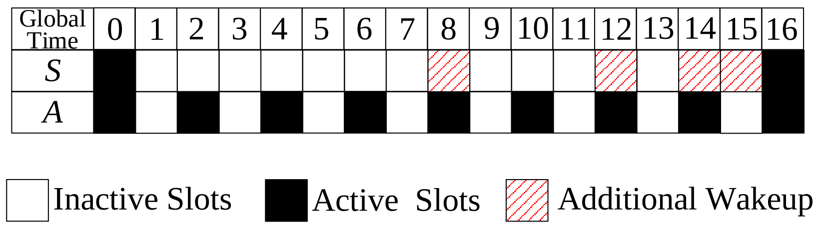

Fig. 11. Selection of Additional Wakeup for Rendezvous Maintenance

In Figure 11, S detects the reduction of common neighbors between S and A at slot 0, and initiates the proactive online rendezvous maintenance. Further, we assume a situation where the reduction of common neighbors between S and A is continuously detected.

— S calculates A's active slots (*i.e.*, slots 2, 4, 6, 8, 10, 12 and 14) before the next normal rediscovery at slot 16;
— S utilizes a binary selection to obtain slot 8 as an additional wakeup, which is in the middle of current slot 0 and the next normal rediscovery at slot 16;
— After waking up at slot 8, S continues to detect the reduction of the common neighbors, so S utilizes the same binary selection again to obtain slot 12 as an additional wakeup, which is in the middle of current slot 8 and the rediscovery slot 16;
— After waking up at slot 12, S selects slot 14 as an additional wakeup, which is in the middle of current slot 12 and the normal rediscovery slot 16;
— Similarly, after waking up at slot 14, S selects slot 15 as an additional wakeup, which is in the middle of current slot 14 and the normal rediscovery slot 16;
— This process continues until no reduction of common neighbors is detected or after A leaves the radio range of S;
— Finally, S drops off A from its neighbor table, if A leaves the radio range of S.

In the above method, S is based on the online information about the reduction of the common neighbors to proactively accelerate the rendezvous maintenance. The rationale behind this method is the fewer the common neighbors, the more likely A leaves the radio range of S.

Note that even without the above proactive online rendezvous maintenance, our regular accelerated discovering process by Acc implicitly expedites the delay of the detection for the fact that a neighbor leaves the ratio range of a discovering device, This is because a discovering device will wake up more in the active slots of the known neighbors for the indirect discovery, according to the design of Acc. Thus, if a known neighbor of S is not broadcasting in the active slot when it is supposed to be, then S removes this neighbor from the neighbor table, which enables S to more quickly detect the leaving of its neighbors, although the introduced proactive online rendezvous maintenance can further accelerate the detection.

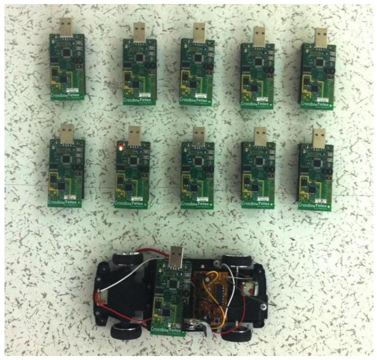

Fig. 12. Testbed Setup

Name \ Energy	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**

Fig. 13. 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 [Dutta and Culler 2008] and WiFlock [Purohit et al. 2011]. To verify whether the accelerated neighbor discovery would perform well on resource-constrained sensor nodes, 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. This mobile node introduces the relative mobility between a discovering device and its neighbors, which is to verify the mobility would not affect the neighbor discovery itself. The testbed is shown in Figure 12.

At individual devices, we set the time slot length to be $25ms$ for two reasons. (i) For direct discovery, a smaller slot leads to a faster discovery, but a too-small slot ($< 5ms$) leads to the jitters introduced by the TinyOS timer library [Dutta and Culler 2008]. (ii) For indirect discovery, a bigger slot reduces collisions of messages and enables 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 $80\mu s$ [Purohit et al. 2011], but in our paper we implement WiFlock only on a standard hardware to examine the principle of its collaborative beaconing mechanism. In our experiment, all schemes have the same energy budget (both original and additional) for devices to ensure a fair comparison. But different schemes use the same energy budget differently in terms of selecting active slots. 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 [Dutta and Culler 2008].

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 13. 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: (i) the percentage of discoveries with respect to cumulative discovery time; (ii) the number of discovered devices in different time intervals; (iii) the average discovery latency in different duty cycles. The first two metrics are to verify the effect of *Acc*'s assistance to the existing schemes in the acceleration of discovery process in the first subsection. The third metric is to

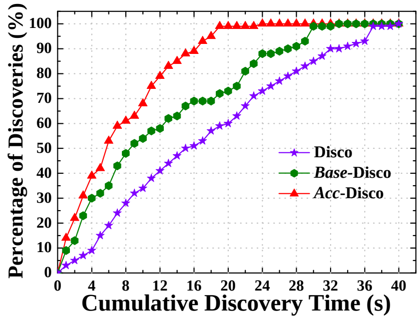

Fig. 14. Disco CDF

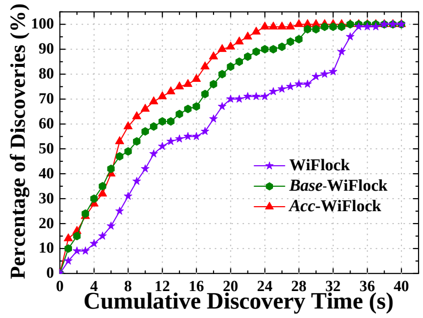

Fig. 15. WiFlock CDF

verify the effect of different duty cycles on the average discovery latency in the second subsection.

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 14 plots the acceleration effect of Disco. In Figure 14, 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 15, we observe the similar results as in Figure 14. 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 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 15, 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, achieves further improvement. This is because by simply measuring the slot gain as the number of active neighbors, *Base*-WiFlock increases performance to

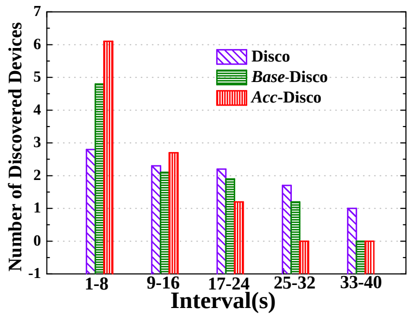

Fig. 16. Disco Distribution

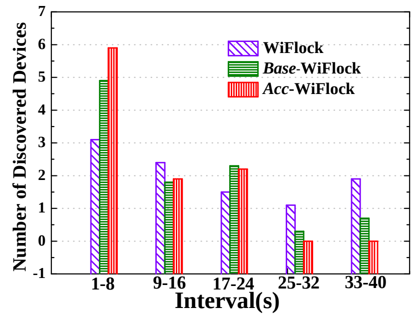

Fig. 17. WiFlock Distribution

a certain level but cannot make a device become active at the most effective slots, as *Acc*-WiFlock does.

Figures 16 and 17 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 16 and 17, 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 indeed considers 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 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 to 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* further accelerates the discovery process.

6.2. Impact of Duty Cycle

Figures 18 and 19 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. 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.

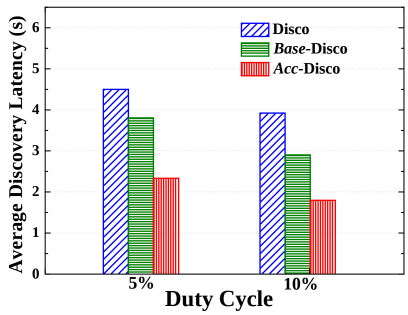

Fig. 18. Disco Latency

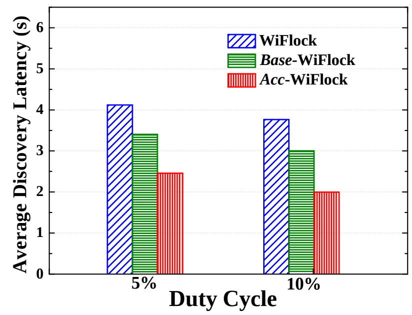

Fig. 19. WiFlock Latency

It also shows that the performance gain between *Acc*- 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 and the original version in WiFlock, *i.e.*, 39%, is smaller than that in Disco, *i.e.*, 47.7%. Note that even with double duty cycles, the average discovery latency does not reduce significantly in all three protocols. This is because by increasing duty cycles Disco guarantees the proportionally-reduced worst-case latency, instead of the average latency.

From Figures 18 and 19, we conclude that when devices become more active, *Acc* more effectively assists the discovering device to accelerate the discovery process by leveraging the known neighbors to discover unknown neighbors.

7. SIMULATION EVALUATION

To evaluate *Acc* serving as an accelerating middleware to support different protocols in larger-scale networks, we simulate *Acc* with four discovery protocols, Disco [Dutta and Culler 2008], U-Connect [Kandhalu et al. 2010], WiFlock [Purohit et al. 2011] and Searchlight [Bakht et al. 2012]. In our 30-mins simulation, 100 mobile devices are uniformly deployed in a square area of $200m \times 200m$. The radio ranges of devices are set from $20m$ to $110m$, which lead to average device densities from 3.6 to 55.36. We use a non-trivial pure random waypoint model as a mobility model [Alparslan and Sohraby 2007], with an average velocity $1m/s$. In addition to the metrics we investigate in the testbed experiment, we evaluate the rendezvous maintenance in the last subsection.

Note that in a mobile multi-hop network, neighboring relations are consistently changing, and it is extremely costly in terms of energy to keep neighbor tables up to date, *i.e.*, immediately discovering a device when it is in one device's communication range. In the evaluation, we define a neighbor of a device A as a device who was continuously in the communication range of A at least a time period p, which is the discovery latency bound of an underlying neighbor discovery scheme. Two devices just transitorily were in communication ranges of each other cannot be seen as neighbors, since they cannot be discovered by each other. Therefore, a neighbor will be discovered by *Acc* in advance or by an underlying protocol eventually. If two devices discover each other and then move out of communication rages of each other within a time period p, then they are not considered as neighbors (false position) and will be removed. In our experiment, all schemes have the same energy budget (both original and additional) for devices to ensure a fair comparison. But different schemes use the same energy budget differently in terms of selecting active slots. We test *Acc* with two metrics, *i.e.*, the percentage of discoveries and discovery latency. For both of them, the discovery delay is calculated from the point when a node is within a discovering node's rage until it was discovered by the discovering node.

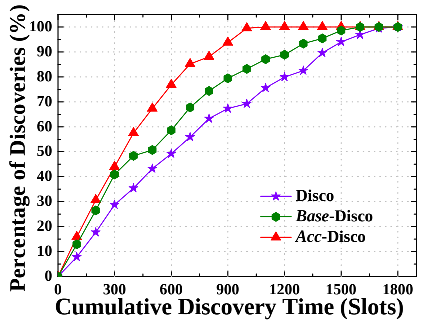

Fig. 20. Disco CDF

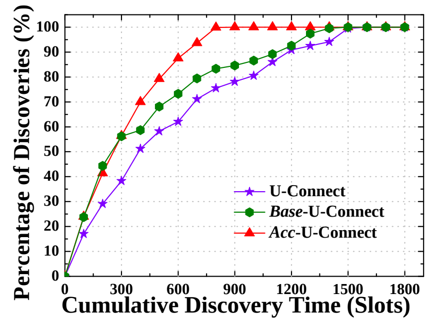

Fig. 21. U-Connect CDF

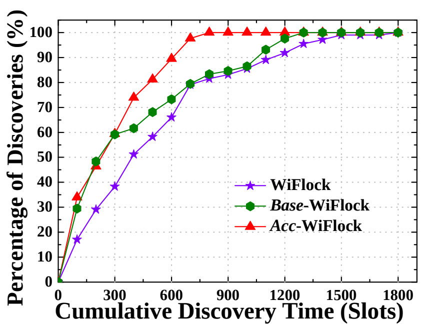

Fig. 22. WiFlock CDF

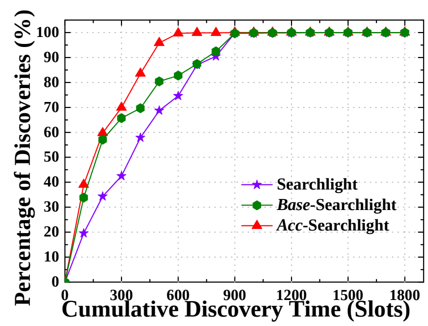

Fig. 23. Searchlight CDF

7.1. Effectiveness in Acceleration of Discovery

In Figure 20, we plot the percentages of discoveries in terms of cumulative discovery time. From Figure 20, 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. This is because some nodes are not neighbors at the beginning of the experiment, but become neighbors later. 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 verify the effectiveness of the temporal diversity and spatial similarity as a slot gain.

For percentage of discoveries, some nodes are not neighbors at first place and then become neighbor due to mobility. We use the cumulative time to track the percentage of actual neighbors being discovered at certain time.

Similarly, in Figures 21, 22 and 23, we plot the same sets of curves for U-Connect, WiFlock and Searchlight. We also observe similar performance trends as in Figure 20. For example, in Figure 21, 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 22, we still find a performance gain between

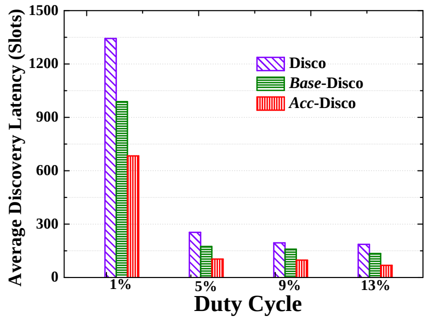

Fig. 24. Disco Latency

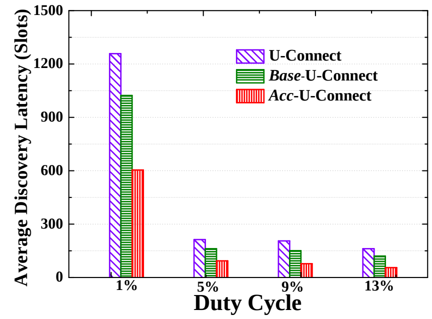

Fig. 25. U-Connect Latency

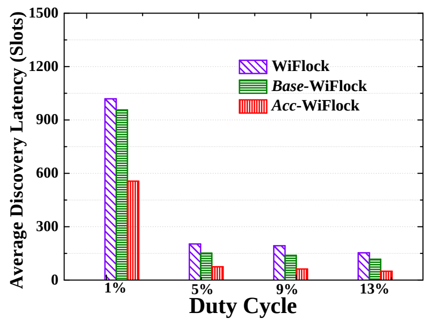

Fig. 26. WiFlock Latency

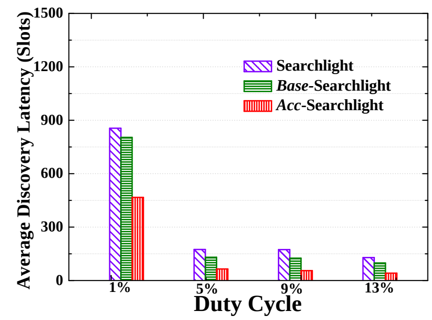

Fig. 27. Searchlight Latency

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. In Figure 23, we find that a performance gain between *Acc*-Searchlight and other versions of Searchlight becomes smaller, due to the fact that Searchlight significantly reduces the worse-case discovery delay.

From results in Figures 20, 21, 22 and 23, we suggest that *Acc* serves as an accelerating middleware for various schemes to accelerate the discovery process. Specifically, the performance gain of *Acc* is bigger in the early stage of 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 24, 25, 26 and 27.

In all figures, 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 four 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 24, 25, 26 and 27. we also observe that in general, as the duty cycle increases, the performance gain between versions with *Acc*

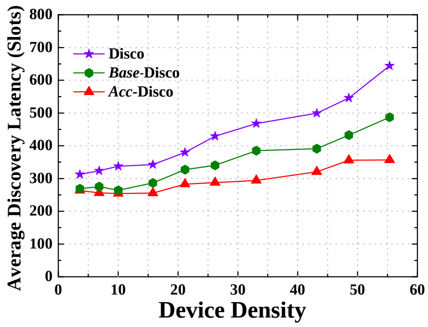

Fig. 28. Disco Latency

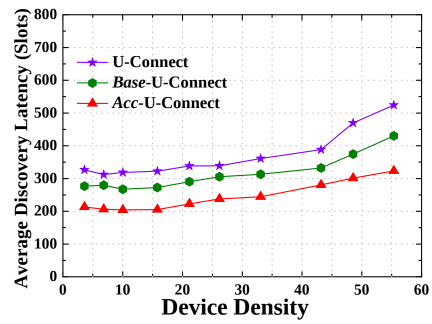

Fig. 29. U-Connect Latency

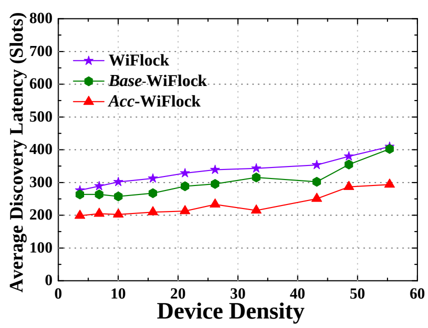

Fig. 30. WiFlock Latency

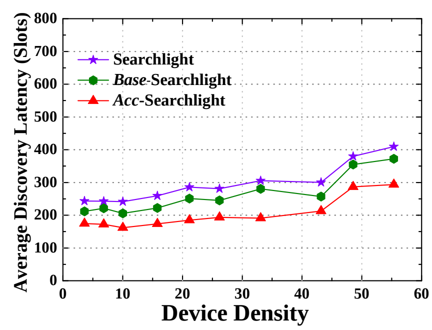

Fig. 31. Searchlight Latency

and original versions also increases. For example, in Figure 24, 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 the network become active more frequently, leading to more neighborhood information sharing. Among other three schemes, we find Searchlight has the best performance, which affirms our observation in the previous subsection.

Based on the above results, we conclude that the higher the duty cycle, the better the performance gain for *Acc* assisted scehmes. This is because with a higher duty cycle, a device can have more active slots which can be used for Acc for acceleration. But such acceleration effects are limited when the active slots are fewer in a lower duty cycle.

7.3. Impact of Device Density

In this subsection, we investigate the impact of the device density on the average discovery latency of four discovery schemes. The impact of device density on the average discovery latency is shown in Figures 28, 29, 30 and 31, respectively.

From all four figures, we find that as the device density increases, the average discovery latency increases for all four neighbor discovery protocols. This is due to the fact that at the higher densities, the devices have more neighbors, leading to more collisions and thus more time to find all the neighbors. 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 Figure 28

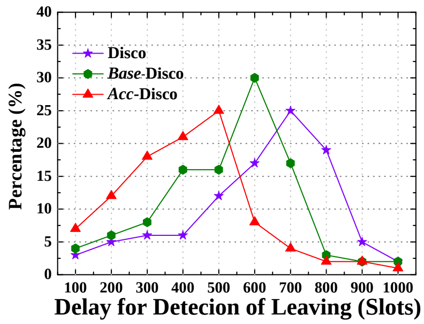

Fig. 32. Disco Latency

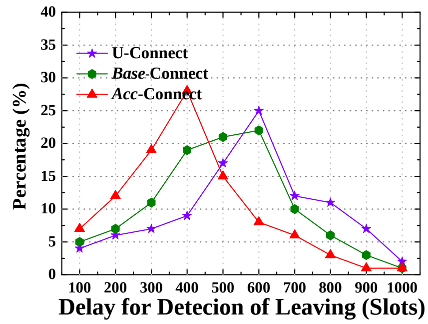

Fig. 33. U-Connect Latency

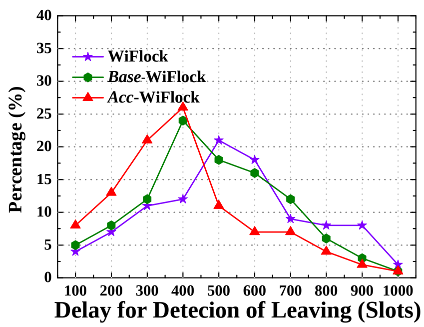

Fig. 34. WiFlock Latency

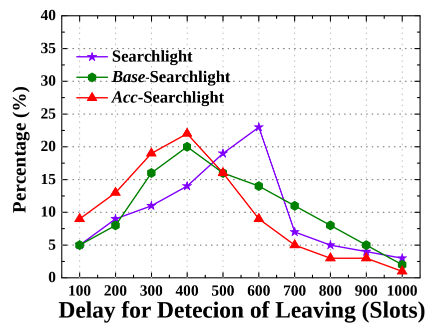

Fig. 35. Searchlight Latency

about Disco. This is because more known neighbor devices are able to share neighborhood information with discovering devices, thus accelerating the neighbor discovery process. We also observe the similar results in Figures 29, 30 and 31. For example, in Figure 31, when the average number of neighbors increases from 3.6 to 55.36, the average discovery latency in *Acc*-Searchlight and Searchlight increases to 290 slots and to 310 slots, respectively.

Based on the above results, we conclude that the higher the device density, the higher the average discovery latency. 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.

7.4. Effectiveness in Proactive Online Rendezvous Maintenance

In this subsection, we investigate the detection delay in the rendezvous maintenance mechanism in Figures 32, 33, 34 and 35. The legacy protocols, *e.g.*, Disco [Dutta and Culler 2008], utilize the rediscovery as the rendezvous maintenance. In contrast, Acc provides an online rendezvous maintenance mechanism for them to make a discovery device S to wake up at some additional slots to detect the neighbors who are believed to be leaving the radio range of S. In this paper, we use a metric called the detection delay to evaluate the performance of online rendezvous maintenance mechanism. It is given by the time difference from the slot when a neighbor leaves the radio range of S to the slot when S detects that this neighbor leaves. We use a baseline design called

$Base$ to show the effectiveness of Acc. The baseline selects additional wakeup slots in random, whereas Acc utilizes the introduced binary selection. We assume that a fast maintenance is required by users, and all protocols are under the same duty cycle for a fair comparison. Due to its proactive online rendezvous maintenance, the protocols assisted with Acc would have a better performance than the original protocols and $Base$ assisted protocols.

The distributions of the detection delay about Acc on different protocols are given Figures 32, 33, 34 and 35, respectively. We find that the Acc assisted protocols have the lowest average detection delay, which indicates that with the help of Acc, a neighbor discovery protocol can quickly detect the leaving of a neighbor, thanks to the common neighbor based maintenance mechanism. Further, Acc assisted protocols have a better performance than $Base$ assisted protocols, which is because Acc's binary selection is more effective than the random selection of $Base$. Based on the results, we conclude that Acc's proactive online rendezvous maintenance mechanism assists existing neighbor discovery to reduce the detection delay of the leaving of neighbors.

8. CROWD-ALERT APPLICATION

In this section, to prove the real-world value of Acc, 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 by an Acc assisted neighbor discovery, after they drop off passengers, *i.e.,* a faster neighbor discovery is demanded.

Based on the current setting of taxicab systems in metropolitan areas, every taxi is installed with a wireless communication interface (WiFi or WiMax) to send its locations and status (with or without passengers) back to base stations for accounting purposes. Thus, we envision a taxicab would primarily use its radio in the infrastructure mode since it has to communicate with a base station frequently for accounting. Only during gaps between the communications with base stations, a taxi can set its radio to the ad hoc mode for peer to peer neighbor discovery. So it leads to a duty cycle between the infrastructure and ad hoc mode not for energy but for radio usage. Based on broadcasted status of nearby taxis, a taxi driver 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. We will describe our application in further details and evaluate the efficiency of this scheme in discovering neighbors in a timely fashion.

8.1. Application Background

In our application, every taxicab broadcasts its own status record (*i.e.*, date and time, availability, direction, GPS coordinates, *etc*) to its neighboring taxis during the time it is not communicating with base stations. 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 metrics can maximize the probability of picking up the next nearby passengers.

Generally, the fewer the competing taxis, the higher the probability of picking up passengers. With distributedly collected status records about neighboring taxis, our

Fig. 36. App Screenshot

Crowd-Alert computes the location distribution of competing taxis that also aim to pick up new passengers. Similarly, the more the 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 obtain a distribution based on the indication is outside the scope of this paper. To focus on system levels, 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.

8.2. Application Evaluation

We embed *Crowd-Alert* function into one of our taxicab-booking app for the taxicab network in Shenzhen. In Figure 36, we show the app screenshots about finding a route, checking nearby taxicabs, and potential passengers.

But due to privacy reasons and limited installations, we cannot evaluate *Crowd-Alert* based on the data from the app in large scales. Instead, we evaluate the *Crowd-Alert* application using a real world dataset collected from taxis in Shenzhen during 6 months. We first introduce the dataset, and then present the evaluation results in terms of reduction of discovery latency and accelerating the navigation.

8.2.1. Dataset. The dataset consists of 6 months of GPS traces from $14,453$ taxis. The data is used by the government for the urban transportation pattern search. Each taxi uploads its records every 15 to 30 seconds, with each record consisting of the following parameters: (i) Plate Number; (ii) Date and Time; (iii) GPS Coordinates; and (iv) Availability: whether or not a passenger is in this taxi when the record is uploaded.

Taxicab Network Summary	
Collection Period	6 Months
Collection Date	01/01/12-06/30/12
Numbe of Taxicabs	14,453
Number of Passengers	98,472,628
Total Travel Distance	594,031,428 (KM)
Total Fare	2,255,052,932 (CNY)
Average Travel Distance	6.032 (KM)
Average Fare	22.9 (CNY)

Fig. 37. Statistics

Figure 37 summarizes details of the used datasets. Based on the above GPS trace records, we can obtain a location distribution of competing taxis or served passengers, as shown in Figures 38 and 39.

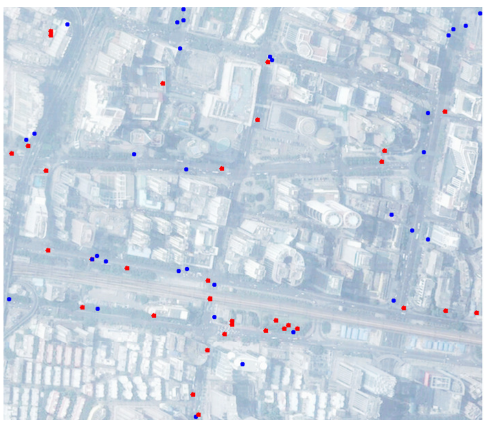

Fig. 38. Distribution of Competing Taxis

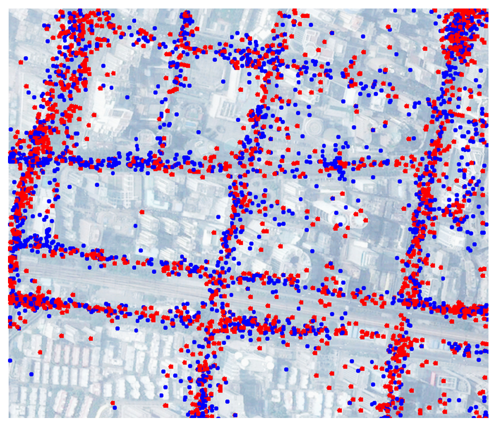

Fig. 39. Distribution of Served Passengers

Figure 38 shows a taxi distribution of an area about 1 square kilometer (GPS Coordinates $XXX.538$-$XXX.547 \times XXX.108$-$XXX.117$) based on a $10s$ uploading window in the rush hour of one day, *i.e.*, 5PM. Red points indicate the taxis with passengers, and blue points indicate the taxis without passengers. Figure 39 shows a served passenger distribution in the same area as in Figure 38 in a two hour 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.

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 discovery in this taxi network. With a total duty cycle $\frac{4}{30}$, we compare four schemes Disco, U-connect, WiFlock and Searchlight with and without the assistance of *Acc*. This duty cycle rate is decided by the fact that a taxi has to communicate with base stations about accounting 26s per 30s. The rest of the time can be used for peer to peer neighbor discovery based on the ad hoc mode. We assume that the taxi is equipped with a radio that has a large communication radius and a communication range of 3 km is used. A smaller communication radius (*e.g.*, $100m$ in a WiFi interface) does not allow our system to fully exploit the quick discovery scheme in a taxi network. This is because an $100m$ communication radius is too small for vehicular networks where even the length of a static taxi is about $5m$. Therefore, a navigation based on such a small communication radius will lead to an extremely low density of taxis, and may not have any obvious performance difference under various discovery schemes.

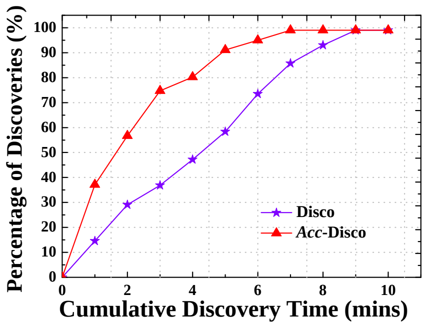

Fig. 40. Disco Latency

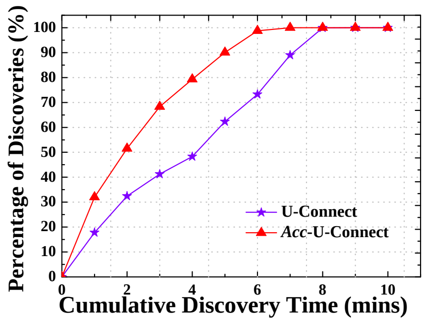

Fig. 41. U-Connect Latency

From Figure 40, 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 22.2% 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 3 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 41, 42 and 43, we plot the same sets of curves for U-Connect, WiFlock and Searchlight. In Figures 41, 42 and 43, we also observe similar performance trends between *Acc*-assisted versions and original versions. For example, in Figure 41, 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 Figures 42 and 43, we still observe a performance gain between *Acc*-WiFlock and WiFlock as well as *Acc*-Searchlight and Searchlight.

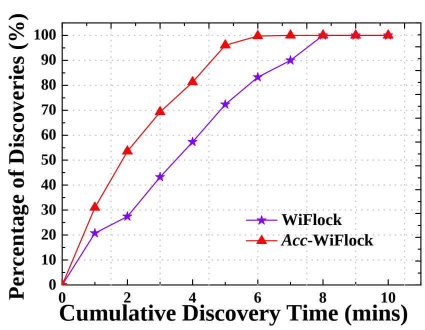

Fig. 42. WiFlock Latency

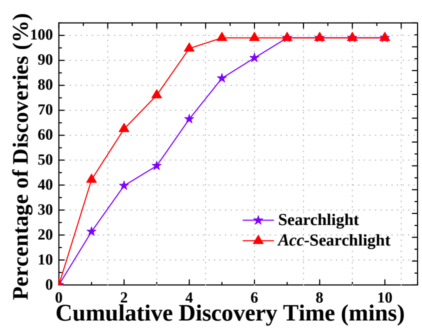

Fig. 43. Searchligh Latency

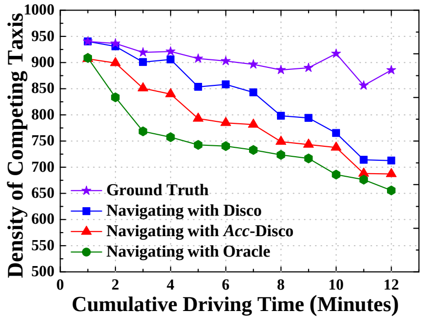

Fig. 44. Density in One Smart Taxi

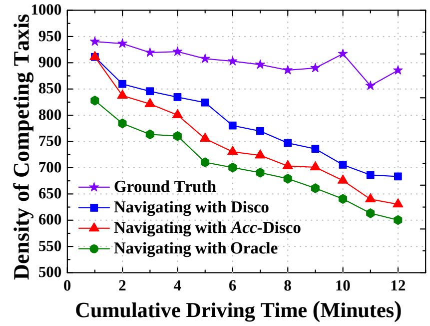

Fig. 45. Density in 10% Smart Taxis

From the results in Figures 40, 41, 42 and 43, we conclude that *Acc* can accelerate various discovery schemes in this taxi network, 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 of $\frac{4}{30}$, we compare 3 navigating results based on different discovery results of discovery schemes.
(i) Navigating with Disco: navigating taxis with the results of Disco;
(ii) Navigating with *Acc*-Disco: navigating taxis with the results of *Acc*-Disco;
(iii) Navigating with Oracle: navigating taxis with the results of an Oracle discovery scheme where a taxi can *instantly* know these two distributions without delay.

Under all navigations, a taxi has the same preferable directions for fewer competing taxis, more served passengers or a ratio between them. But since the employed discovery schemes are different, a navigation with a faster discovery scheme may achieve better performance. The performance is characterized by three metrics: competing taxis density, served passengers density, and a ratio between them. 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. 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 $3km$ communication radius, in Figure 44 and Figure 45, respectively.

Only one smart taxi: We show the situation where one smart driver uses our app to find the optimal route. In Figure 44, 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 increases or decreases, 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

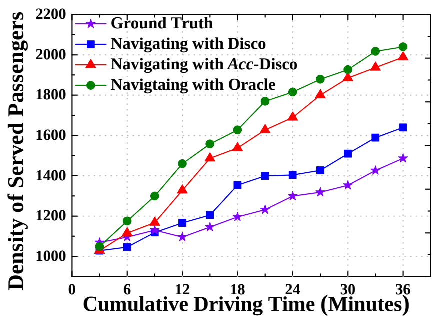

Fig. 46. Density of Passengers

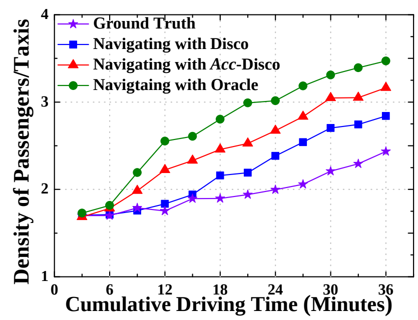

Fig. 47. Density of Passengers/Taxis

fewer competing taxis, so after 3 minutes the density of competing taxis within its range drops. Compared to Ground Truth, after 12 minutes, under Disco the density decreases about 16%, while under *Acc*-Disco the density decreases about 25%, whereas Oracle outperforms Disco and *Acc*-Disco in all the cumulative driving times with a maximal performance gain of 5.1% and 10.1%, 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 the 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:** We show the situation where the taxi drivers from one company (accounted for 10% of taxicabs) use our app to find the optimal route. Figure 45 plots results of 10% of total taxis using our application. Compared to the results in Figure 44, all strategies have better performance, except for Ground Truth, which remains the same. This is because the more the taxis use our applications, the more the taxis will select the direction with lower taxi density, which in turn will achieve a more uniform taxis distribution. Comparing Figure 44 to Figure 45, we note that the performance gain between navigating with *Acc*-Disco and Disco increases from 25% to 29%, 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 46. 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 38 and 39, the density of served passengers is denser than densities of competing taxis, so we use a $0.5km$ radius to compute the density. Figure 46 plots the comparisons of cumulative densities of the served passengers. With an increase in the 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. 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 10% 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 21% and 35%, 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.

(c) Density of Served Passengers and Competing Taxis

Built upon two previous subsections, we investigate the effectiveness of *Acc* to assist navigating schemes lead taxis to a direction with more served passengers and fewer competing taxis at the same time. Thus, we use a ratio equal to the number of served passengers and competing taxis as a new metric for the navigation, instead of considering served passenger and competing taxis, separately. In Figure 47, with an increase in the cumulative driving time, the ratio between the density of passengers and the density of competing taxis in a taxi's neighborhood also generally increases for all the schemes. Similar to Figure 46, we find that compared to Disco and Ground Truth, *Acc*-Disco or Oracle always more quickly navigates the taxicabs to a direction with both more passengers and fewer competing taxis. The performance gains among all four schemes are similar to those in Figure 47.

9. CONCLUSION

In this work, we analyze, design, implement and evaluate *Acc*, an augmenting layer for the acceleration of neighbor discovery in existing deterministic discovery schemes. Our technical endeavors provide a few valuable insights, which are hoped to be useful to realize *Acc* based on-demand neighbor discovery applications in various domains. Specifically, (i) we found that known neighbors can help a device learn unknown neighbors indirectly, which is the key insight about our on-demand discovery; (ii) we designed *Acc* as an independent middleware in the networking architecture, without merging it into existing neighbor discovery protocols, which makes *Acc* a transparent accelerator without the dependence on the above applications or the below neighbor discovery protocols; (iii) we characterized a slot with a novel concept called temporal-spatial coverage to indicate the utility of a discovery device to wake up in the slot, and this characterization is fully distributed and values the slots with neighbors having higher temporal diversity and larger spatial similarity; (iv) we designed a real-world taxi application where *Acc* is used to accelerate the process of taxicab drivers finding efficient routes, which serves as a real-world example to justify the motivation behind the design of *Acc*.

REFERENCES

Denizhan N. Alparslan and Khosrow Sohraby. 2007. Two-dimensional Modeling and Analysis of Generalized Random Mobility Models for Wireless Ad Hoc Networks. *IEEE/ACM Trans. Netw.* 15, 3 (June 2007), 616–629. DOI:http://dx.doi.org/10.1109/TNET.2007.893873

Mehedi Bakht, Matt Trower, and Robin Hilary Kravets. 2012. Searchlight: Won'T You Be My Neighbor?. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom '12)*. ACM, New York, NY, USA, 185–196. DOI:http://dx.doi.org/10.1145/2348543.2348568

James Biagioni, Tomas Gerlich, Timothy Merrifield, and Jakob Eriksson. 2011. EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. ACM, New York, NY, USA, 68–81. DOI:http://dx.doi.org/10.1145/2070942.2070950

Prabal Dutta, Paul M. Aoki, Neil Kumar, Alan Mainwaring, Chris Myers, Wesley Willett, and Allison Woodruff. 2009. Common Sense: Participatory Urban Sensing Using a Network of Handheld Air Quality Monitors. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*. ACM, New York, NY, USA, 349–350. DOI:http://dx.doi.org/10.1145/1644038.1644095

Prabal Dutta and David Culler. 2008. Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*. ACM, New York, NY, USA, 71–84. DOI:http://dx.doi.org/10.1145/1460412.1460420

Prabal Dutta and Lakshminarayanan Subramanian. 2010. Human-Enabled Microscopic Environmental Mobile Sensing and Feedback.. In *AAAI Spring Symposium: Artificial Intelligence for Development*. AAAI. http://dblp.uni-trier.de/db/conf/aaaiss/aaaiss2010-1.html

Jeremy Elson and Kay Römer. 2003. Wireless Sensor Networks: A New Regime for Time Synchronization. *SIGCOMM Comput. Commun. Rev.* 33, 1 (Jan. 2003), 149–154. DOI:http://dx.doi.org/10.1145/774763.774787

Facebook. 2013. Facebook Places. (2013). http://www.facebook.com/about/location

Foursquare. 2013. Foursquare. (2013). http://www.foursquare.com

Raghu K. Ganti, Fan Ye, and Hui Lei. 2011. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine* 49, 11 (2011), 32–39.

Google. 2013. Google Latitude. (2013). http://www.google.com/latitude

Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. 2005. CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In *3rd ACM Conference on Embedded Networked Sensor Systems (SenSys '05)*.

Hyewon Jun, Mostafa H. Ammar, Mark D. Corner, and Ellen W. Zegura. 2006. Hierarchical Power Management in Disruption Tolerant Networks with Traffic-aware Optimization. In *Proceedings of the 2006 SIGCOMM Workshop on Challenged Networks (CHANTS '06)*. ACM, New York, NY, USA, 245–252. DOI:http://dx.doi.org/10.1145/1162654.1162662

Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan (Raj) Rajkumar. 2010. U-connect: A Low-latency Energy-efficient Asynchronous Neighbor Discovery Protocol. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '10)*. ACM, New York, NY, USA, 350–361. DOI:http://dx.doi.org/10.1145/1791212.1791253

Shouwen Lai, B. Ravindran, and Hyeonjoong Cho. 2010. Heterogenous Quorum-Based Wake-Up Scheduling in Wireless Sensor Networks. *IEEE Trans. Comput.* 59, 11 (2010), 1562–1575.

Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. 2010. A survey of mobile phone sensing. *IEEE Communications Magazine* 48, 9 (2010), 140–150.

Hengchang Liu, Jingyuan Li, Zhiheng Xie, Shan Lin, Kamin Whitehouse, John A. Stankovic, and David Siu. 2010. Automatic and Robust Breadcrumb System Deployment for Indoor Firefighter Applications. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM, New York, NY, USA, 21–34. DOI:http://dx.doi.org/10.1145/1814433.1814438

Ting Liu, Christopher M. Sadler, Pei Zhang, and Margaret Martonosi. 2004. Implementing Software on Resource-constrained Mobile Sensors: Experiences with Impala and ZebraNet. In *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*. ACM, New York, NY, USA, 256–269. DOI:http://dx.doi.org/10.1145/990064.990095

Michael J. McGlynn and Steven A. Borbash. 2001. Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks. In *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing (MobiHoc '01)*. ACM, New York, NY, USA, 137–145. DOI:http://dx.doi.org/10.1145/501431.501435

Emiliano Miluzzo, Michela Papandrea, Nicholas D. Lane, Andy M. Sarroff, Silvia Giordano, and Andrew T. Campbell. 2011. Tapping into the Vibe of the City Using VibN, a Continuous Sensing Application for Smartphones. In *Proceedings of 1st International Symposium on From Digital Footprints to Social and Community Intelligence (SCI '11)*. ACM, New York, NY, USA, 13–18. DOI:http://dx.doi.org/10.1145/2030066.2030071

M. Mitzenmacher and U. Upfal. 2007. Probabilitty and Computing.

Nintendo. 2012. Nintendo 3ds - streetpass. (2012). http://www.nintendo.com/3ds/hardware

H. L. Ivan Niven and Herbert S. Zuckerman. 1991. An Introduction to the Theory of Numbers.

Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. 2010. Energy-efficient Rate-adaptive GPS-based Positioning for Smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM, New York, NY, USA, 299–314. DOI:http://dx.doi.org/10.1145/1814433.1814463

Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. 2009. MobiClique: Middleware for Mobile Social Networking. In *Proceedings of the 2Nd ACM Workshop on Online Social Networks (WOSN '09)*. ACM, New York, NY, USA, 49–54. DOI:http://dx.doi.org/10.1145/1592665.1592678

Aveek Purohit, Bodhi Priyantha, and Jie Liu. 2011. WiFlock: Collaborative group discovery and maintenance in mobile sensor networks. In *IPSN*. 37–48.

Jasmin Sasin. 2012. Shenzhen Ranks Fifth in the World in Terms of Population Density. (2012). http://www.shenzhen-standard.com

Softonic. 2012. Bluehoo. (2012). http://www.bluehoo.com

Sony. 2013. PlayStation. (2013). http://us.playstation.com/psvita

Synerge. 2013. Who' s near me. (2013). http://www.synergetechsolutions.com/whos-nearme.aspx

Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. 2010. Cooperative Transit Tracking Using Smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*. ACM, New York, NY, USA, 85–98. DOI:http://dx.doi.org/10.1145/1869983.1869993

Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Yueng Hsieh. 2002. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. In *The Conference on Computer Communications (INFOCOM '02)*.

Wikipedia. 2013. Location Based Game. (2013). http://en.wikipedia.org/wiki/Location-based_game

Tingxin Yan, Vikas Kumar, and Deepak Ganesan. 2010. CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 77–90.

Tingxin Yan, Matt Marzilli, Ryan Holmes, Deepak Ganesan, and Mark Corner. 2009. mCrowd: A Platform for Mobile Crowdsourcing. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*. ACM, New York, NY, USA, 347–348. DOI:http://dx.doi.org/10.1145/1644038.1644094

Desheng Zhang and Tian He. 2012. pCruise: Reducing Cruising Miles for Taxicab Networks. In *RTSS*. 85–94.

Desheng Zhang, Tian He, Yunhuai Liu, Yu Gu, Fan Ye, Raghu K. Ganti, and Hui Lei. 2012. Acc: Generic On-demand Accelerations for Neighbor Discovery in Mobile Applications. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*. ACM, New York, NY, USA, 169–182. DOI:http://dx.doi.org/10.1145/2426656.2426674

Rong Zheng, Jennifer C. Hou, and Lui Sha. 2003. Asynchronous Wakeup for Ad Hoc Networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing (MobiHoc '03)*. ACM, New York, NY, USA, 35–45. DOI:http://dx.doi.org/10.1145/778415.778420

APPENDIX: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. (i) In slot t_0, a device S has already discovered a portion of its neighbors in $n_{t_0}^{(S,S)}$. (ii) 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 $N^{(S,S)}$. (iii) All the discovered neighbors are uniformly distributed in $N^{(S,S)}$. (iv) To minimize the effect of duty cycles, duty cycle patterns for different devices are the same.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.□