SearchLight: A Systematic Probing-based Asynchronous Neighbor Discovery Protocol

Mehedi Bakht and Robin Kravets
Dept. of Computer Science, University of Illinois at Urbana-Champaign
Email: {mbakht2,rhk}@cs.illinois.edu

***Abstract*—The rapid deployment of millions of smart phones has resulted in a demand for proximity-based social networking applications. However, the usefulness of these applications is limited by the lack of effective and energy efficient neighbor discovery protocols. While probabilistic approaches perform well for the average case, they exhibit long tails resulting in high upper bounds on neighbor discovery time. Recent deterministic protocols, including Disco and U-Connect, improve on the worst case bound, but do so by sacrificing average case performance. In response to these limitations, we present Searchlight, an asynchronous neighbor discovery protocol that combines both deterministic and probabilistic components. For symmetric nodes that all have the same duty cycle, this novel combination achieves an average case performance comparable to the probabilistic approaches and improves on the deterministic worst case bounds. Additionally, we show that for asymmetric cases, Searchlight performs comparably to the deterministic protocols when there is a high degree of asymmetry, but can improve on their performance if any of the nodes maintain the same duty cycle. We validate Searchlight through a series of analysis, simulation and real-world experiments on smartphones that show considerable improvement in discovery latency over existing approaches.**

I. Introduction

Ranging from smart phones to sensor motes, mobile devices capable of communicating wirelessly are everywhere. While the communication usually takes place between a device and an installed infrastructure, most of these devices are also equipped with radios that support direct device-to-device communication. This ad hoc networking potential can be exploited to support peer-to-peer communication based on physical proximity. Although people may be geographically co-located for different reasons, including special events (e.g., conferences, sports events, etc.) or common locations (e.g., an office building or coffee shop), recent surveys [7], [12] show that there is a strong demand for communication among such geographically co-located people. In response to such demand, applications (e.g., MobiClique [10], WiFace [12]) are emerging that seek to exploit opportunistic contacts between mobile users to extend online social networking services to the physical domain (e.g., a user would get notified if a Facebook "friend" is nearby) or facilitate creation of new social graphs based on physical proximity.

Because of the reliance on opportunistic encounters, a precondition to the success of these emerging applications is the ability of a node to efficiently discover the presence of other nodes in its transmission range. The more devices that can be discovered, the better the performance of these mobile social networking applications will be. However, for a device running on battery, and hence a limited energy budget, it is not practical to continuously search for neighbors. A more feasible approach is to keep the wireless interface in a sleep state most of the time and periodically wake it up to execute a discovery process. The interval between wakeups can be adjusted to operate at a particular duty cycle.

The success of such duty-cycling schemes depends on ensuring that the wakeup times of two neighboring nodes overlap. This is not hard to achieve when node clocks can be synchronized, for example through GPS [6]. However, when nodes are not time synchronized, which may be the case for networks of mobile sensors and smartphones, it becomes challenging for a periodic scheme to ensure such an overlap within a reasonable time bound while operating at low duty-cycles. Further challenges arise because nodes may either operate at the same duty cycle (the *symmetric* case) or may have dissimilar energy requirements and operate at different duty cycles (the *asymmetric* case). Existing asynchronous discovery protocols address these challenges by either probabilistically deciding on wakeup times (i.e., Birthday protocol [8]) or deterministically designing a schedule for wakeup times that is guaranteed to overlap (i.e., Quorum-based protocols [5], [11], Disco [3] and U-Connect [4]). While protocols adopting the first approach have good average case behavior and protocols belonging to the second group have better bound on worst-case latency, none of these protocols perform well in *both* cases.

To overcome these limitations, we present Searchlight, an asynchronous neighbor discovery protocol that strikes a balance between the two conflicting goals of low-power operation and small discovery latency. By adopting a systematic approach that has both deterministic and probabilistic components, Searchlight achieves average-case performance comparable to the probabilistic protocols and significantly improved worst-case bounds for symmetric operation in comparison to the current best deterministic protocols. In the asymmetric case, all protocols operate similarly. However, if multiple nodes share the same duty cycle, even when other nodes use different duty cycles, Searchlight can take advantage of this partial symmetry and again provide improved performance. As validation of our protocol, we present extensive simulation results that show that Searchlight performs better than the existing protocols in terms of average discovery latency in a variety of scenarios (up to 25% improvement over recent protocols for very low duty cycles). To gauge the potential

of Searchlight in a real network setting, we implemented the protocol on commercial off-the-shelf hardware, more specifically Android G1 smartphones. Since previous asynchronous neighbor discovery algorithms were all implemented on sensor motes, our implementation provides valuable new insight to issues related to execution of such protocols on smartphones with WiFi interfaces. Our evaluations show that the performance of the Searchlight implementation closely matches the simulation results.

The rest of this paper is as follows. Section II briefly describes existing approaches for asynchronous neighbor discovery and discusses why they fail to meet all of the performance goals. Section III describes Searchlight in detail, including analysis that shows why it is better than existing protocols for the symmetric case. Section IV presents our simulation-based performance evaluation of Searchlight in comparison to other neighbor discovery protocols. A prototype implementation of Searchlight on a smartphone testbed is described in Section V. We conclude and outline the direction of our future research in Section VI.

II. Energy-efficient Asynchronous Neighbor Discovery Protocols

Asynchronous neighbor discovery algorithms mostly work on a time-slot basis, where time is assumed to be divided into slots of equal size and all nodes agree on the size of a slot. Based on the protocol used, nodes decide to remain awake during specific slots, which are called *active* slots, and sleep during the remaining slots. During an active slot, the node may send/receive or do both, depending on application requirements. Successful discovery takes place between two neighbors whenever their active slots overlap. To be energy efficient, a discovery scheme needs to use as few active slots as possible to discover neighbors within a reasonable time limit. Current approaches to energy-efficient asynchronous neighbor discovery fall broadly into three categories - probabilistic, quorum-based, and deterministic. The relative strengths and weaknesses of these existing schemes can be judged by answering the following three questions:

- Flexibility: Can the protocol handle both symmetric and asymmetric duty cycles?
- Average-case Performance: Does the protocol do well in *most* of the cases?
- Worst-case Latency: Does the protocol provide an acceptable strict bound on the worst-case discovery latency?

Most well-known among probabilistic approaches is a family of "birthday protocols" [8] where nodes transmit/receive or sleep with different probabilities. This scheme works well in the average case and allows asymmetric operation. However, the main drawback of the birthday protocol is its failure to provide a bound on the worst case discovery latency, leading to long tails on discovery probabilities.

In the Quorum-based protocols [5], [11], time is divided into sets of m^2 contiguous intervals. These m^2 intervals are arranged as a 2-dimensional $m \times m$ array and each host can pick one row and one column of entries as awake intervals. This ensures that no matter what row and column are chosen, any two nodes have at least two overlapping awake intervals. While the Quorum protocol provides a reasonable bound on worst-case latency, it performs much worse than the probabilistic approach in the majority of the cases. Also, the initial approach [11] lacks flexibility since m is a global parameter and hence supports only symmetric operation. Lai et al. [5] improved that scheme to handle asymmetric cases when there are only two different schedules in the entire network. Another approach that works mainly for the symmetric case is the application of block design using difference sets to the problem of asynchronous neighbor discovery [13]. For the asymmetric case, designing the appropriate schedule following the proposed scheme becomes similar to the vertex-cover problem, which is an NP-complete problem.

Deterministic protocols overcome this limitation of being applicable to only symmetric cases and can handle both symmetric and asymmetric operation while still providing a strict bound on worst-case latency. In Disco, each node chooses a pair of prime numbers such that the sum of their reciprocals are as close as possible to the desired duty cycle. The nodes then wake up at multiples of the individual prime numbers. If one node chooses primes p_1, p_2 and another node chooses p_3, p_4, the worst-case discovery latency between these two nodes will be $min\{(p_1 \cdot p_3), (p_1 \cdot p_4), (p_2 \cdot p_3), (p_2 \cdot p_4)\}$, provided the two primes in the pair are not equal. A more recent deterministic approach, U-Connect [4], uses a single prime per node. Instead of just waking up only 1 slot every p slots, the nodes also wake up $\frac{p+1}{2}$ slots every p^2 slots. The worst-case latency for U-Connect is p^2, which is similar to Disco. However, for the energy-latency product, a metric proposed by the authors to evaluate energy-efficiency of asynchronous neighbor discovery protocols, U-Connect provably fares better than Disco in the symmetric case. Although these deterministic protocols have good worst-case performance, in the majority of cases, they are worse than the birthday protocol.

To successfully meet all of the goals of flexibility, good average-case performance and reasonable worst-case latency, we present a new protocol named Searchlight. Searchlight follows a deterministic approach and provides a strict bound on worst-case latency, which is provably *better* than existing protocols in the symmetric case and similar when duty cycles are asymmetric. Additionally, it also incorporates randomization techniques that result in discovery latency very close to the probabilistic approach in the average case.

III. Searchlight

In this section, we describe the Searchlight protocol in details. We also provide the worst-case latency bound for both symmetric and asymmetric cases, and show that Searchlight is better than existing protocols in terms of energy efficiency using a previously proposed metric [4].

Searchlight is a periodic slot-based discovery scheme where a period consists of t contiguous slots. t is known as the period length and is determined based on the the specific duty cycle that a node wants to operate at. In every period, there are

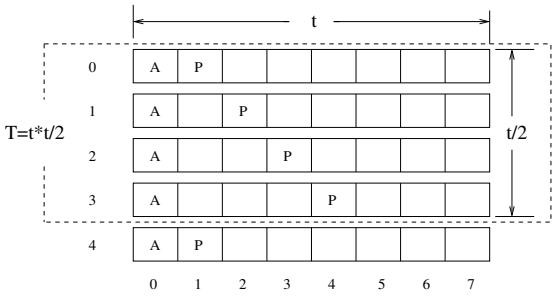


Fig. 1. Searchlight with sequential probing (t=8)

two active slots- an *anchor* slot (A) and a *probe* slot (P). The position of the anchor slot is fixed and it is the first slot (slot 0) in a period. If t is the same for two nodes, then the anchor slots overlap only if the relative phase offset is less than one timeslot. For all other offsets, the two anchor slots would never meet (assuming no clock drift) since the offset would remain constant during the encounter. This means that the relative position of the anchor slot of one node will remain the same with respect to that of the other node and will be in the range $[1, t-1]$.

The design of Searchlight is based on this key observation about the constant relative offset. Essentially, Searchlight introduces an additional active slot, called a probe slot, in each period. The objective of the probe slot is to search for the anchor slot of the other node in a systematic way. While the constant relative offset only holds in the symmetric case, we show Searchlight also operates in the asymmetric case.We next describe two approaches for determining the schedule for the probe slots.

A. Sequential Probing

In the sequential mode, referred to as Searchlight-S, the position of the probe slot is determined by a counter that starts at 1, increments by 1 every period, ends at $\lfloor \frac{t}{2} \rfloor$ and then starts at 1 again (see Figure 1). In other words, if P_i denotes the position of the probe slot in the i-th period, then

$$P_{i+1} = ((P_i + 1)\%\lfloor \frac{t}{2} \rfloor) + 1. \quad (1)$$

The position of the probe slot actually follows the pattern $\{1, 2, ..., \lfloor \frac{t}{2} \rfloor\}$ and this pattern gets repeated every $\lfloor \frac{t}{2} \rfloor$ periods, which we call the hyper-period T. For example, for $t = 9$, Searchlight-S uses the pattern {1,2,3,4} to determine the position of the probe slot in each period.

B. Randomized Probing

In randomized probing, called Searchlight-R, a node systematically moves around its probe slot to find the anchor slots of its neighbors. While this does not affect a node's ability to find an anchor slot, it enables this probe to find other probe slots. Essentially, if the probe slots of two nodes follow the same pattern, they will be in sync with each other, which greatly reduces the probability of a probe-probe overlap. By randomizing the probing, Searchlight benefits from the ideas of the Birthday Protocols and enables probe slots to overlap with a higher probability..

We illustrate this point with the following example (see Figure 2). Assume that two nodes A and B are neighbors, they have a relative phase offset of one slot, and $t = 8$. When they first meet, A's probe slot is at position 2, while B's at position 3 (with respect to its own anchor slot). Since both A and B follow the pattern {1,2,3,4}, the next positions of the probe slots will be 3 and 4 respectively. Thus, A and B's probe slot "chase" each another without ever overlapping at any point.

In comparison, Searchlight-R allows nodes to *randomly* pick any permutation of values from 1 to $\lfloor \frac{t}{2} \rfloor$ as the pattern for moving the probe slot. For example, for $t = 9$, instead of being restricted to just {1,2,3,4}, nodes can choose any pattern which is a permutation of the integers 1,2,3 and 4 (e.g., {1,4,3,2},{1,2,4,3}). This modification creates the possibility for two neighbors to pick different patterns, resulting in increased probability of discovery through overlap between the probe slots.

Going back to the earlier example, assume that A randomly chooses the pattern {1,4,2,3} while its neighbor B, chooses the pattern {1,3,2,4} (see Figure 3). When they first meet, the probe slots of A and B are at positions 4 and 4 respectively and their relative offset is 1 slot. Because of the phase offset, the probe slots miss each other initially but meet in the next period when A's probe slot moves to position 2 and B's slot moves to position 1. Thus, the use of different patterns instead of the sequential one results in quicker discovery through overlap of probe slots.

C. Discovery Latency

In this section, we present the analysis for worst-case discovery latency under Searchlight for the symmetric case.

Lemma 3.1: The worst-case discovery latency under Searchlight with parameter t is equal to $\frac{t^2}{2}$ slots.

For two nodes x and y, let $\phi(x, y)$ be the phase offset (in slots) from the anchor slot of x (A_x) to the anchor slot of y (A_y). Similarly, let $\phi(y, x)$ be the phase offset from A_y to A_x (see Figure 4).

Clearly,

$$\phi(x, y) + \phi(y, x) = t. \quad (2)$$

In the symmetric case, where both nodes use the same t, $\phi(x, y)$ and $\phi(y, x)$ remain constant during the contact. It follows from Equation (2) that $min(\phi(x, y), \phi(y, x)) \leq \frac{t}{2}$.

For both Searchlight-S and Searchlight-R, the probe slot goes through all positions from 1 to $\frac{t}{2}$ every $\frac{t}{2}$ periods. Now, let us denote the probe slots of x and y as P_x and P_y respectively. P_x will meet A_y within $\frac{t}{2}$ periods as long as $1 \leq \phi(x, y) \leq \frac{t}{2}$. Similarly, P_y will meet A_x within $\frac{t}{2}$ periods as long as $1 \leq \phi(y, x) \leq \frac{t}{2}$.

Since at least one from ($\phi(x, y)$,$\phi(y, x)$) is guaranteed to be less than or equal to $\frac{t}{2}$, the worst-case latency can be at most $\frac{t}{2}$ periods or $\frac{t^2}{2}$ slots.

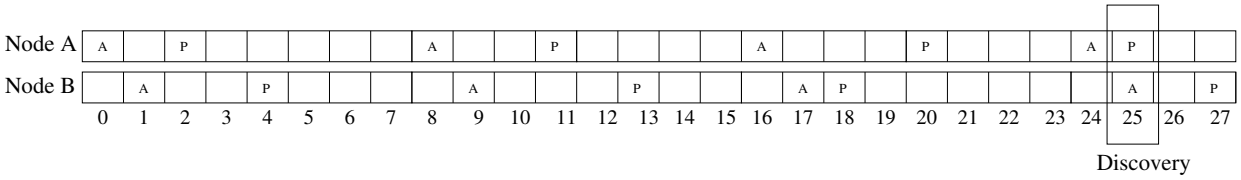


Fig. 2. Overlap with sequential probing (t=8)

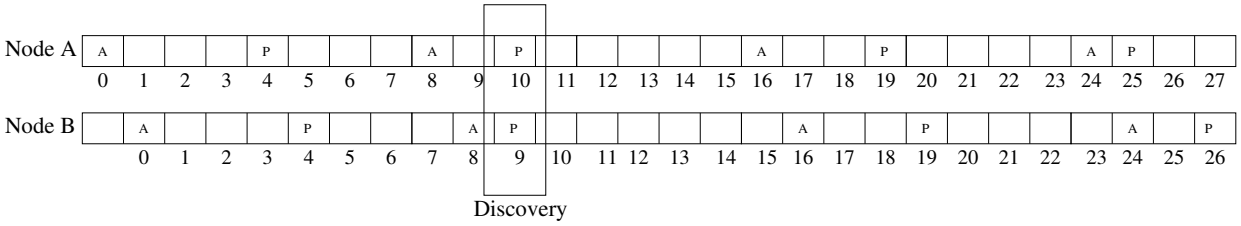


Fig. 3. Overlap with randomized probing (t=8)

D. Energy-Latency Metric

Energy-latency, Λ [4], is a metric that is defined as the product of the average energy consumption (in terms of active slots) P and the worst-case neighbor discovery latency L in an ideal communication channel. Using this metric, we show that Searchlight is a 1.41 approximation algorithm in contrast to U-Connect's 1.5- and Disco/Quorum's 2-approximation algorithm. It should be noted that this metric only works for the symmetric case.

To evaluate Searchlight using this metric, we start with the worst-case latency L_s, from Lemma 3.1:

$$L_s = \frac{t}{2} \text{ periods } = \frac{t^2}{2} \text{ slots .} \tag{3}$$

Next, average energy consumption P is defined as:

$$P = \frac{\text{number of active slots in T}}{\text{Period of the discovery schedule } (T)}. \tag{4}$$

For Searchlight, the active-slot schedule repeats every hyper-period or $T_s = t \cdot \frac{t}{2}$ slots. Given A_T and P_T, the number of anchor slots and probe slots every T slots, the number of active slots every T_s slots is $A_T + P_T$. Therefore, the average energy consumption P_s in Searchlight is given by:

$$P_s = \frac{1}{T}(A_T + P_T) = \frac{\frac{t}{2} + \frac{t}{2}}{T} = \frac{2}{t}. \tag{5}$$

From equations (3),(5), Λ_s for Searchlight is:

$$\Lambda_s = P_s L_s = \sqrt{2L}. \tag{6}$$

For the theoretically optimal schedule [4], [13], the PL product is:

$$\Lambda_o = \sqrt{L - \frac{3}{4}} + \frac{1}{2}. \tag{7}$$

As can be seen from equations (11) and (7), asymptotically, the Searchlight protocol is at most a $\sqrt{2}$-approximation algorithm for the symmetric asynchronous neighbor discovery problem.

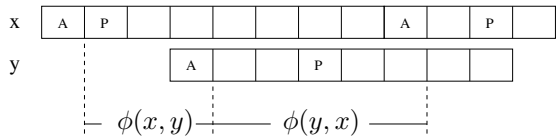


Fig. 4. Phase offsets (t=8)

$$\lim_{L\to\infty} \frac{\Lambda_s}{\Lambda_o} = \lim_{L\to\infty} \frac{\sqrt{2L}}{\sqrt{L - \frac{3}{4}} + \frac{1}{2}} = \sqrt{2}. \tag{8}$$

E. Asymmetric case

The asymmetric case arises when two nodes with different duty cycles try to discover each other. Since available energy in a device can vary considerably, such a scenario is not unlikely. A good asynchronous neighbor discovery protocol should be flexible enough to perform well in both symmetric and symmetric cases. Among existing protocols, Quorum cannot handle asymmetric cases while, for Disco, the choice of the primes have to be tuned for either asymmetric or symmetric cases but not both.

If it is known beforehand that Searchlight needs to handle asymmetric cases, the only change to the original protocol is the choice of value for period t, which needs to be restricted to primes only. This is done to ensure that for any two nodes operating at different duty cycles, their period lengths will be relatively prime, i.e., they will have no common factors other than 1. In terms of performance, the worst-case latency for Searchlight changes for the asymmetric case. This is because for two nodes operating at different duty cycles, the relative offset no longer remains constant. Hence, the systematic probing by the probe slot for the anchor slot is not guaranteed to succeed within a particular time limit. Instead, the worst-case bound in the asymmetric case is based on the overlap between anchor slots.

Since period lengths are relatively prime, it follows from the Chinese Remainder Theorem [9] that the two anchor slots would overlap at least once every $t_1 \cdot t_2$ slots, where t_1 and t_2 are the two different period lengths. For U-Connect, the worst-case latency in the asymmetric case is the same as in the symmetric case, which is $p_1 * p_2$, where p_1 and p_2 are the two different primes. While this latency bound is similar to that of Searchlight, it should be noted that U-Connect uses smaller primes than Searchlight's period lengths when operating at the same duty cycle. This means effectively, for the same duty cycle, U-Connect has a slightly lower latency bound. However, the randomized component of Searchlight-R makes up for this larger worst-case latency by faring better in the average case, as we will see in the evaluation section(see Section IV). Also, in a real life scenario, the duty cycle will not be different for every single pair of nodes. A more realistic scenario is that there will be 2/3 defined levels of duty cycling, and a node will dynamically chose one based on its energy needs. This means that in any network, there will be always pairs of nodes that will operate at the same duty cycle. Searchlight will perform *best* in those cases, where as for the asymmetric pairs, it will perform as good as other existing protocols.

F. Use of more probe slots

Since the use of probe slots significantly improves the latency bound in the symmetric case, it is natural to ask whether introducing more probe slots would result in better performance. For example, instead of one probe slot per period, we can have two probe slots where the first one will run from 1 to $\frac{t}{4}$, and the other one will run from $\frac{t}{4}$ to $\frac{t}{2}$. Will this result in better performance? We answer this question by using the Energy Latency metric.

With two probe slots, the schedule will repeat every $\frac{t}{4}$ periods. So, the worst-case latency L_s is :

$$L_s = \frac{t}{4} \text{ periods } = \frac{t^2}{4} \text{ slots .} \tag{9}$$

The number of active slots every $t/4$ periods is $\frac{3t}{4}$. Therefore, the average energy consumption P_s is given by:

$$P_s = \frac{\frac{3t}{4} + \frac{t}{4}}{T} = \frac{3}{t}. \tag{10}$$

From equations (9),(10), Λ_s for two probe slots is :

$$\Lambda_s = P_s L_s = \frac{3}{2} \cdot \sqrt{L}. \tag{11}$$

Comparing with Equation 7, we can see that the use of an additional probe makes Searchlight an 1.5-approximation algorithm for the symmetric neighbor discovery problem, which is worse than using one probe slot.

IV. Evaluation

The primary goal of our evaluation is to show that Searchlight achieves significant performance gains over other asynchronous neighbor discovery protocols by virtue of its systematic probing-based approach. Specifically, we evaluate how long it takes for different protocols to discover neighbors when they spend the same amount of energy, i.e., operate at the same duty cycle. The time from when two nodes first get in each other's transmission range to the time when they actually discover each other is known as the *discovery latency*. We look at the CDF of discovery latencies to understand the overall trend of a protocol. To measure what an individual node can usually expect, we look at average latency. We compare the performance of our approach with two recent deterministic protocols, Disco and U-Connect, and also with the Birthday protocol, the most well-known probabilistic approach. Our evaluation is based on two kinds of simulation - state-based and simulator-based.

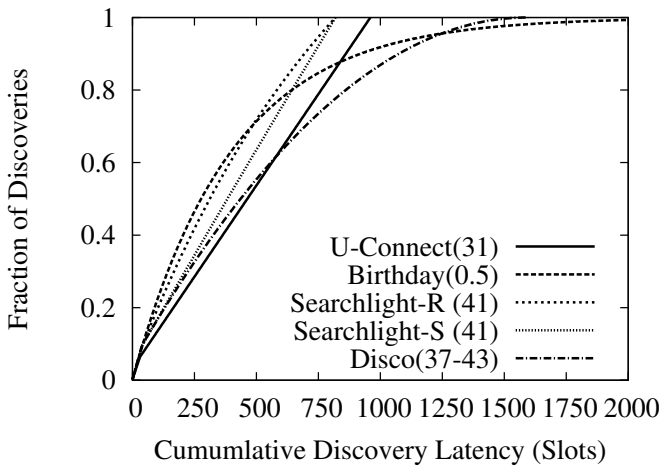


Fig. 5. Cumulative Discovery Latency for 5% duty cycle

A. State-based Simulation

Except for the Birthday protocol, all other protocols basically follow a discovery schedule to determine when to sleep and when to wake up. This schedule repeats every T slots, which we call the hyper-period. When a node is in a particular slot in its schedule, that slot index can be considered the $state$ of the node at that point and is always in the range $[0, T-1]$. When two nodes with hyper periods T_1 and T_2 come into each other's transmission range, their states have one of $T_1 \cdot T_2$ possible combinations. For a given combination, the discovery latency is always the same. For Disco, U-Connect and Searchlight-S, we use this observation to loop through all possible combinations and determine the latency for each case. The same approach is not feasible for Searchlight-R since for a given t, a node can choose any of the $(t-1)!$ patterns to determine the schedule of its probe slot. Instead, we run the protocol 1000 times with different seeds. At each run, we generate a new schedule for both nodes. Then, for that particular schedule pair, we loop through all possible state combinations like we do for Searchlight-S. For the Birthday protocol, we use closed form expressions for determining the CDF and the expected value of discovery latency [8].

1) Symmetric case: First, we compare the performance of different protocols when nodes operate at the same duty cycle. We look at the cumulative distribution of discovery

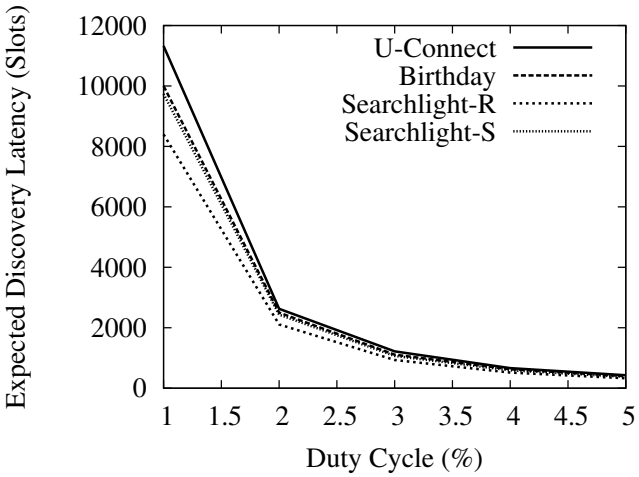


Fig. 6. Expected Latency vs Duty Cycle

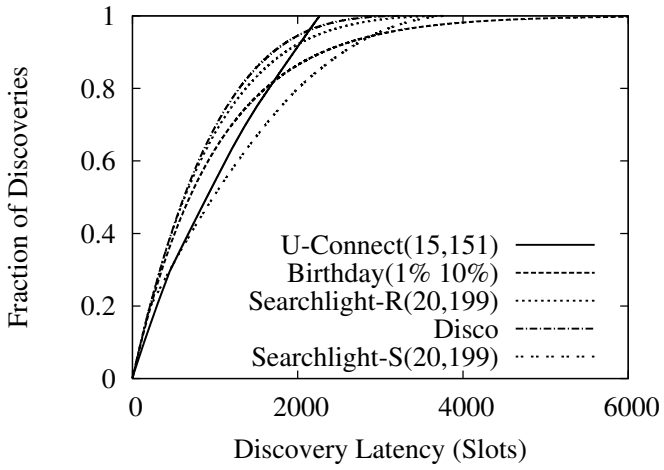


Fig. 7. 1%-10%

latencies for all protocols operating at a 5% duty cycle, Disco uses the primes (37,43), Birthday uses probability = 0.5, U-Connect uses the prime 31 and both versions of Searchlight use the prime 41 (see Figure 5). Searchlight-R *always* achieves lower latency than all other protocols except for the Birthday protocol. For 65% of the time, Searchlight-R performs on par with the Birthday protocol or slightly lags behind. Beyond that, the probabilistic nature of the Birthday protocol leads to a long tail and Searchlight-R achieves the lowest latency. In comparison to U-Connect, Searchlight-R achieves better latency all along, including a worst-case latency improvement of around 16%. Searchlight-S always performs better than U-Connect and Disco but lags behind Searchlight-R in the average case. However, latency in the worst-case is the same for Searchlight-S and Searchlight-R, which is the lowest among all protocols.

Next, we look at the expected discovery latency of different protocols for different duty cycles (see Fig. 6). For all duty cycles, Searchlight-R has the lowest expected latency and the difference with other protocols increases as the duty cycle decreases. When nodes operate at 1% duty cycle, Searchlight-R can reduce expected latency by as much as 25% for U-Connect and 16% for the Birthday protocol. Expected latency for Searchlight-S lies between that of Searchlight-R and the Birthday protocol for all duty cycles. This difference between the performance of Searchlight-R and Searchlight-S clearly demonstrates the advantage of incorporating randomization in moving the probe slot.

However, there is diminishing rate of returns in terms of how much improvement in discovery latency can be made by increasing the duty cycle (see Figure 6). For all protocols, we see that the expected latency drops considerably when the duty cycle is increased from 1% to 2%, but this rate of decrease slows down when protocols switch to higher duty cycles.

Overall, in the symmetric case, these results confirm that Searchlight does provide the lowest worst-case bound for a given duty-cycle as was shown earlier using the energy

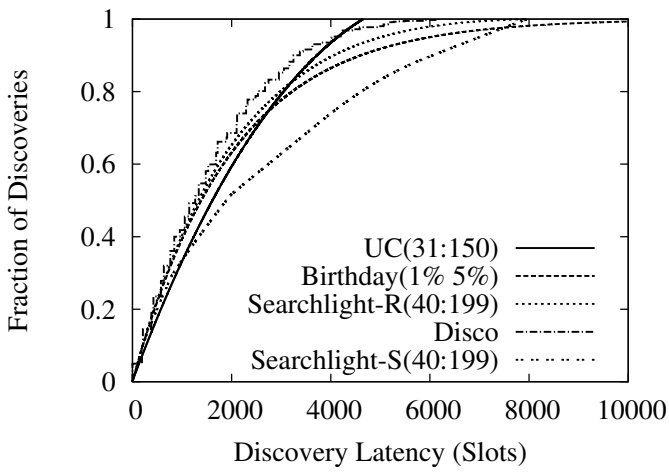


Fig. 8. 1%-5%

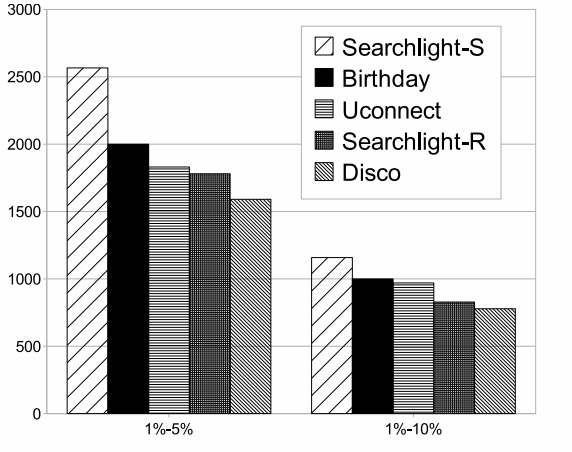


Fig. 9. state-based simulation: Average Latency

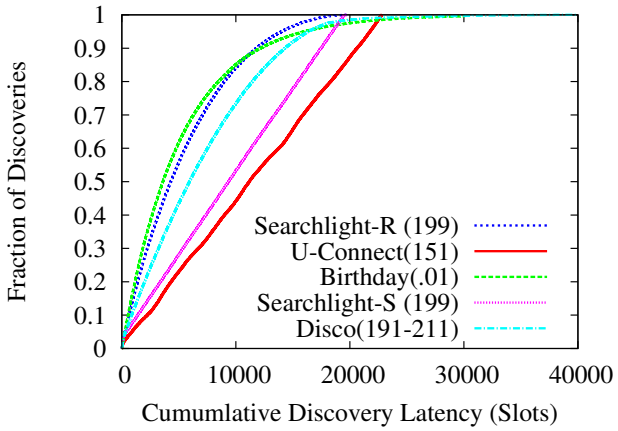


Fig. 10. ns2 simulation: Cumulative Latency 1% duty cycle

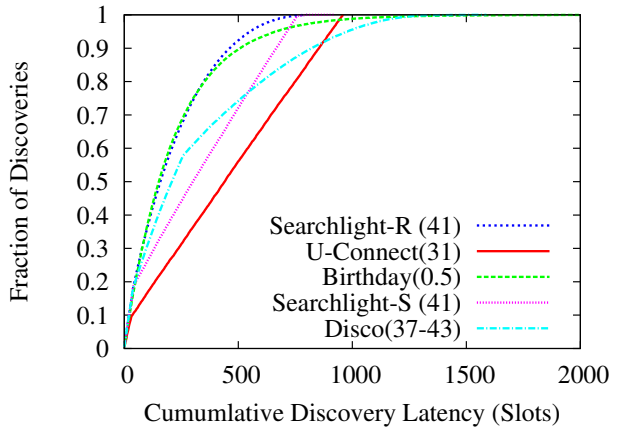


Fig. 11. ns2 simulation: Cumulative Latency 5% duty cycle

latency metric. In addition, the results show that Searchlight, specifically Searchlight-R, performs best on average as well.

2) Asymmetric case: The evaluation of the asymmetric case is particularly important because the worst-case latency is different for Searchlight in this scenario. As opposed to the $t \cdot \frac{t}{2}$ bound for the symmetric case, the worst-case bound in the asymmetric case is $t_1 \cdot t_2$ where t_1 and t_2 are the two different period lengths (t_1 and t_2 must be relatively prime). While this is the same as the bound for U-Connect, it must be noted that U-Connect uses smaller primes for the same target duty-cycle. In other words, for a particular pair of duty cycles, U-Connect has lower worst-case bound in the asymmetric case. This makes it interesting to evaluate if worse worst-case latency for Searchlight leads to worse average case performance.

When two nodes operate at 1% and 10% duty cycles respectively (see Figure 7), Searchlight-R performs better than U-Connect up to the 90-th percentile but has a longer tail from that point onwards. The Birthday protocol performs worse than Searchlight-R all along and also has a much longer tail. For the asymmetric scenario when the two duty cycles are 1% and 5% (see Figure 8), Disco performs best in almost all scenarios. Searchlight-R performs slightly worse than Disco but better than U-Connect up to the 80-th percentile, but then moves to a long tail.

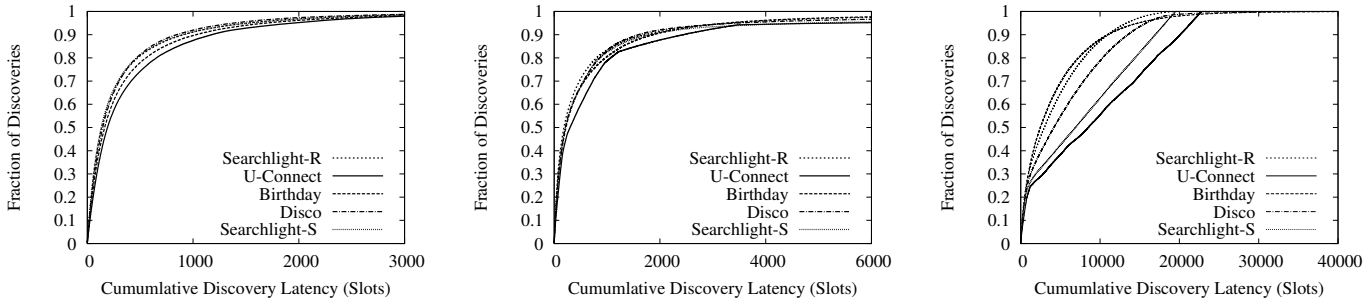
To further investigate this issue, we look at at average latency for both the scenarios (see Figure 9). In both cases, Searchlight-R performs slightly worse than Disco but outperforms all other protocols. These results show that while the worst-case latency is higher, the average case performance of Searchlight-R is comparable to the best performing protocol in the asymmetric case.

B. Simulator-based Simulation

Exhaustively simulating all possible state combinations is not feasible for more than two nodes. Additionally, the simpler state-based simulations assume that nodes agree on the slot boundary. Because of these limitations and to take interference into account, we used the *ns-2* simulator [2] to simulate the performance of the protocols in a 10-node network where all nodes are in transmission range of each other (i.e., they form a clique).

1) Symmetric case: Figures 10 and 11 show the performance of different protocols when the nodes operate at 1% and 5% duty cycles respectively. For both Searchlight-S and U-Connect, the cumulative distribution function is linear and Searchlight-S has a higher slope. On the other hand, both Searchlight-R and Birthday perform pretty similar for the majority of the cases (up to the 90-th percentile) but then the performance of Birthday starts to drop off considerably and ends with a long tail. While these trends are similar to what we observed in the state-based simulation, a significant difference is the relative performance of Disco. In Figure 5, Disco does well only initially but fares worse than all protocols after the 60-th percentile. But in ns-2 simulation, Disco does better than Searchlight-S and U-Connect for a much bigger fraction of the cases. This happens because with 10 nodes, more probable scenarios dominate the performance. Among the deterministic protocols, the worst-case discovery latency is largest for Disco. But in a 10 node network, there are 45 different pairs of nodes and the relative phase offsets that result in larger discovery latencies occur less frequently and hence have less impact on overall performance.

2) Asymmetric case: For the asymmetric case, we simulated three different scenarios. In the first case, all nodes operate at different duty cycles (1, 2, 3, ...,10). This is an instance of a completely asymmetric scenario since no two nodes have the same duty cycle. As we have discussed before, the worst-case latency for Searchlight is worse in the asymmetric case than in the symmetric case. But in the simulations, not surprisingly, the worst-case occurs rarely and the average case dominates. Since average case performance of both versions of Searchlight is good even in asymmetric cases, we see Searchlight doing better than both U-Connect and Birthday in terms of CDF of discovery latencies (see Figure 12(a) while marginally lagging behind Disco. The reason behind Disco's better performance is its use of two primes, where one prime can be much smaller than the other



(a) 1-2-3-4..10 (b) 1-1-1-5..10 (c) 1-1-1-1..10

Fig. 12. ns2 simulation: Asymmetric

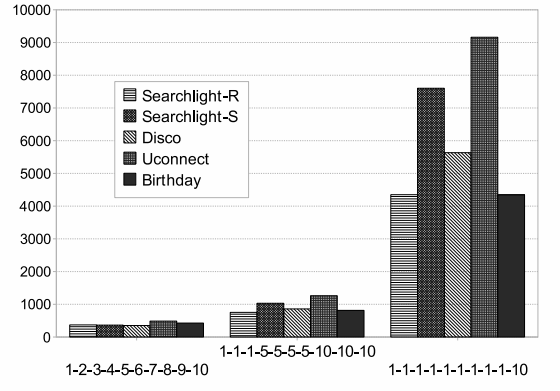


Fig. 13. ns2 simulation: Average Latency (Asymmetric)

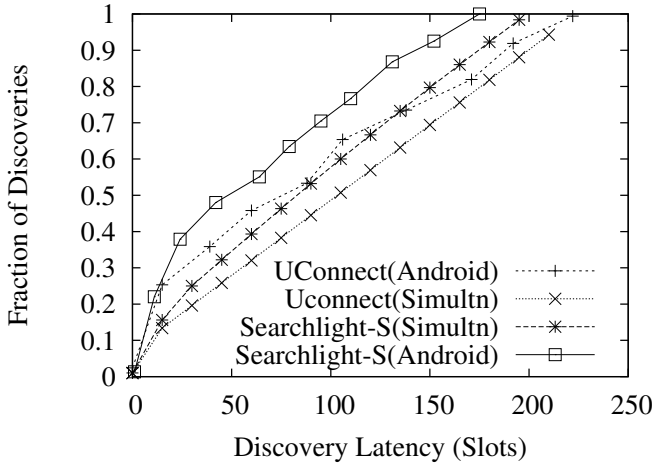


Fig. 14. Comparison of Implementation and Simulation

one. For example, to operate at 8% duty cycle, the only suitable prime pair for Disco is (19,37) while for 9%, it is (17,31). So for a node pair where one operates at 8% duty cycle and other at 9%, the worst case latency for Disco is $17 \cdot 19 = 323$ slots. Thus, in the asymmetric case, the product of the two smaller primes dominate, resulting in better performance. But this performance gain in the asymmetric case results in much worse performance for Disco in the symmetric case. For example, for two nodes operating at 8% duty cycle, the worst case latency for Disco is $37 \cdot 19 = 703$ slots. For the same duty cycle, the worst-case latency for Searchlight and U-Connect is 300 and 361 slots respectively.

In the second scenario, three nodes operate at 1% duty cycle, 4 nodes at 5% and the the remaining three at 10%. This is not a completely asymmetric scenario since each node has at least two other nodes operating at the same duty cycle. The presence of symmetric cases boost the performance of Searchlight-R and its performance matches that of Disco in the CDF graph while outperforming other protocols (see Figure 12(b)).

In the third scenario, 9 nodes operate at 1% duty cycle while the remaining node operate at 10% duty cycle. This is almost a completely symmetric scenario with only 9 pairs operating asymmetrically. Hence, Searchlight-R, which excels in the symmetric case, outperforms all other protocols convincingly in terms of discovery latency (see Figure 12(c)).

We also look at the average latency for all three scenarios to get a better understanding of the relative performance of the protocols (see Figure 13). In the completely asymmetric case, Disco and the two versions of Searchlight perform similarly, but Disco marginally does better. However, when the scenario starts including symmetric cases, Disco and Searchlight-S start falling behind and Searchlight-R achieves the lowest average latency. U-Connect, on the other hand, has the worst average case latency in all the three scenarios.

From the results of both kinds of simulations, it is evident that Searchlight-R is the best protocol for the symmetric case in terms of both median and worst-case behavior. For the asymmetric case, Searchlight-R has worse worst-case latency bound, but makes up for it by performing better in the average case. Searchlight-S, on the other hand, lags behind Searchlight-R in all scenarios. This clearly demonstrates the benefit of adopting the randomization-based approach to determining probe slot schedules.

V. Implementation

One of the main objectives for designing asynchronous neighbor discovery protocols is to facilitate ad hoc communication between handheld devices like smartphones. To

gauge how Searchlight achieves this goal in practice, we implemented the protocol on HTC Dream G1, a smartphone by HTC that supports the open source Google Android mobile device platform [1]. We first describe different implementation issues and then present some preliminary results.

A. Implementation Issues

1) Slot Duration: We implemented Searchlight to use the Wifi radio of the G1 phone for neighbor discovery. Earlier protocols for asynchronously discovering neighbors were all implemented on sensor nodes, allowing them to use small slots in the order of milliseconds [3] or even microseconds [4]. However, unlike sensor radios (e.g., CC2420), Wifi radios have a non-negligible transition latency from sleep to transmit/receive. On the G1 phone, we found out that from the application level, the time to bring the wireless interface up is around 5 to 6 seconds, Because of this latency, we decided on a slot size of 7 seconds.

2) pre slots: Because of the non-negligible start-up time, we introduced the notion of pre slots. A pre slot basically precedes any active slot and switches on the interface. For example, assume that a node needs to be active during slot 10. Now, if the command to wake up the radio is issued at the beginning of slot 10, it will take almost the whole slot duration for that command to return and the effective awake time during that slot will be reduced to 1-2 seconds. To get around this problem, the wake up command now gets issued at the start of the preceding slot which is slot 9 in this case. Such slots are called pre slots and their positions are determined based on the active slot schedule.

3) Active slot: When the protocol starts up, it creates an active slot schedule based on the given t. In an active slot, a hello message containing node id gets sent at the very beginning and at the very end. In between, the node continuously listens for hello messages from other nodes. When it gets a hello message, it adds the name of the sender to a friend list, if that id was not already there. Initially, a node randomly chooses a slot index to start from where the range of the slot index is $[0,t \cdot \lfloor \frac{t}{2} \rfloor]$. We also added a "Pause and Reset" button which allows a user to restart the discovery process by again choosing a random starting slot.

B. Evaluation

We implemented Searchlight-S and U-Connect on five G1 phones and logged the discovery latency for around 250 runs with 10% duty cycle. We compared the values with our simulation results (see Figure 14). The implementation results actually turned out to be better than simulation results, with more discoveries taking place at lower latencies. The overall trends for both implementations were linear, which agree with the simulation results. Between the two implementations, Searchlight-S fares much better than U-Connect.

VI. Future Work

Solving the problem of energy efficient asynchronous neighbor discovery is an important pre-condition for more widespread use of ad hoc communication between mobile devices, including sensors and smartphones. In this paper, we present Searchlight, a new asynchronous neighbor discovery protocol that addresses this problem by adopting a systematic probing based approach to provide better bounds on discovery latency than any existing protocol when nodes have similar energy requirements. Extensive simulation results show that Searchlight achieves the best average case discovery latency in most of the scenarios and performs on par with other protocols in the remaining cases. Searchlight was also successfully implemented on a smartphone testbed, and showed performance trends similar to the simulation results.

In the future, we intend to extend the implementation of Searchlight on other mobile platforms including sensor nodes. Also, we would like to investigate how the protocol can dynamically adapt to energy requirements, contact patterns and other factors to adjust its duty cycle. For example, when neighbor count is low, a higher duty cycle might be required to find enough neighbors that meet application requirements. On the other hand, in a crowded place where number of co-located nodes is high, a smaller duty cycle might suffice to find a reasonable number of neighbors.

References

[1] Android - an open handset alliance project. http://developer.android.com.

[2] ns2 network simulator. http://www.isi.edu/nsnam/ns/.

[3] Prabal Dutta and David Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 71–84, New York, NY, USA, 2008. ACM.

[4] Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan (Raj) Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *IPSN '10: International Conference on Information Processing in Sensor Networks*, pages 350–361, 2010.

[5] Shouwen Lai, Binoy Ravindran, and Hyeonjoong Cho. Heterogenous quorum-based wakeup scheduling in wireless sensor networks. *IEEE Transactions on Computers*, 99(PrePrints), 2010.

[6] Ting Liu, Christopher M. Sadler, Pei Zhang, and Margaret Martonosi. Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet. In *MobiSys 2004*, pages 256–269. ACM, 2004.

[7] M. Matuszewski, N. Beijar, J. Lehtinen, and T. Hyyrylainen. Understanding attitudes towards mobile peer-to-peer content sharing services. In *Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on*, pages 1–5, 25-29 2007.

[8] Michael J. McGlynn and Steven A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 137–145, New York, NY, USA, 2001. ACM.

[9] Ivan Niven and Herbert S. Zuckerman. *An Introduction to the Theory of Numbers*. John Wiley and Sons (WIE), 1991.

[10] Anna K. Pietiläinen, Earl Oliver, Jason Lebrun, George Varghese, and Christophe Diot. MobiClique: middleware for mobile social networking. In *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, pages 49–54. ACM, August 2009.

[11] 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.

[12] Lan Zhang, Xuan Ding, Zhiguo Wan, Ming Gu, and Xiang-Yang Li. Wiface: a secure geosocial networking system using wifi-based multi-hop manet. In *MCS '10: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services*, pages 1–8. ACM, 2010.

[13] Rong Zheng, Jennifer C. Hou, and Lui Sha. Asynchronous wakeup for ad hoc networks. In *MobiHoc 2003*, pages 35–45, New York, NY, USA, 2003. ACM.