Analysis and Design of Low-Duty Protocol for Smartphone Neighbor Discovery

Xiangfa Guo
National University of Singapore
xiangfa@comp.nus.edu.sg

Binbin Chen
Advanced Digital Sciences Center
binbin.chen@adsc.com.sg

Mun Choon Chan
National University of Singapore
chanmc@comp.nus.edu.sg

***Abstract*—An effective neighbor discovery service on smartphones is required for many emerging applications — from proximity-based interactions to opportunistic phone-to-phone collaborations. For a smartphone neighbor discovery service to be usable and attractive, it needs to meet two conflicting goals: 1) phones should discover neighbors fast enough (in seconds), and 2) the service's energy footprint should be negligible so it can be "always on" while incurring little impact on battery life.**

Researchers have developed an impressive collection of neighbor discovery protocols to meet these two goals. By putting these protocols into concrete smartphones settings, we identify different key factors that limit their performance. Guided by our analysis, we focus on locally synchronized protocols, where phones use time information from nearby Wi-Fi Access Points (APs) to help neighbor discovery. By overcoming the key challenges for such protocols, especially, the scalability problem under increasing number of APs and neighbors, we design a new protocol, R2, that achieves low discovery delay (< 30 seconds for at least 80% of all connections) with a low duty cycle (1%).

I. Introduction

Smartphones are perhaps the most ubiquitous mobile computing platforms today. A key aspect of mobile computing is to exploit the proximity-based peer-to-peer communications and collaboration, e.g., via Wi-Fi Direct [1] or Bluetooth. If phones can discover neighboring phones and other wireless devices (e.g., wearable devices or smart sensors embedded in the environment), a wide array of applications become possible, including proximity based mobile apps (e.g., Badoo[1] and Skout[2]) and opportunistic collaboration (e.g., information gathering/sharing [2], [3], traffic forwarding [4], [5], participatory sensing [6]).

Currently, many implementations of these types of applications use a centralized communication (i.e., centralized client-server) approach to discover neighbors, in which communications always go through a server accessible via the Internet. The reliance on infrastructure support imposes constraints on the client-server approach. First, Internet access is always needed. Such a requirement can be expensive to meet if cellular communication is used and user is roaming or may not be possible if Wi-Fi network is the only option but there is no open/free Wi-Fi networks available. Second, the use of cellular network for low-bit-rate location update and neighbor notification is also energy inefficient. Last but not least, making the user location information available at third-party servers imposes significant risk to user privacy. In this work, we adopt an approach based on direct wireless communication. This approach addresses the above mentioned drawbacks as there is no need for Internet access and all communications and information are kept local. Direct communication is also faster and more power efficient.

Researchers have developed an impressive collection of neighbor discovery protocols [7]–[14], and many of these protocols are designed for wireless sensor networks with strict energy constraints. The smartphone neighbor discovery problem can be as challenging as (if not more challenging than) the well investigated wireless sensor network neighbor discovery problem due to the following two reasons.

First, smartphones need to discover their neighbors with short delay (e.g., in seconds). This is because in smartphone settings a large fraction of contacts are short (shorter than 30s) and the ability to utilize such short contacts can increase the number of reachable unique devices significantly. A study of three different mobility traces (Section II) shows that up to 70% of the contacts are shorter than $30s$ and up to 80% more unique devices can be reached if these short contacts are utilized. Second, smartphone neighbor discovery is by itself not a standalone application but an underlying service to support proximity-based applications. As such, it needs to be "always on" (independent from whether there is any neighbor or whether the phone screen is ON) while consuming as little energy as possible. A key technique for achieving energy efficiency is duty cycling, in which devices sleep most of the time and wakeup infrequently to discover each other. While duty cycling saves energy, it increases the discovery delay and decreases the discovery ratio. Existing protocols often evaluate their discovery delay performance in the range of 1% to 10% duty cycle. A close-up examination of smartphone power consumption profile (Section II) reveals that for Wi-Fi-based neighbor discovery, 1% should rather be treated as an upper limit instead of a lower limit.

Our results. In the search for practical smartphone neighbor discovery solutions, we first conduct a thorough review of the design space (asynchronous vs. synchronous, and for the latter, globally synchronized vs. locally synchronized) of neighbor discovery protocols. Somewhat surprisingly, all protocols we analyzed have the same asymptotic delay scaling properties

[1]https://play.google.com/store/apps/details?id=com.badoo.mobile
[2]https://play.google.com/store/apps/details?id=com.skout.android

with respect to the decrease in power budgets. The key for their performance turns out to be the different parameters that determine the hidden constant factor. This observation helps identify locally synchronized protocols as the most promising for achieving both negligible energy footprint and low discovery delay. The infrastructures are widely deployed, e.g., the Wi-Fi Access Point (AP) and Zigbee static motes for docking application [15].In such protocols, phones can sync with the time information transmitted by the infrastructure. Key advantages of this approach are the following. First, since the management frames that are being utilized are broadcasted by APs periodically or on demand by default, no change is required on existing APs. Second, a phone can listen to these periodic beacons through overhearing and trigger the broadcast of AP beacon messages without being associated with an AP. Hence, neither Wi-Fi credential nor Internet access is required. Third, locally synchronized protocols can incur significantly smaller ($< 0.1\%$) synchronization overhead than globally synchronized protocols.

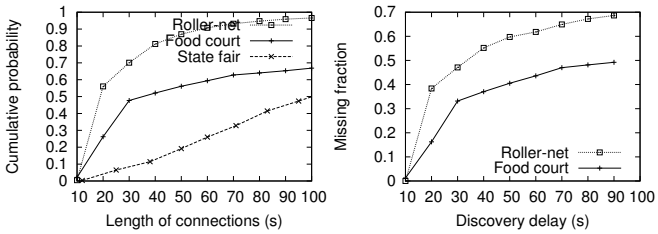
However, locally synchronized protocols present their own challenges. Besides the need to obtain accurate local time references from in-situ Wi-Fi networks even with phone mobility, one fundamental challenge is to deal with "too many neighbors", since phones often cluster together in urban settings (e.g., classroom, food court, or bus terminal). With a large number of neighbors, the synchronous nature of locally synchronized protocols causes excessive contention, which increases the (re)discovery delay linearly or even super linearly. Another immediate concern is the "too many APs" problem, i.e., can two neighboring phones still efficiently select the same time reference, when each phone overhears dozens of APs?

We design R2, a locally synchronized neighbor discovery protocol that overcomes the above discussed challenges. For a range of different urban environments, R2 provides low discovery delay and consumes low power at the same time. R2's Rendezvous-Recon mechanism decouples the discovery delay from neighborhood size, hence addresses the "too many neighbors" problem. Further, R2 solves the "too many APs" problem using a simple minimum-κ strategy with κ as small as 3. Here each phone simply uses a global hashing function to hash the MAC address of each AP it can hear. Then it syncs with the κ APs with the minimum hash values.

Roadmap. Section II discusses the concrete challenges of smartphone neighbor discovery. Section III reviews major existing protocols and identifies the key parameters for different categories of protocols. Section IV presents R2, a low-delay low-duty-cycle locally synchronized neighbor discovery protocol. Section V evaluates R2's performance against the state-of-the-art. After a brief discussion on related work in Section VI, we conclude the paper in Section VII.

II. Motivation

We first investigated contact lengths and power budget for neighbor discovery service, and then motivate the concrete settings for smartphone neighbor discovery protocols.



(a) (b)

Fig. 1. (a) Distribution of contact lengths. (b) Fraction of unique devices (i.e., repeated encounters between the same pair of devices are only counted once) that would be missed under increasing discovery delay.

Name	Scenario
Roller-net	62 internal and 1039 external nodes, in a sports event
Food court	2000 nodes, foodcourt, area of $1500m^2$, 3 hours
State fair	19 nodes, North Carolina state fair, 3 hours

TABLE I
THE TRACES USED FOR EVALUATION

Discovery delay: How short it needs to be? To get an intuitive understanding of the delay requirement for smartphone neighbor discovery, lets consider the scenario where two pedestrians walk in opposite directions towards each other at a normal walking speed of $1m/s$. Suppose their phone-to-phone transmission range is $30m$. Hence, the contact for their phones can last only $30s$. To gain a general contact duration statistics, we analyze three different mobility traces as listed in the Table I.

The roller-net[3] trace contains contact information on 62 users interacting in an outdoor sports event. The state fair trace[4] has 19 nodes with location information. We further assume that the transmission range is 50m. The food court trace records smartphones' presence in a food court by capturing Wi-Fi packets from the phones. We assume that a smartphone is in the food court if a beacon containing its MAC address can be heard and any 2 phones in the food court are within wireless communication range of each other.

In the analysis, we do not consider contacts shorter than 10s. Figure 1(a) shows the CDF of the contact duration for different mobility traces. The result clearly shows that a significant proportion of the contacts are short. In particular, about 70% of the contacts for the roller-net trace last between 10s and 30s. For the two traces (roller-net and food court) with a large number of nodes, Figure 1(b) shows the missing ratio of unique discoverable devices under varying discovery delay. This further illustrates the importance of discovering short contacts: if the discovery delay increases to $30s$, around 30% to 45% of unique devices would be missed. This translates to 40% to 80% more discoverable unique devices if phones can discover short contacts ($10s$ - $30s$) in addition to the longer ones ($>30s$).

Duty cycle: How low is low enough? Existing efforts [7]–[14] often evaluate their protocols with duty cycles in the range between 1% and 10%. While evaluating these values, it is

[3]http://crawdad.cs.dartmouth.edu/upmc/rollernet
[4]http://crawdad.cs.dartmouth.edu/ncsu/mobilitymodels

	Definition	Example values
D	Worst-case discovery delay	Desirably $\leq 30s$
f	(Generalized) duty cycle	Desirably $\leq 1\%$ for Wi-Fi
P	Active radio power	Around $0.5W$ for Wi-Fi
T	Minimum length of wakeup slot	$20ms$ for Wi-Fi
Additional parameters for synchronous protocols		
x	Maximum clock drift rate	$\pm 20ppm$
Γ	Energy consumed by one synchronization operation	NTP over cellular: $\Gamma_g > 28J$ Local Wi-Fi AP: $\Gamma_l = 0.01J$
Y	Maximum synchronization error	NTP over cellular: $Y_g = 20ms$ Local Wi-Fi AP: $Y_l = 2ms$
κ	# of local reference clocks	3

TABLE II
NOTATIONS USED IN THIS PAPER.

important to note that as an underlying service, a neighbor discovery protocol has to be run in the background at *all* time, regardless of whether a phone has phone neighbors around or whether it is in active use. Hence, even a seemingly low duty cycle can convert to substantial energy consumption. As an illustrating example, consider a phone with a $6Wh$ battery[5] and a recharging cycle of 48 hours. As reported in a recent study [16] and confirmed by our own measurements (see more details in Section V-B), the Wi-Fi interface on a smartphone typically consumes around $0.5W$ of power when being used. With the neighbor discovery service over Wi-Fi being operated at a duty cycle of 10%, this service actually consumes 40% (i.e., $0.5W \times 48h \times 10\%/6Wh$) of the total battery capacity, which is clearly unacceptable. In fact, even when operating at a seemingly low duty cycle of 2%, this service's energy footprint is still 8%, which is likely noticeable to the users. Given that neighbor discovery service by itself does not provide direct value to the smartphone users, any visible extra power consumption can discourage users from running such a service. We argue that for Wi-Fi based neighbor discovery, a lower duty cycle (e.g., 1%) should be treated as an important design constraint.

III. Neighbor Discovery: An Analytical Review

This section reviews the major neighbor discovery protocols in the literature. We categorize existing protocols into two groups: 1) *asynchronous protocols* that allow devices to have arbitrary clock offsets, and 2) *synchronous protocols* that synchronize nodes according to common time reference(s).

Table II summarizes our key notations. We denote the minimum length of a wakeup slot by T. For power consumption, while asynchronous protocols only use energy in neighbor searching, synchronous protocols require extra energy for synchronization. To provide a fair comparison between different protocols, we generalize the definition of standard duty cycle to include extra overheads if any. We denote this *generalized duty cycle* by f. For example, if the active power of a cellular interface (e.g., 3G) is two times of Wi-Fi interface's active power, the use of a cellular interface for 1% of time for synchronization translates to an extra 2% of duty cycle on Wi-Fi interface. For performance metric, we denote a protocol's worst-case discovery delay by D. As in previous work, there are two implicit assumptions for reasoning about D: 1) the contact duration is long enough, and 2) all wireless transmissions succeed. For synchronous protocols, we further assume that neighbors are synchronized to a common time reference.

[5]As a reference, the battery capacity of iPhone 5 is 5.45Wh.

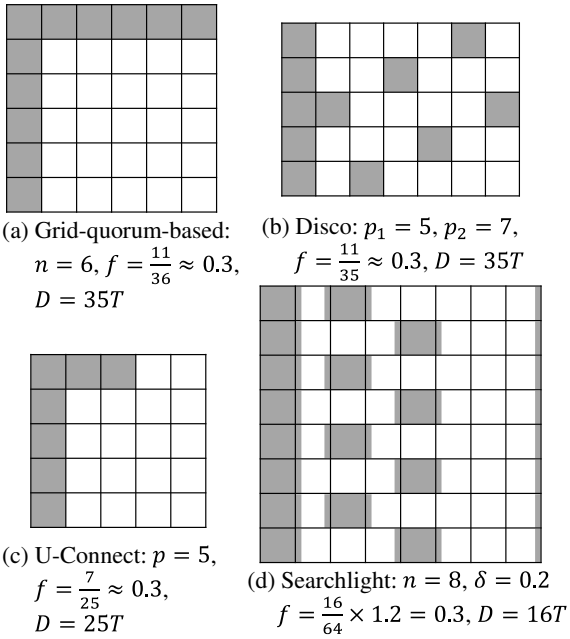


Fig. 2. Wakeup slots (shaded) of four asynchronous deterministic protocols.

A. Asynchronous Protocols

One of the earliest neighbor discovery protocols is *Birthday protocols* [7] proposed by McGlynn and Borbash for static ad hoc wireless sensor networks. In these *asynchronous probabilistic protocols*, each node independently and randomly decides whether to transmit, listen or sleep at each time slot. To improve the discovery performance, one can choose the probabilities based on neighborhood sizes and power budgets. However, Birthday protocols' probabilistic nature results in unbounded worst-case discovery delay guarantee, i.e., $D = \infty$. This has motivated the research on *asynchronous deterministic protocols* to provide worst-case discovery delay guarantees.

Tseng et al. [8] proposed the *quorum-based protocol*. A quorum system is a collection of sets where any two sets have a non-empty intersection. Inspired by this, the proposed protocol lets each node divide time into a sequence of periods, each consisting of n^2 time slots. All nodes use the same n. Consider one period as an $n \times n$ array. Each node selects one row and one column to wakeup, making the duty cycle $f = (2n - 1)/n^2 \approx \frac{2}{n}$. Choosing the wakeup slots in this way ensures that any two nodes have at least 2 common wakeup slots every period of time, regardless of their time offsets. Hence, the worst-case discovery delay is $D = (n^2 - 1)T \approx \frac{4T}{f^2}$. Jiang et al. [9] further proved that no quorum-based protocol can achieve $D < \frac{T}{f^2}$. They also proposed protocol variants that achieve constant-factor reduction of D.

Dutta and Culler [10] proposed the *Disco protocol*, which is inspired by the Chinese Remainder Theorem. In Disco, each node selects two different primes (p_1 and p_2) and wakes up

when its local time is a multiple of either p_1 or p_2. This makes the duty cycle $f = \frac{p_1+p_2}{p_1 p_2}$. If all nodes use the same pair of primes p_1 and p_2, the worst-case delay is $D = p_1 p_2$. Since $f = \frac{p_1+p_2}{p_1 p_2} > \frac{2}{\sqrt{p_1 p_2}}$, we have $D > \frac{4T}{f^2}$. If p_1 and p_2 are within a constant ratio, we still have $D = \theta(\frac{T}{f^2})$. One desirable property of Disco is that each phone can freely choose its own pair of primes according to its power budget.

Kandhalu et al. [11] proposed the *U-Connect protocol*, which combines the co-prime property with the quorum-based protocol [8]. Different from [8], U-Connect allows each node to choose its own period length, as long as the chosen length equals to $p \times p$ for some prime p. Furthermore, U-Connect cuts the wakeup slots from one column plus one row (as in [8]) down to one column plus roughly half of a row. With this, U-Connect reduces the duty cycle to $f = \frac{p+\lceil \frac{p+1}{2} \rceil}{p^2} \approx \frac{1.5}{p}$ (with a saving of around 25%), while still providing similar worst-case delay guarantee as [8].

Bakht et al. [12] proposed the *Searchlight protocol* to further improve over U-Connect. In Searchlight, while a node still uses an $n \times n$ array of slots as a period and wakes up in one column, it does not restrict the rest of its wakeup slots to a single row. Instead, a node chooses one *probe slot* every row and *systematically* changes the probe slot position across rows. For Searchlight, its duty cycle $f = \frac{2n}{n^2} = \frac{2}{n}$ and worst-case discovery delay $D = n\lfloor \frac{n}{2} \rfloor T \approx \frac{2T}{f^2}$. To further reduce D, Searchlight makes each wakeup duration slightly longer than one slot (by a fraction of δ), so that the systematic probing can skip odd positions of each row and probe faster. This roughly halves the worst-case discovery delay to $D = n\lceil \frac{\lfloor \frac{n}{2} \rfloor}{2} \rceil T \approx \frac{T}{f^2}$.

More recently, Wang et al. proposed the BlindDate protocol [17], which further improves the worst case delay of SearchLight by 10%. However, the asymptotic form of the delay performance remains the same. Figure 2 illustrates the wakeup slot schedule for these four asynchronous deterministic protocols. Note that their worst-case discovery delay guarantees are all asymptotically $\theta(\frac{T}{f^2})$.

We also study the neighbor discovery methods used in the Bluetooth technology, based on the protocol description of its Find Me Profile [18]. The protocol works as follows: 1) For two Bluetooth devices to discover each other, they need to assume two different roles: One acts as an advertiser, who broadcasts packets with a fixed advertising interval T_a. The other acts as a scanner, who wakes up every scan interval T_s, and each time it wakes up for a duration of a scan window d_s to listens for an advertiser's packets. This neighbor discovery protocol is hence asynchronous in nature. 2) The Find Me Profile employs a two phase design, as shown in Table III. For the first 30 seconds after a user actively presses some button to initiate the neighbor discovery process, since the scan window d_s is no smaller than than the advertising interval T_a, the discovery always succeeds if there is no packet loss. However, this comes at the cost of high energy consumption for these 30 seconds. For the scanner, if $T_s = 30ms$, it needs to be always awake. Even for $T_s = 60ms$, it needs to be awake 50% of the time. For the advertiser, if it each time needs $10ms$ to send the packets, it also needs to be awake in around 33% to 50% of the time. After the first 30 seconds, to save energy, the design significantly lowers the duty cycle, by letting both the advertiser and the scanner wake up much less often. For the scanner, it wakes up for $d_s = 11.25ms$ every $T_s = 1.28s$ or $2.56s$, which translates to less than 1% of duty cycle. For the advertiser, if it wakes up for $10ms$ every time, an advertising interval T_a of 1s to 2.5s also translates to no more than 1% of duty cycle.

Phase	T_a	T_s	d_s
First 30s	20ms - 30ms	30ms-60ms	30ms
After 30s	1s - 2.5s	1.28s or 2.56s	11.25ms

TABLE III
PARAMETERS USED IN BLUETOOTH FIND ME PROFILE.

For the performance of Bluetooth neighbor discovery under this low duty cycle mode, an advertiser and a scanner may always miss each other under certain T_a and T_s settings (e.g., $T_a = T_s = 1.28s$). Similar observation is made by [19]. Hence, it is important to choose a T_a and T_s pair that can realign the wakeup windows of an advertiser and a scanner over time even if they miss each other initially. Under such settings, Bluetooth neighbor discoverys behavior becomes similar to that of the DISCO protocol. We perform a simulation study based on the setting of $d_a = 10ms$, $T_a = 1.01s$, $d_s = 11.25ms$, and $T_s = 1.28s$, and we let each node to assume both the advertiser and scanner role (so that any two nodes can discover each other). Under such a setting, we found that the performance of the studied Bluetooth protocol is very close to the DISCO protocol under a similar duty cycle. Hence, due to space limitation, we will not discuss the Bluetooth protocol separately in the following.

B. Synchronous Protocols

Two recent efforts leverage time synchronization to improve smartphone neighbor discovery performance.

Globally synchronized protocol. Li and Sinha [13] proposed the *Recursive Binary Time Partitioning (RBTP) protocol*, which shortened discovery delay by synchronizing phones via global time sources like NTP servers. They showed that small synchronization error (in the order of couples of milliseconds) can be achieved in experiments of 8 phones. The key design of RBTP is to deal with the case when phones have different duty cycle budgets, say f_1 and f_2. By letting a node wakeup at slots that recursively partition a time frame in a binary fashion, RBTP can achieve a worst case delay of $D < 2\frac{T}{\min(f_1,f_2)}$ (assuming no synchronization overhead and time error). Note that RBTP requires all devices to be able to access a global time reference (e.g., NTP servers), which may be infeasible for some devices (e.g., small embedded devices) and undesirable for others (e.g., roaming phones).

In synchronized neighbor discovery protocol, periodic time synchronization is required, as clock drift cannot be perfectly compensated, as discussed in Section V-A. We next analyze the impact of clock drift x, global synchronization overhead Γ_g, and initial synchronization error Y_g on the performance of a globally synchronized protocol like RBTP.

Assume each phone performs a time synchronization operation every period of C_{sync}, where C_{sync} can be tuned to optimize for discovery delay. Immediately after the time synchronization operation, one wakeup slot will be $T + Y_g$ long, where Y_g is the initial error for global synchronization. To cater for the (unknown) clock drift, the wakeup duration should be prolonged linearly as the time from the last synchronization point increases. Toward the end of one syncing period, one wakeup period should last for $T + Y_g + 2xC_{sync}$. The multiplicative factor of 2 in the last term is to cater for the drift in both directions. Hence, on average the wakeup duration is $T + Y_g + xC_{sync}$ long. To provide a delay bound of D, a phone needs to wakeup every interval of D, this leads to a total of $\frac{C_{sync}}{D}$ wakeup slots in a syncing period. Consider the extra energy consumed by a global synchronization operation, Γ_g, as divided by the active power P of selected radio, the total wakeup duration in a C_{sync} period is:

$$\frac{\Gamma_g}{P} + (T + Y_g + xC_{sync})\frac{C_{sync}}{D} = fC_{sync} \quad (1)$$

Let $C_{sync} = \frac{(1+a)\Gamma_g}{Pf}$, where $a > 0$ is the ratio between the energy for neighbor discovery and that for synchronization:

$$\begin{aligned} D &= \Big(T + Y_g + x\frac{(1+a)\Gamma_g}{Pf}\Big)\frac{1+a}{fa} \\ &= \frac{x\Gamma_g}{Pf^2}\Big(a + 2 + \frac{1}{a}\Big) + \frac{T+Y_g}{f}\Big(1 + \frac{1}{a}\Big) \quad (2) \end{aligned}$$

With $a = \sqrt{1 + \frac{Pf(T+Y_g)}{x\Gamma_g}}$, we can minimize D and obtain:

$$\begin{aligned} D &\in \Big(\frac{4x\Gamma_g}{Pf^2} + \frac{T+Y_g}{f}, \frac{4x\Gamma_g}{Pf^2} + \frac{2(T+Y_g)}{f}\Big) \quad (3) \\ &= \theta\Big(\frac{x\Gamma_g}{Pf^2} + \frac{T+Y_g}{f}\Big) \quad (4) \end{aligned}$$

Under decreasing f, the first term in Equation (4) dominates. Somewhat surprisingly, a globally synchronized protocol has the same asymptotic delay performance, i.e., $\theta(\frac{1}{f^2})$, as asynchronous deterministic protocols. Differently, the key parameters are now $\frac{x\Gamma_g}{P}$ instead of T.

Locally Synchronized Protocol. Now we extend the above analysis to a locally synchronized protocol, which uses nearby signaling messages, for example messages broadcast by Wi-Fi AP as local time reference(s). The advantages of such protocols are that since the synchronization can be done based on the timestamps in the AP beacons, there is no need for the phones to have permission to associate with these APs or gain Internet access through them. There is also no need to make any changes to the APs.

A discovery protocol proposed by Camps-Mur and Loureiro [14] called E^2D is based on such a locally synchronized approach. However, the functionality provided by E^2D is different from other neighbor discovery protocols described earlier. The objective of E^2D is information sharing and not pair-wise node discovery. A node belongs only to one cluster and the algorithm aims to increase cluster size rather than pair-wise node discovery. In locally synchronized protocols, two nodes can discover each other only if they share a common time reference, or in E^2D context, be in the same cluster. In practical settings, it is likely that some nodes cannot discover one another because they are in different clusters. In order increase discovery probability, a node has to use multiple (κ) APs as time references.

For the analysis, assume each phone scans for local time references every period of D_0. Once a phone chooses a time reference, it wakes up periodically when the referred time becomes a multiple of D_1. Considering both components, the total worst-case delay becomes $D = D_0 + D_1$.

Suppose a phone scans surrounding APs and gets its reference clocks every C_{sync} period. For D_0, the worst case happens when two phones become neighbors at the moment when one of them just finishes a sync operation, in this case, $D_0 = C_{sync}$. Now we consider D_1. For locally synchronized protocols, a phone uses κ APs as local time references. Denote the overhead for one syncing operation by Γ_l. Similar to the analysis of globally synchronized protocols (see Section III-C), considering the maximum clock drift during a sync cycle, the average wake-up time for neighbor discovery is $T + Y_l + xC_{sync}$, where Y_l is the initial inaccuracy of local synchronization. To achieve the worst case delay of D_1, a phone needs to wakes up around $\frac{C_{sysnc}}{D_1}\kappa$ times in a synchronization cycle. The total wake-up duration in a period is the sum of synchronization wake-up and the neighbor discovery wake-up, as follows:

$$\frac{\Gamma_l}{P} + (T + Y_l + xC_{sync})\frac{C_{sync}}{D_1}\kappa = fC_{sync} \quad (5)$$

Let $C_{sync} = \frac{(1+a)\Gamma_l}{Pf}$, where $a > 0$ is the ratio between the energy used for neighbor discovery wake-up and the synchronization wake-up. Then we have

$$D_1 = \kappa\Big(\frac{x\Gamma_l}{Pf^2}a + \Big(\frac{x\Gamma_l}{Pf^2} + \frac{T+Y_l}{f}\Big)\frac{1}{a} + \frac{2x\Gamma_l}{Pf^2} + \frac{T+Y_l}{f}\Big) \quad (6)$$

Further considering $D_0 = C_{sync} = \frac{(1+a)\Gamma_l}{Pf}$, we have:

$$D = \frac{\kappa x\Gamma_l}{Pf^2}\Big(a + \frac{1}{a} + 2\Big) + \frac{\Gamma_l}{Pf}\Big(1 + a\Big) + \frac{\kappa(T+Y_l)}{f}\Big(1 + \frac{1}{a}\Big) \quad (7)$$

It is easy to see that with $a = \sqrt{1 + \frac{\kappa Pf(T+Y_l) - f\Gamma_l}{\kappa x\Gamma_l + f\Gamma_l}}$, D is minimized and its minimum value falls into the range of:

$$\Big(\frac{4\kappa x\Gamma_l}{Pf^2} + \frac{\Gamma_l + \kappa P(T+Y_l)}{Pf}, \frac{4\kappa x\Gamma_l}{Pf^2} + \frac{2(\Gamma_l + \kappa P(T+Y_l))}{Pf}\Big) \quad (8)$$

Asymptotically, we have:

$$D = \theta\Big(\frac{\kappa x\Gamma_l}{Pf^2} + \frac{\Gamma_l + \kappa P(T+Y_l)}{Pf}\Big) \quad (9)$$

where Γ_l and Y_l are energy consumption and initial error for local synchronization respectively.

With decreasing f, the first term in Equation (9) dominates, and the locally synchronized protocol scales at $\theta(\frac{1}{f^2})$. This is asymptotically the same as all the other protocols.

Category	Representative protocols	Worst-case delay D
Asynchronous probabilistic	Birthday protocols [7]	Unbounded
Asynchronous deterministic	Quorum-based protocols [8], Disco [10], U-Connect [11], Searchlight [12]	$\theta(\frac{T}{f^2})$
Globally synchronized	RBTP [13]	$\theta(\frac{x\Gamma_g}{Pf^2} + \frac{T+Y_g}{f})$
Locally synchronized	E^2D [14]	$\theta(\frac{\kappa x\Gamma_l}{Pf^2} + \frac{\kappa(T+Y_l)}{f})$

TABLE IV
PERFORMANCE OF NEIGHBOR DISCOVERY PROTOCOLS.

	$f = 0.5\%$	$f = 1\%$	$f = 5\%$
T=20ms	800s	200s	8s
T=1s	40,000s	10,000s	400s

TABLE V
WORST-CASE DELAY D OF ASYNCHRONOUS PROTOCOLS

C. Which Protocols To Use?

We now study the performance of different neighbor discovery protocols and their key parameters. For asynchronous protocols, we will simply quote the existing results. For synchronous protocols, since neither RBTP nor E^2D explicitly considers synchronization overhead Γ, initial synchronization error Y, and clock drift x, we perform the required analysis to quantify their performance. Table IV summarizes the results from our analytical review, which shows how the worst-case delay D of different protocols vary with the duty cycle f and other key parameters.

Asynchronous protocols. As reviewed earlier, probabilistic protocols like Birthday protocols do not provide bounded worst-case delay, while all asynchronous deterministic protocols provide the same asymptotic guarantee $D = \theta(\frac{T}{f^2})$. Jiang et al. [9] proved a matching lower bound for generic quorum-based protocols. We now discuss numerical examples based on the latest and most efficient protocol Searchlight [12], for which $D \approx \frac{T}{f^2}$.

The value of T can vary by two orders of magnitude: from more than 1s (e.g., when controlling Wi-Fi from application layer [12]), to dozens of milliseconds (with proper support from the chip and device driver[6]). Table V shows the worst-case delay guarantee D of Searchlight under different duty cycles ($f = 0.5\%$, $f = 1\%$, and $f = 5\%$) and time slot lengths ($T = 1s$, and $T = 20ms$). If the objective is to have $D \leq 30s$, the only admissible configuration in Table V is $f = 5\%$ and $T = 20ms$. If low duty cycle, e.g., $f \leq 1\%$ is needed, one has to further reduce T. A quick calculation shows that this requires $T \leq 3ms$, which is practically challenging if not impossible. In summary, the $\theta(\frac{T}{f^2})$ term fundamentally restricts asynchronous protocols from simultaneously achieving low delay and low duty cycle.

Globally synchronized protocols. We now discuss numerical examples based on RBTP. Note that RBTP's design assumes a common/global time reference, but it is not bound to any specific synchronization method. For synchronization overhead Γ_g, Li and Sinha reported that one instance of NTP synchronization takes on average $18s$ [13]. This high overhead is because RBTP uses NTP, which requires the transmission of multiple (e.g., 10) packets over WAN (e.g., cellular network). We refer to this version of RBTP as RBTP-NTP. Consider that there is an additional long tail phase ($10s$ to $20s$, varied by the carriers) after transmission and cellular interface consumes more than $1W$ of power even in the tail phase [20]. We have $\Gamma_g > 1W \times (18s + 10s) = 28J$ for RBTP-NTP. While this value may vary depending on different settings, we feel that the overhead of a global synchronization method will likely remain high, if the synchronization is done via a cellular network interface. The needed energy can be reduced if one uses WiFi interface instead to obtain the global time reference. However, this will require the phone to have internet access through the WiFi network, which is a strong assumption. One can also use GPS as a global time reference, however, it comes with its own drawback that it will not work well indoor or in areas with high-rise buildings. Our discussion in the following of this paper is based on this assumption. If a protocol can obtain a global clock with lower overhead (e.g., through Wi-Fi network), its performance will also change accordingly.

Due to the noisy round trip time over WAN, the synchronization accuracy is in the order of dozens of ms, e.g., $Y_g = 20ms$ as reported in [13]. Suppose the maximum clock drift rate is $x = 20PPM$. Consider Wi-Fi neighbor discovery with Wi-Fi's active power at $P = 0.5W$, the constant factor before the $\frac{1}{f^2}$ term for RBTP-NTP (see Equation 3) is $\frac{4x\Gamma_g}{P} > \frac{4\times 20\times 10^{-6}\times 28J}{0.5W} \approx 5ms$. Recall that for Searchlight the factor before $\frac{1}{f^2}$ is $T = 20ms$. RBTP-NTP outperforms asynchronous protocols under this specific setting. However, for $f = 1\%$, RBTP-NTP's worst-case delay $D > \frac{5ms}{0.01^2} = 50s$, which is still not quite satisfactory.

If we compare RBTP-NTP with Searchlight over an alternative interface (e.g., ZigBee) that has lower power consumption (e.g., $P = 0.1W$) and shorter wakeup slot length (e.g., $T = 5ms$), RBTP-NTP's factor before the $\frac{1}{f^2}$ term increases to $\frac{4x\Gamma_g}{P} > \frac{4\times 20\times 10^{-6}\times 28J}{0.1W} > 20ms$, while Searchlight's corresponding factor reduces to $T = 5ms$. In such a setting, Searchlight can actually outperform RBTP-NTP. Again, neither can meet $D < 30s$ requirement if $f \leq 1\%$.

Locally synchronized protocols. We observe that a locally synchronized protocol's factor for the $\frac{1}{f^2}$ term can be significantly smaller than all other protocols. In particular, to find the timestamps of nearby APs, a phone needs less than $20ms$ for sending a probe request and receiving probe replies (see our evaluation results in Section V-C). Hence the synchronization overhead can be as low as $\Gamma_l = 0.5W \times 20ms = 0.01J$, which is more than three orders of magnitude smaller than the global synchronization overhead. Hence, if κ and other hidden constants are not particularly large, locally synchronized protocols have the potential to outperform both asynchronous protocols and globally synchronized protocols for small f

We illustrate the constant factor for $1/f^2$ of minimum delay by looking at some practical settings. Let $P = 1W$ (for a

[6]http://www.gainspan.com/docs2/GS2011M-PB.pdf

Wi-Fi radio), $\Gamma_l = 0.01J$, $x = 20ppm$, and $\kappa = 3$, the factor before the $1/f^2$ term, i.e., $4\kappa x\Gamma_l/P$ (see Equation (8)), is $4 \times 3 \times 20 \times 10^{-6} \times 0.01J/0.5W = 0.0048$ms, which is significantly smaller than the factor of around 5ms for RBTP-NTP and the factor of around 20ms for asynchronous protocol. Hence, for values of f around 0.1% to 1%, discovery delay in locally synchronized protocol is dominated by the constant terms in $1/f$ rather than $\frac{1}{f^2}$.

This observation indicates while the asymptotic behavior for all the protocols are the same, locally synchronized protocols present a viable option to simultaneously achieve low delay and low duty cycle in practical settings. To the best of our knowledge, the only protocol that takes a locally synchronized approach is the E^2D protocol proposed by Camps-Mur and Loureiro [14]. However, E^2D is designed for information sharing and not for pair-wise node discovery. Hence, it suffices in E^2D for a node to find some neighbors (instead of all possible neighbors). In the next section, we will present our design of a locally synchronized protocol that provides efficient pair-wise neighbor discovery capability.

IV. R2: A New Locally Synchronized Protocol

A. Overview

We design a new locally synchronized neighbor discovery protocol, called $R2$, that can provide low delay and performs well with low duty cycle ($\leq$ 1%). R2 is designed with the following urban setup in mind: There are a large number of WiFi APs that provide dense coverage, but many users may not have the permission for Internet access through any of these WiFi networks. Phones synchronize based on the timestamp (granularity of microsecond) in the beacon messages sent by WiFi APs. When a smartphone transmits a probe request message, WiFi APs in the vicinity reply with probe response messages containing the respective timestamps. A smartphone selects a subset of the APs who have replied as reference APs. Two phones can synchronize if they share at least one common reference AP. Such probe request/reply exchanges do not require any change to the existing Wi-Fi APs and also do not require the phones to have the credentials to associate with the APs. As each phone synchronizes its local clock with an AP in an independent manner, the overhead increases linearly with the number of phones. In addition, as synchronization is performed infrequently, when there is a change in environment or to mitigate effect of clock drift, the overhead is low.

In the rest of this section, we will present the basic operations of R2. After that we highlighting R2's design that handles the following three issues: 1) too many neighbors, 2) too many nearby APs, and 3) mobility.

B. Basic Operations of R2

R2's name comes from its Rendezvous-Reconnaissance design that allows a node to vary its behavior depending on the mode of operation. In general, the Rendezvous slots are used for broadcasting of neighbor information collected, while each Reconnaissance (or Recon) slot is used by a smaller number of nodes for discovery and/or update of their connectivity status.

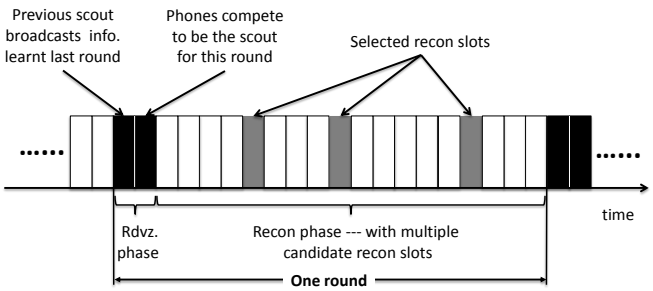


Fig. 3. Illustration of R2's basic operations in rounds.

As illustrated in Figure 3, each round consists of 2 rendezvous (rdv.) slots and one or more recon slots. We call this a *Rendezvous-Recon round*, or a *round* for short. There are two phases in a round, a *rendezvous* phase where new nodes can be discovered based on information collected from the previous round and a *reconnaissance* phase where new and existing nodes announce their presence. $R2$ considers all phones that are synced to the same AP as one cluster. A smartphone can participate in multiple clusters. Since each cluster operates independently and in the same way, we will describe the rendezvous and reconnaissance phases considering one cluster.

Rendezvous: The rendezvous phase occurs at the beginning of a round and serves the purpose of sharing information efficiently. There are only two slots in this phase. In the first slot, a node, called the *scout*, broadcasts the information it has collected in the previous round. All other nodes, including new nodes that just joined, listen to this broadcast message to learn the identities of potential nodes in the neighborhood. In the ideal case (no collision or packet loss and assume a fully connected neighborhood), every node gets an update about every neighbor based on information contained in this message. Specifically, all nodes (including the next scout) know about the number of neighbors so they can coordinate efficiently next round. If no scout has been selected in the previous round, or if the previous scout fails to report the neighborhood size, no information is shared and the next scout will assume a large neighborhood size for the next round. While this estimate can be inaccurate, this will only affect the efficiency of that single round if the next scout can successfully finish that round and provide an accurate estimate for its successor. We will further discuss the robustness of R2 under other non-ideal cases towards the end of this subsection.

In the second slot, all nodes compete to be the scout node for this current round. Each node selects a random backoff interval and transmits a packet containing its identity if it does not hear any other transmission. If a node hears such an announcement packet, the scout node has been selected and it will not transmit any more packet in this slot. If the slot duration is sufficient long, collision is unlikely.

Reconnaissance: In the reconnaissance phase, a node wants to know: 1) whether its neighbors are still in vicinity, and 2) when is the next slot that it can reach a particular neighbor if needed. The goal now is for each phone to minimize the maintenance (or so called rediscover) delay, which measures how long a phone can get an update about the state of a

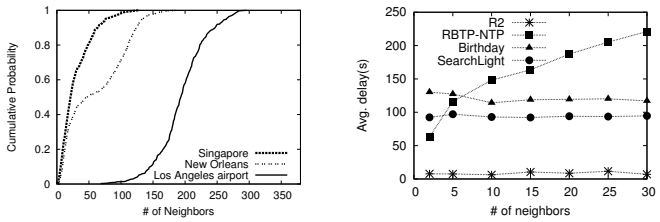
neighboring phone.

In this phase, there are many *recon* slots available. The number of available recon slots depends on the duration of a round (e.g. 2s) and slot length (say 20ms). If all slots are of duration 20ms, in a 2s round, there will be a 40ms rendezvous phase and a 1.96s (98 slots) reconnaissance phase. In the reconnaissance phase, the scout node previously selected listens, while other nodes transmit once in a chosen recon slot. In order to reduce the number of slots the scout node has to listen to, a threshold m is utilized. For a given slot length, m is chosen as the average number of transmissions possible in a slot with minimum collision. The number of recon slots that a scout node listens to is dynamically adjusted according to the number of nodes (N) that the scout node thinks is in the neighborhood and is set to $\lceil \frac{N}{m} \rceil$. This number $\lceil \frac{N}{m} \rceil$ is broadcasted by the scout node when it competes successfully in the rendezvous phase earlier. All other nodes only transmit a message containing its identity once. The exact recon slots used is determined by a pre-determined seed and pseudo number generator. In each of these active recon slots, up to m nodes transmit on the average. At the end of a reconnaissance phase, the scout node will have heard transmission from all nodes in the cluster. This information will be broadcasted by the scout to all nodes in the rendezvous phase of the next round and the process repeats.

Robustness of R2: Considering a single round, the scout selection may fail due to collision (though as described above, R2's design reduces the failure probability by having one dedicated slot for choosing scout). Even when a scout is successfully chosen, the scout may miss some neighbors in the corresponding recon slots due to poor channel condition. The scout may even become totally disconnected from the cluster due to its mobility, hence disappears in the next round's rendezvous slot. Despite of all these failure possibilities in a single round, the robustness of R2 fundamentally lies in its built-in independence between different rounds. In particular, even when the information collection fails for one round. In the next round, a new scout selection will be performed and the process repeats. In other words, the effect of "bad events" does not propagate across different rounds. On the other hand, the observation of bad events is actually exploited by R2 protocol to improve its robustness. In particular, if a scout observes too many nodes in the neighborhood and collision is too high in the recon slots, it can extend the number of recon slots to minimize such bad events. This resilience also applies to cases whereby there is transmission/interference in the recon slots. In such cases, only transmission in that particular slot is affected.

C. Analysis of R2's Performance with Increasing Neighbors

Compared to asynchronous protocols, one fundamental challenge for synchronous protocols is to deal with "too many neighbors". In urban settings, phones often cluster together (e.g., consider a classroom, a food court, or a bus terminal). As shown in Figure 4 (a), in the town area of Singpore and New Orleans, there can be more than 50 neighbors for 20% to 50% of the time. The trace collected in a crowded waiting area of the Los Angeles international airport shows an even higher density, with the number of neighbors ranging from 100 to nearly 300. With an increasing number of neighbors, a synchronous protocol can cause excessive contention, if all neighbors wake up at around the same time and compete to announce their presence. This dramatically increases the discovery delay. This issue cannot be solved by only allowing new nodes to send, since rediscovering (or maintaining) existing neighbors is often as important as (if not more important than) new discovery. For example, in the *flocking use pattern* described by Dutta and Culler [10], a teacher needs to keep tracking a group of children, making sure no one is missing.



(a) (b)

Fig. 4. (a) Distribution of the number of neighbors in urban settings. (b) Performance of different protocols with increasing neighborhood sizes.

Figure 4(b) plots the average discovery delay of different protocols in a clique of nodes with increasing neighborhood sizes. The protocols operate at a duty cycle of 1%. See Section V-D for more details about our evaluation setup. As shown in the figure, asynchronous protocols like Birthday protocol and SearchLight are not much affected by the neighborhood size. However, the discovery delay for RBTP-NTP increases rapidly with the neighborhood size. In fact, for a neighborhood size of $N = 15$, its performance becomes worse than the asynchronous protocols.

Consider a clique of N nodes. On expectation, each node serves as the scout in $\frac{1}{N}$ fraction of rounds, for which it wakes up in $\frac{N}{m}+2$ slots (2 for the rendezvous slot, and $\frac{N}{m}$ for listening to other nodes in the recon slots) each round. For the other $1 - \frac{1}{N}$ fraction of rounds, a node only wakes up in 3 slots (2 rendezvous slot plus 1 recon slot) each round. Hence, for each node, its expected number of wakeup slots in each round is:

$$\frac{1}{N} \times (\frac{N}{m} + 2) + (1 - \frac{1}{N}) \times 3 < 4 \qquad (10)$$

Hence, even with increasing number of neighbors, R2's round length needs to be at most 4 times. Since the length of the Rendezvous-Recon round decides R2's worst-case delay D, R2's delay remains constant despite the growth of N, hence is perfectly scalable with neighborhood size. One can trivially generalize the above reasoning to $\kappa > 1$, by simply considering each of the κ instances independently.

D. Having "Too Many APs"

Another concern of locally synchronized protocols is whether two neighboring phones can effectively select the same time reference, especially in urban areas where each

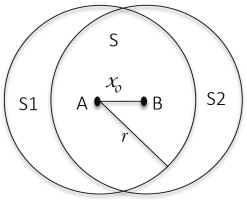
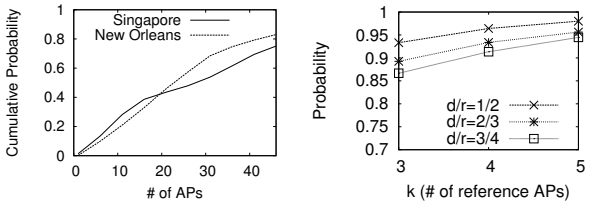


Fig. 5. Areas where reference APs for two neighboring phones can appear.



(a) (b)

Fig. 6. (a) The number of APs a phone can overhear in urban settings. (b) Probability for two phones to select at least one common reference Wi-Fi AP.

phone can hear dozens of APs. Figure 6 (a) plots the distribution of the number of APs a phone can overhear (within a 10 seconds interval) when a user walks around the town areas of two different cities. Each trace lasts for 5 hours and covers an area around $7km^2$. As shown in the figure, for both cities, a phone overhears more than 20 APs for more than half of the time, and overhears more than 50 APs for around 20% of the time.

If a phone uses all nearby APs, κ becomes a large value. Recall that the delay of locally synchronized protocols scales by $\frac{\kappa x \Gamma_l}{Pf^2}$, while for globally synchronized protocol it scales by $\frac{x\Gamma_g}{Pf^2}$. A large κ can quickly offset the gain of locally synchronized protocols.

$R2$ uses a simple *minimum-κ strategy* to deal with this issue. Here each phone uses a global hashing function to hash the MAC address of each AP it can hear. Then it syncs with the κ APs with the minimum hash values. While a similar idea is briefly mentioned in [14], there is no analysis of its effectiveness. In this paper, we use both analysis and trace-driven simulation to show that this simple minimum-κ strategy with κ as small as 3 suffices.

In our analysis, we assume APs and phones are distributed uniformly in the space. We assume a unit-disk communication model, where a phone can communicate with an AP if their distance is shorter than r, and two phones can communicate with each other if their distance is shorter than d. Since APs normally have higher antenna gain than phones, r is usually larger than d. We also assume a dense AP environment, where each phone can hear, i.e., receiving beacons from, at least 3 APs.

We let a phone choose κ APs in the following way: it uses a global hashing function to hash each AP's BSSID (i.e., its MAC address) to a number. Denote this hashed number for the ith AP by h_i. Out of all APs that a phone can overhear, it synchronizes with the κ APs of the minimum h_is. We call this the minimum-κ approach.

Consider two neighboring phones. Since they can hear each other, their distance x is in the range of $[0, d]$. Since they are uniformly distributed in the space, as follows:

$$Pr[x < x_0] = (x_0/d)^2, \text{for } x_0 \in [0, d] \quad (11)$$

Consider two phones A and B that are x_0 distance apart. As illustrated in Figure 5, an AP in the union of the two disks (each with a radius of r) can be heard by at least one phone, and an AP in the overlapped section of the two disks (i.e., the "S" section in the Figure) can be heard by both phones. The area of the "S" section is as follows:

$$S = 2r^2 cos^{-1}(\frac{x_0}{2r}) - \frac{x_0}{2}\sqrt{4r^2 - x_0^2} \geq \pi r^2 - \frac{2}{3}\pi r x_0 \quad (12)$$

For phone A, since it chooses the κ APs using the minimum-κ approach and since APs are uniformly randomly distributed, the probability for the event Φ that none of these κ APs falls in the gray section is

$$Pr[\Phi] = (1 - \frac{S}{\pi r^2})^\kappa \leq (\frac{2x_0}{3r})^\kappa \quad (13)$$

If both phone A and phone B choose at least one AP in the gray section, they must choose at least one common AP, i.e., the one with the smallest hashed value out of all APs in the gray section. Hence, the probability that they don't choose a common AP is equal to the probability that at least one of them do not select any AP in the gray section, which shows in the follow equation:

$$2 \times Pr[\Phi] - Pr[\Phi]^2 \leq 2 \times (\frac{2x_0}{3r})^\kappa - (\frac{2x_0}{3r})^{2\kappa} \quad (14)$$

On integrating this probability over all possible $x_0 \in [0, d]$ according to equation 11, the probability for this case (having no common AP in gray area between two randomly selected neighbor phones) becomes no greater than:

$$\frac{4}{\kappa + 2} \times (\frac{2d}{3r})^\kappa - \frac{1}{\kappa + 1} \times (\frac{2d}{3r})^{2\kappa} \quad (15)$$

We evaluate this probability for different ratios of $\frac{d}{r}$ and for different values of κ, and the results are summarized in Table VI. This results agree with the simulation results using real AP locations, as shown by Figure 6(b). The results show a small value of κ is enough. For example, if $\kappa = 3$, for $\frac{d}{r} = 3/4$, $2/3$, and $1/2$, the probability that two phones have at least one common reference AP by following the minimum-κ approach is 90.4%, 93.2%, and 97.1% respectively. Hence, when one can tolerate a small probability of contact missing rate, $\kappa = 3$ suffices. To further validate our analysis results, we conduct a simulation based on real-world AP locations[7].

	$\kappa = 3$	$\kappa = 4$	$\kappa = 5$
$\frac{d}{r} = 1/2$	97.1%	99.2%	99.8%
$\frac{d}{r} = 2/3$	93.2%	97.4%	99.0%
$\frac{d}{r} = 3/4$	90.4%	96.0%	98.2%

TABLE VI
THE PROBABILITY THAT TWO NEIGHBORING PHONES CHOOSE AT LEAST ONE COMMON REFERENCE AP.

[7]http://www.crawdad.org/dartmouth/wardriving/

As shown in Figure 6(b), the results match our analysis well and $\kappa = 3$ suffices for achieving 90% of probability for two neighboring phones to choose a common AP as a time reference.

Note that to find these 3 APs, R2 does not need to perform frequent scanning over many channels. In practice, some channels are much more likely to be used, for example channel 1, 6, or 11. Hence, if R2 starts to scan on a popular channel and discover 3 or more APs, it has found sufficient APs and no further scanning is needed. Furthermore, scanning only needs to be performed periodically, say, once every 5 seconds to detect change in the environment. Taking both factors into account, R2 has relatively low scanning overhead.

E. Obtaining Local Time References Under Mobility

R2 employs the following strategy to deal with phone mobility. In each round during the reconnaissance phase, a node may randomly select a slot for synchronization purpose. In this synchronization slot, the node sends a probe request and listens for the probe reply messages from the nearby APs. The frequency at which a node performs such synchronization depends on the parameter a, the ratio between the energy for neighbor discovery and for synchronization as mentioned in Section III-B. This synchronization ensures that a node can refresh its list of nearby APs quickly. As a side benefit, a node can now use its list of APs to decide whether it has moved and is a newcomer.

V. EVALUATION

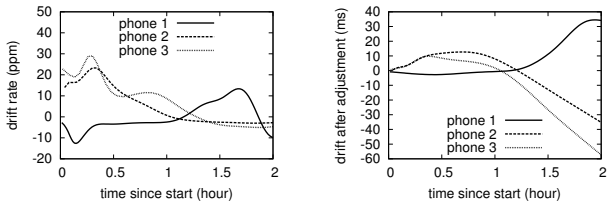
A. Phone Clock Drift Measurement

The performance of synchronized protocols depends on the clock drift rate. Clock drift rate can be affected by different factors, including the quality of the quartz oscillator, temperature, and humidity [21].

We conducted an experiment to measure the clock drift rates on modern smartphones and to understand the effectiveness of a clock drift calibration approach recently proposed in [13]. In our experiment, we distributed one Galaxy S II phone and two Galaxy Nexus phones (A, B) to three volunteers, who carried the phones with them for a few hours. During the experiment, the volunteers moved freely around both outdoor (with relatively high temperature and humidity) and air-conditioned indoor environments. Each phone periodically recorded the time of its local clock and a global reference clock. For a period from time s_r to time e_r (in global reference clock), if a phone's local time changes from s_l to e_l, we denote its clock drift rate r by:

$$r = \frac{(e_l - s_l) - (e_r - s_r)}{e_r - s_r} = \frac{e_l - s_l}{e_r - s_r} - 1 \qquad (16)$$

A positive value of r indicates that the phone's local clock is faster than the reference clock, while a negative r indicates that the phone's clock is slower than the reference clock. We plot the clock drift rate of the three phones in Figure 7(a). The figure shows that different phones have different clock drift rates, and the same phone has different clock drift rates under different environments. For example, the local clock of phone 1 is slower than reference clock in the first hour, but becomes faster than the reference clock in the second hour. The clock drift rates of tested phones vary from around $-15ppm$ to nearly $30ppm$.



(a) (b)

Fig. 7. (a) Clock drift rate. (b) Clock drift after linear adjustment.

To better deal with clock drifting, the original RBTP paper [13] proposes an approach based on linear regression to calibrate the clock drift rate. Following the approach, for each phone, we conduct a linear regression over its first 10 minutes of time readings (note that [13] uses around $20s$ of time readings for calibration). We then use the estimated clock drift rate in this initial calibration period to adjust the phone's time reading for the remaining of the experiment. For example, if the estimated clock drift rate in the initial phase is r, and the start time is t_0 as synchronized to the reference clock, we compute the adjusted local time t' by: $r = (t - t_0)/(t' - t_0) - 1$ Hence,

$$t' = t_0 + \frac{t - t_0}{r + 1} \qquad (17)$$

Figure 7(b) shows the amount of clock drift after such a linear compensation, where the clock drift at the reference time t_r is defined as: $drift = t' - t_r$ A positive value of $drift$ indicates that the phone's clock is ahead of the reference clock (after the compensation), and a negative value indicates that a phone's clock is behind the reference clock. As shown in the Figure 7(b), the local clocks of phone 2 and phone 3 are both ahead of the reference clock in the first hour, and their clocks begin to drift behind the reference clock in the second hour. In comparison, the local clock of phone 1 drifts behind the reference clock first, then moves ahead of the reference clock in the second hour. Due to the varying clock drift rate, even if we use the initial clock drift rate to calibrate the phones, the difference between the local clocks of different phones can still differ by nearly $90ms$ within two hours (as in the case of phone 1 and phone 3).

B. Power Measurement

We are not able to fully implement the evaluated protocols on existing smartphones because the current WiFi's switching overhead is high. There are two possibilities moving forward. First, another network technology, e.g., Zigbee, can be used for message exchange while WiFi is used only to provide beacons for synchronization. Second, the future smartphone operating system may provide access to low level device driver that allows faster switching. To illustrate this, we conduct power measurement below to show that WiFi beacon transmission

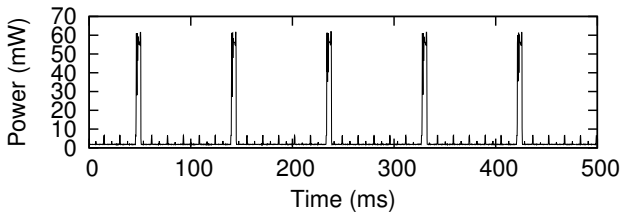


Fig. 8. Zigbee power consumption.

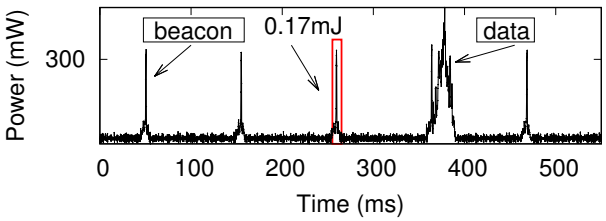


Fig. 9. Wi-Fi power consumption of a Samsung Galaxy S IV phone.

and fast switching can already be performed very efficiently by the underlying WiFi chipsets. Unfortunately, such features is not yet available to the applications.

We first present our measurement on Zigbee power consumption on a TelosB mote. The Zigbee is switched on every 100ms and send a small packet and is switched off after receiving a reply. In Figure 8, we can see that the time for one round of packet exchange takes about 4ms to 5ms. The total power used is about 262 μJ. The results show that 10ms is a reasonable slot for Zigbee interface, when a node can switch on interface and exchange a packet with a nearby node.

The Wi-Fi-based neighbor discovery protocols we have discussed all require a smartphone to frequently switch its Wi-Fi interface between active and non-active states to save power. In SearchLight's implementation [12], the authors reported a long delay (2s) for each Wi-Fi on/off switching operation on smartphones. Such a long delay is also observed in our own experiments. This apparently makes the current Wi-Fi chip non-suitable for low-duty-cycle neighbor discovery. For example, with a 2s of wakeup interval and when operating at a 1% of duty cycle, a smartphone can only wake up every 200s on average. However, the surge of Internet-of-Things applications has promoted a recent industry trend towards developing low-power Wi-Fi chips that come with much lower state switch overhead — some new chips can switch between ON/OFF states in as short as several milliseconds [22]. This trend favours the use of Wi-Fi-based local synchronization solution.

In this power measurement experiment, we examine the behavior of existing Wi-Fi chips in their power-save mode (when associated with an AP) to illustrate the potential for using Wi-Fi for neighbor discovery purpose. According to the IEEE 802.11 standard, a Wi-Fi client device in power-save mode periodically (every one or multiple beacon intervals) wakes up to receive the information about the presence of its buffered data from its associated AP. In our experiment, we use a MonSoon power monitor[8] to measure the power consumption of a Samsung Galaxy S II, S III, and S IV phone operating in power-save mode. The results show, as expected, Wi-Fi switch overhead (between active state and inactive state) decreases with newer generation phones. Here we discuss the results for the Galaxy S IV phone.

The lower peaks in Figure 9 illustrate the power consumed by the periodic wakeup of a Wi-Fi chipset to receive the AP's beacon frame. The phone wakes up and waits for the arrival of the beacon frame from the AP about every 100ms. It then immediately returns to inactive mode after receiving the beacon frame if there is no further communication. For such interactions, with the proper firmware support for power-save mode, the measurement shows that the Wi-Fi interface can turn on, receive the beacon packet, and then turn off within 3ms with peak consumption at around 300mW and an average energy usage of around 0.17mJ for each wakeup event. If WiFi's active transmission power is 0.5W, 0.17mJ per 100ms translates to about $0.17mJ/(0.5W \times 100ms) = 0.34\%$ of duty cycle, this is below the 1% target we aim at.

The highest peak in Figure 9 shows the power consumption profile of a wakeup period with additional message exchanges. Specifically, the smartphone sends one ping request packet and receives one reply packet. The peak power consumption increases to around 600mW and the whole active period spans for nearly 35ms.

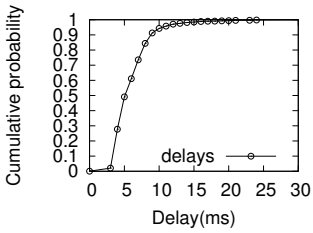
While very short active periods in the power-save mode are currently only available for a Wi-Fi device associated with an AP, the power profile measurements provide evidence on what is feasible for Wi-Fi-based neighbor discovery. In particular, even for existing Wi-Fi chips, if proper firmware support is provided, they should be able to support a wakeup duration of 10ms to 50ms. For concreteness, our discussions and evaluation assume Wi-Fi active power of 500mW and a wakeup slot length of 20ms (Table II). We assume $20ms$ instead of $3ms$ as measured in the power save mode, since: 1) R2 can only control the Wi-Fi chip via the interface exposed by the driver, hence it cannot turn on / off the Wi-Fi interface as fast as the logic the driver internally implements for the power saving mode. 2) R2 needs to conduct more operations than the default behavior in the power saving mode. To achieve a duty cycle of 1%, R2 will transmit/receive once every 2s, on the average, not every 100ms. Note that R2 does not actually require the mobile device to be associated to an AP and hence does not need to perform periodic listening every 100ms of the beacons. R2's reception of APs' beacons is for the mobile devices to obtain time references without association. The operations to obtain new local time references are needed only in the case that a device moves. Hence, it can be done every few seconds.

C. System Evaluation

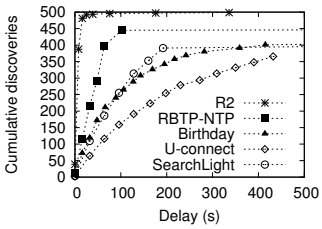
In this Section, we demonstrate the feasibility of a locally synchronized protocol on smartphones.

First, we show that publicly deployed Wi-Fi APs can provide timely beacons for synchronization with high probability. It is known that in response to a probe request from a Wi-Fi

[8]http://www.msoon.com/LabEquipment/PowerMonitor/



(a) Probe reply delay from APs.



(b) 6-phone experiment result.

Fig. 10. The measurement results obtained from smartphones.

device, a Wi-Fi AP will response with probe reply. This is true even if the mobile device has no access permission. We run an experiment where a smartphone periodically sends probe requests and records the delay of receiving respective probe replies from surrounding APs. We run the programme at 5 different locations, where a phone can hear packets from more than 10 APs. It is also run at different time of the day, when we expect varied background Wi-Fi data traffics. The results are shown in Figure 10(a). From the results, we can see that more than 95% of probe replies can be received within 10ms after a phone broadcasts a probe request indicating timely feedback is generally available.

Next, we measure the ability of smartphones to continuously synchronize with each other. The experiment uses two smartphones. Initially, one of the smartphones sends a probe request and the AP responses with a probe reply that can be heard by both phones, which are rooted and set the Wi-Fi interface to monitor mode. The first phone then periodically sends a single hello message (message transmission time is less than 1ms over Wi-Fi) every one second after the probe reply is received. The second smartphone listens for the hello message and computes the difference between the expected time of reception and actual reception time. The first smartphone sends a probe request every 5 minutes to mitigate the effect of clock drift. Results show that with an allowed time offset of 10ms, 93% of the total hello messages can be received over a 6 hour period. A short synchronization timeslot of more than 10ms is thus sufficient for discovery purpose if the local synchronization is performed every few minutes.

Finally, we perform a small experiment whereby we distribute Galaxy Nexus phones to 6 volunteers who are asked to move around two floors of a building. Wi-Fi on each phone runs in monitor mode, and broadcasts one hello message every second. At the same time, each phone records all the hello messages it has received. In this way, we collected the ground truth on phone-to-phone proximity and AP beacons with timestamps over a 6 hour period. We evaluate the performance of various protocols using the trace collected and the results with 1% duty cycle are shown in Figure 10(b). While all phones are always awake duration the trace collection, for each protocol we only consider the periods that a phone is supposed to be awake according to its schedule. See Section V-D for more details about the different protocols. The results show that R2 works efficiently and is able to discover significantly more neighbors than other protocols.

D. Performance Evaluation

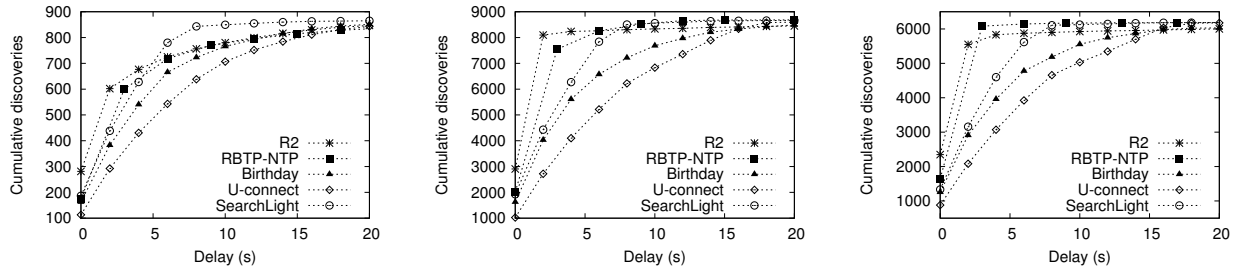
We now use simulation to compare R2's performance to the following protocols: (1) Birthday protocol, (2) Searchlight, (3) RBTP-NTP, and (4) U-connect. We use the default values listed in Table II for our evaluation. For each protocol, we determine its parameter settings according to the duty cycle used. For RBTP-NTP, we set the parameter a in Equation (2) to the optimal value (i.e., close to 1). This means that around half of RBTP-NTP's energy consumption is used for neighbor discovery and the other half for synchronization. For R2, if a phone cannot hear any AP, it runs the Birthday protocol. While we analyze the worst-case delay of different protocols earlier (under certain assumptions) as a useful metric to examine, we conducted trace-driven experiments to show how many contacts a specific protocol can actually discover. As most existing work in neighbor discovery protocol designs, we use this as the key metric to compare different protocols.

We use the traces listed in Table I with some modifications. For roller-net, which does not have location information for nodes hence prevents us from simulating the dynamic set of Wi-Fi APs that a mobile node overhears. Instead, we generate a static trace with 30 nodes and 9 APs that are all within the communication range of each other. In this baseline case, all protocols can eventually discover all neighbors. For the state fair trace, we randomly place APs in the area so as to include local synchronization source for R2. The phone-to-phone transmission range (d) is 50m and phone-to-AP range (r) is 100m.

For performance metric, we consider one-way discovery delay. For two nodes A and B, the delay for A to discover B is the time gap between the start of the contact to the first time A receives a message from B directly.

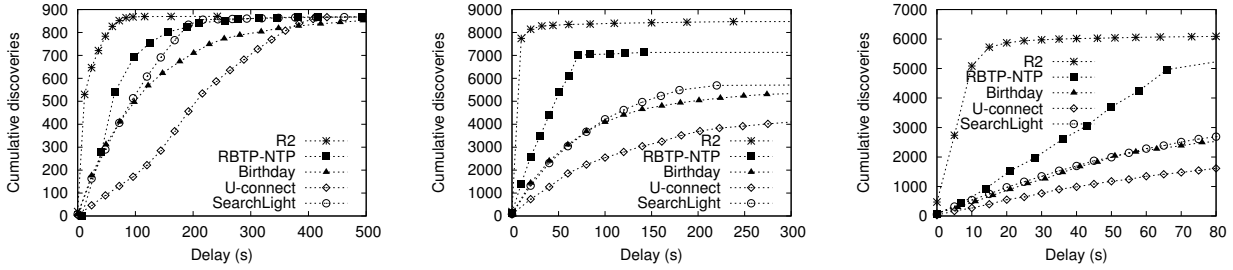
Figures 11 and 12 show the results for duty cycles of 5% and 1% respectively. For all protocols, their performance degrades as duty cycle decreases. This is true for all traces. The key takeaway is thus to examine the changes in relative performance of different protocols and whether they can meet the low-delay low-duty-cycle requirements.

With a high duty cycle of 5%, it can be observed that the asynchronous protocols, i.e., Searchlight and U-connect, perform relatively well and R2 does not perform significantly better than the other protocols. In fact, one can argue that Searchlight performs the best in the static scenarios. Overall, all protocols can discover most of the contacts with short discovery delay ($< 20s$). For our targeted low duty cycle of 1%, the results clearly show that synchronous protocols outperform asynchronous protocols. For all three traces, the two protocols with the best performance are always R2 and RBTP-NTP. As expected, R2 outperforms RBTP-NTP due to its lower synchronization overhead. In fact, out of all protocols, only R2 can discover the majority ($> 80\%$) of the contacts within $30s$. We have also evaluated with lower duty cycles of $< 1\%$. In general, the performance improvement of R2 over the other protocols increases with decreasing duty cycles. The relative performance of different protocols under varying duty



(a) Static (b) Food Court (c) State Fair

Fig. 11. Cumulative number of neighbor discoveries of different protocols and traces (5% duty cycle)



(a) Static (b) Food Court (c) State Fair

Fig. 12. Cumulative number of neighbor discoveries of different protocols and traces (1% duty cycle).

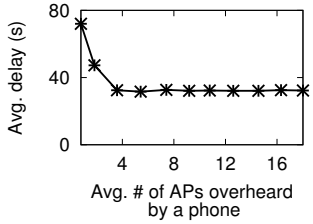


Fig. 13. R2's performance with varying number of APs.

cycles matches well with our analysis (see Table IV).

Another important observation in Figure 12 is that R2 outperforms all other protocols in all the three traces despite of their varying density of phones. In particular, there are nearly 100 of simultaneously present phones during peak hours for the food court trace. This is consistent with the controlled study presented earlier in Figure 4(b).

Finally, we evaluate the effects of the number of Wi-Fi APs on R2's performance. We add different number of APs uniformly randomly over the space into the state fair trace. As shown in Figure 13, R2 incurs long discovery delay only when the average number of APs a phone can overhear is extremely small (i.e., below 4). In this case, very often a phone does not have sufficient number (i.e., 3) of APs to use as time references. Beyond that, the average discovery delay sharply decreases and maintains at around $30s$. This result shows R2's performance scales with the increasing number of APs. This is also consistent with our analysis presented earlier in Figure 6(b).

VI. Related Work

We have conducted an analytical review of the major categories of neighbor discovery protocols in Section III. While the protocols we previously reviewed [7]–[13] except for [14] are designed mainly for pairwise neighbor discovery, there are several recent efforts (e.g., [23]–[25]) that focus on improving collaborative neighbor discovery performance for a cluster of devices.

Purohit et al. proposed WiFlock [23], which optimizes the neighbor discovery process together with the group formation and maintenance process. Specifically, WiFlock synchronizes the activities of a group of neighboring nodes through distributed coordination, so as to speed up the propagation of group membership information. Zhang et al. proposed Acc [24], which leverages two-hop neighborhood information (as conveyed in the beacon messages of one node's already discovered neighbors) to accelerate the neighbor discovery process. Zhang et al. also proposed the EQS [25] protocol, which extends the quorum system to leverage indirect discovery for further reduction of energy consumption. Similar to these works, R2 also leverages distributed coordination as well as available information from neighboring nodes to facilitate discovery and group maintenance. R2's design differs in the following two aspects: First, the existing collaborative protocols provide enhancement to asynchronous protocols, while R2 is by design a locally synchronized protocol; since R2 uses in-situ APs as local time references, this gives a natural way for R2 to synchronize all nodes that adopt the same reference AP. Second, to enhance its scalability with increasing neighborhood size, R2 breaks the symmetry by introducing the role of scout and amortizes the overhead by changing the node that serves as the scout in each round.

Some more recent research efforts have specifically focused on neighbor discovery for smartphones. In [26], Bracciale et al. posted and investigated the following question: how many contacts can a smartphone discover given a fixed battery energy

budget? Based on the analysis of the stochastic characteristics of mobile nodes meeting process, they proposed a simple globally synchronized discovery protocol that adapts the duty cycle according to the environment. Similar to [13], they didn't explicitly consider the time synchronization overhead. Hence, they didn't compare the potential gain of a locally synchronized protocol like R2 due to the reduced synchronization overhead. Han et al. proposed the eDiscovery protocol [27] for Bluetooth communication, which dynamically adjusts Bluetooth inquiry durations and intervals according to the environment to improve on the tradeoff between energy and number of neighbors discovered. However, since eDiscovery is an asynchronous protocol, its worst case asymptotic performance is still subject to the analytical bounds presented in Section III. Want et al. proposed AIR protocol [28], which opportunistically uses ambient acoustic events for fast neighbor discovery. AIR is similar to R2 in the sense that they both use locally available information to synchronize the phones and speed up the neighbor discovery process. AIR's evaluation results also show that locally-synchronized protocols can potentially perform better than asynchronous protocols. However, AIR's duty cycle and discovery delay depend on the ambient audio events which are often out of the control of the protocol. Further, AIR does not explicitly consider the impact of the phone density. Han and Li proposed P-Game protocol [29], which considers the optimization over different phone density. P-Game is designed for one-to-K neighbor discovery, where K is a pre-specified number of neighbors to discover. In comparison, R2 aims to discover all neighbors.

VII. Conclusion

In this work, we extensively investigated the smartphone neighbor discovery problem, especially practical settings under power consumption constraints. We analytically reviewed different categories of neighbor discovery protocols. The investigation and review guided us to design R2, a locally synchronized protocol that achieves low neighbor discovery delay with low duty cycle. Its Rendezvous-Recon mechanism and minimum-κ design effectively addresses the scalability issue faced by locally synchronized protocols.

Acknowledgment

The authors thank the anonymous reviewers for their insightful comments. This study is partly supported by the Agency for Science, Technology and Research (A*STAR), Singapore, under both the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center and SERC Grant 1224104049.

References

[1] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *Wireless Communications, IEEE*, 20(3), 2013.

[2] Murat Demirbas, Murat Ali Bayir, Cuneyt Gurcan Akcora, Yavuz Selim Yilmaz, and Hakan Ferhatosmanoglu. Crowd-sourced sensing and collaboration using twitter. In *WoWMoM 2010*. IEEE, 2010.

[3] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 2010.

[4] Aaron Yi Ding, Bo Han, Yu Xiao, Pan Hui, Aravind Srinivasan, Markku Kojo, and Sasu Tarkoma. Enabling energy-aware collaborative mobile data offloading for smartphones. In *SECON*. IEEE, 2013.

[5] Girisha De Silva, Binbin Chen, and Mun Choon Chan. Collaborative cellular tail energy reduction: Feasibility and fairness. In *in Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN16)*, January 2016.

[6] Ellie D'Hondt, Matthias Stevens, and An Jacobs. Participatory noise mapping works! an evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring. *Pervasive and Mobile Computing*, 2013.

[7] Michael J McGlynn and Steven A Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *MobiHoc*, 2001.

[8] Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Yueng Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. In *INFOCOM*, 2002.

[9] Jehn-Ruey Jiang, Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Hwang Lai. Quorum-based asynchronous power-saving protocols for IEEE 802.11 ad hoc networks. *Mobile Networks and Applications*, 2005.

[10] Prabal Dutta and David Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *SenSys*, 2008.

[11] Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *IPSN*, 2010.

[12] Mehedi Bakht, Matt Trower, and Robin Hilary Kravets. Searchlight: won't you be my neighbor? In *Mobicom*, 2012.

[13] Dong Li and Prasun Sinha. RBTP: Low-Power Mobile Discovery Protocol through Recursive Binary Time Partitioning. *Mobile Computing, IEEE Transactions on*, 13(2):263–273, 2014.

[14] D. Camps-Mur and P. Loureiro. E^2D Wi-Fi: A Mechanism to Achieve Energy Efficient Discovery in Wi-Fi. *Mobile Computing, IEEE Transactions on*, PrePrints, 2014.

[15] Jin Shuaizhao, Wang Zixiao, Leong Wai Kay, Ben Leong, Dong Yabo, and Lu Dongming. Improving neighbor discovery with slot index synchronization. In *Mobile Adhoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE, 2015.

[16] Li Sun, Ramanujan K Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. Modeling wifi active power/energy consumption in smartphones. In *ICDCS*, 2014.

[17] Keyu Wang, Xufei Mao, and Yunhao Liu. Blinddate: A neighbor discovery protocol. *Parallel and Distributed Systems, IEEE Transactions on*, 26(4):949–959, April 2015.

[18] Bluetooth Find Me Profile Specification, available at https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239389.

[19] Philipp Kindtz, Marco Saur, and Samarjit Chakraborty. Neighbor discovery latency in BLE-like duty-cycled protocols, available at http://arxiv.org/abs/1509.04366, 2015.

[20] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *MobiSys*. ACM, 2012.

[21] Fred L Walls and Jean-Jacques Gagnepain. Environmental sensitivities of quartz oscillators. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 39(2):241–249, 1992.

[22] GainSpan company. https://www.gainspan.com/docs2/low_power_wi-fi_for_smart_ip_objects_wp_cmp.pdftec.

[23] Aveek Purohit, Bodhi Priyantha, and Jie Liu. Wiflock: Collaborative group discovery and maintenance in mobile sensor networks. In *IPSN 2011*. IEEE.

[24] Desheng Zhang, Tian He, Yunhuai Liu, Yu Gu, Fan Ye, Raghu K Ganti, and Hui Lei. Acc: generic on-demand accelerations for neighbor discovery in mobile applications. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM, 2012.

[25] Desheng Zhang, Tian He, Fan Ye, Raghu K Ganti, and Hui Lei. Eqs: Neighbor discovery and rendezvous maintenance with extended quorum system for mobile sensing applications. In *ICDCS 2012*. IEEE.

[26] L. Bracciale, P. Loreti, and G. Bianchi. The sleepy bird catches more worms: revisiting energy efficient neighbor discovery. *Mobile Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[27] Bo Han, Jian Li, and A. Srinivasan. On the energy efficiency of device discovery in mobile opportunistic networks: A systematic approach. *Mobile Computing, IEEE Transactions on*, 14(4):786–799, April 2015.

[28] Keyu Wang, Zheng Yang, Zimu Zhou, Yunhao Liu, and L. Ni. Ambient rendezvous: Energy-efficient neighbor discovery via acoustic sensing. In *INFOCOM 2015*, April 2015.

[29] Junze Han and Xiang-Yang Li. Pickup game: Acquainting neighbors quickly and efficiently in crowd. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pages 82–90, Oct 2014.