# Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications

Prabal Dutta
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
prabal@cs.berkeley.edu

David Culler
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
culler@cs.berkeley.edu

## ABSTRACT

We present Disco, an asynchronous neighbor discovery and rendezvous protocol that allows two or more nodes to operate their radios at low duty cycles (e.g. 1%) and yet still discover and communicate with one another during infrequent, opportunistic encounters without requiring any prior synchronization information. The key challenge is to operate the radio at a low duty cycle but still ensure that discovery is fast, reliable, and predictable over a range of operating conditions. Disco nodes pick a pair of prime numbers such that the sum of their reciprocals is equal to the desired radio duty cycle. Each node increments a local counter with a globally-fixed period. If a node's local counter value is divisible by either of its primes, then the node turns on its radio for one period. This protocol ensures that two nodes will have some overlapping radio on-time within a bounded number of periods, even if nodes independently set their own duty cycle. Once a neighbor is discovered, and its wakeup schedule known, rendezvous is just a matter of being awake during the neighbor's next wakeup period, for synchronous rendezvous, or during an overlapping wake period, for asynchronous rendezvous.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design—*Wireless communication*; C.2.2 [**Computer-Communications Networks**]: Network Protocols

## General Terms

Algorithms, Design, Experimentation, Performance

## Keywords

Protocol, Neighbor Discovery, Rendezvous, Wireless


*SenSys'08,* November 5–7, 2008, Raleigh, North Carolina, USA.


## 1. INTRODUCTION

The interaction of things – energy-constrained mobile objects with other mobile and static objects – provides a fertile ground for application-driven research. However, the low-power, asynchronous neighbor discovery problem poses one challenge for these applications: how can two systems that are awake infrequently and perhaps rarely co-located discover each other without any prior knowledge of their potential encounters, and without external assistance? The key issue is that nodes must operate their radios at low duty cycles to maximize lifetime, and yet be actively vigilant to detect the emergence of new links and the attrition of old ones. These two requirements – low-power operation and active vigilance – are at odds with each other since optimizing for one may come at the expense of the other.

This paper presents *Disco*, a practical solution to the asynchronous neighbor discovery and rendezvous problem that works by scheduling radio wake times at multiples of prime numbers, ensuring deterministic pairwise discovery and rendezvous latencies without requiring global coordination of duty cycles or superframe structure. The algorithm selects a pair of prime numbers such that the sum of their reciprocals is equal to an application's desired radio duty cycle. Each node increments a local counter with a globally agreed-upon period and, if this local counter is divisible by either of the primes, the node turns on its radio for one counter period. This protocol, a simple adaptation of Sun Zi's two-millenia old Chinese Remainder Theorem [18], ensures discovery in bounded time, even if nodes independently set their own duty cycle. Section 3 presents the Disco design starting with a simplified version of the algorithm to show correctness, and then relaxes the simplifying assumptions to flesh out a protocol that works in practice.

To use Disco, an application first chooses a desired duty cycle or discovery latency, and optionally a node class (nodes in the same class do not need to minimize discovery latency between themselves). Disco automatically selects primes that match the desired duty cycle or discovery latency and then turns on the radio at every multiple of the chosen primes. During each such wakeup, a node can listen, beacon, or do both, depending on application requirements. Section 4 presents the details of our Disco implementation.

Disco performs well on key performance metrics like discovery latency and rendezvous frequency as a function of duty cycle. Disco also offers great flexibility for applications: nodes can independently select their own duty cycles and still ensure discovery, or nodes can be assigned to different classes such that inter-class discovery times are guar-

anteed to be much faster than without class assignments, or nodes can adjust duty cycles to achieve a particular discovery latency, or nodes can choose to beacon, listen, or do both, during their wake times. Section 5 compares Disco's performance with earlier work and sensitivity to several parameters through an in-depth simulation study. Section 6 presents empirical data collected from our implementation.

The flexibility afforded by Disco is motivated by the different needs, duty cycles, and interaction patterns of emerging applications. We identify three common patterns in mobile-to-mobile or mobile-to-static interactions that we call *talking*, *docking*, and *flocking*, and briefly discuss them to set the context for this work.

In the *talking* pattern, two mobile nodes encounter each other, communicate, and part ways. Such encounters can provide insight into many real-world social network effects [4]. Tracking social interactions can help epidemiologists study the spread of disease in schools or sociologists better understand children's social development patterns. In the workplace, many interactions occur face-to-face and outside the purview of computing: water cooler conversations pass along important gossip, many executives and managers "manage by walking around," and individuals spread information epidemically. And, we are all familiar with detecting zebra-to-zebra encounters [14], tracking networking researcher interactions [12], and logging hiker sightings after their trail-side encounters with other hikers [10].

In the *docking* pattern, a mobile node discovers a static node situated at a rendezvous point. For example, a company might want to ensure that a guard is doing his rounds by checking the encounter logs in all corners of an office. Other applications include uploading cargo transit history at readers [15], reporting cattle movement data during feeding times at the trough [28], tracking hikers via their encounters with trail-side waypoints [10], and tracking researchers' whereabouts in the lab using Active Badges [27].

In the *flocking* pattern, a group of nodes move together as a unit. For example, an elementary school teacher might want to know if all the children are on the bus before it departs from the zoo and, if a child is missing, when that child was last seen and by whom. A railroad might want to determine the manifest of a train en route or the order of its constituent rail cars, especially as it switches cars in and out at sidings and rail yards. A particularly forgetful person might want to ensure that his personal items travel together and be notified if he leaves one behind [1]. Or, one node in a constellation of sensors monitoring a Parkinson's patient might want to inform the other sensors if it discovers an access point where data can be uploaded [19].

## 2. RELATED WORK

For wall-powered or rechargeable nodes, the asynchronous neighbor discovery problem has a simple solution: a node periodically beacons its presence and any always-on neighbor that receives the beacon considers the first node to be a neighbor. The problem is also simple with some external assistance: a node attempts to discover and join a network just after being powered on or reset by a human operator or just after being initially deployed [17]. Discovery is also aided greatly by external synchronization: a node only beacons and listens for neighbors for just a brief period after each minute, quarter-hour, or hour, for example, if nodes can synchronize their clocks using GPS [14]. The problem is also simple if the nodes are deployed in a static network and expect to have one or more neighbors at all times: nodes maintain time synchronization [25], send packets with long preambles [20], or repeatedly send the same packet until it is acknowledged [2].

These simple techniques work because encounters are predictable: a sender has a reasonable expectation that the receiver is nearby, that its duty cycle is known, and that it will be awake soon. The discovery problem becomes more challenging in energy-constrained, mobile environments since a node may not know whether any neighbors are present, and what duty cycles those neighbors might operate at, given the widely varying energy availability and usage observed in practice for both mobile [23] *and* static nodes [24]. Since idle listening often dominates the system power budget [5], the most expedient – perhaps only – way to balance the power supply and demand is to reduce the listen duty cycle. Another reason asymmetric duty cycles are useful is that they allow nodes with different roles or capabilities, like cattle collars and static data sinks [28], to interact.

Once listen periods must be adjusted to reflect the available energy or differing workloads, sampling protocols that employ low-power listening and assume a fixed listen period, like B-MAC [20] and X-MAC [2], become less appealing for neighbor discovery. This occurs because the required preamble length to ensure discovery is no longer a network-wide constant. Likewise, slotted protocols like S-MAC [29] that periodically listen for a whole synchronization period to discover neighbors [13] assume global agreement on the length of this period. Although Disco requires global agreement on the period of a counter, which is comparable to a single communications slot in slotted protocols, this dependence does not preclude Disco nodes from independently choosing their own duty cycles.

Prior work in asynchronous neighbor discovery has employed stochastic, quorum, and combinatorial techniques. McGlynn and Borbash proposed "birthday protocols" for asynchronous neighbor discovery in static ad hoc networks [17]. They considered the problems of energy conservation during node deployment and energy-efficient neighbor discovery following deployment using a scheme in which nodes listen, transmit, or sleep with different probabilities. Their work concludes that mobile ad hoc networks would not use discovery, but that discovery would be useful in static ad hoc networks. Even in static networks, however, they schedule an explicit discovery phase that quickly terminates. Our proposal differs in a few key ways. First, we note that discovery can be quite valuable in mobile networks when the nodes move slowly or have modest dwell times near peers. Second, we suggest that discovery should be a fundamental and continuous service in both mobile *and* static networks, rather than a one-time event. Finally, we note that probabilistic discovery leads to aperiodic and unpredictable rendezvous latencies, and long tails on discovery probabilities, reducing its appeal.

Tseng et al. propose a quorum-based protocol for multihop ad hoc networks [26]. Their protocol divides time into a sequence of beacon intervals which are grouped into sets of $m^2$ contiguous intervals, where $m$ is a global parameter. In each group, the $m^2$ intervals are arranged as a two-dimensional $m \times m$ array in a row major manner. A node arbitrarily picks one column and one row of entries to transmit and receive, respectively, for a total of $2m - 1$ in-

tervals, in each group of $m^2$ intervals. Since $m$ is a global parameter, all nodes use the same duty cycle, which limits flexibility. Jiang et al. generalize this result to any quorum protocol that satisfies a rotation closure property [11]. We note that both sampling protocols, like B-MAC and X-MAC, and scheduled protocols, like S-MAC, implicitly use quorum-based neighbor discovery. Their use differs only in the details of what happens during each interval and whether transmissions are row major and listening is column major in each group of $m^2$ intervals as is the case for sampling protocols, or the reverse for slotted ones.

Zheng et al. apply optimal block designs using difference sets to the problem of asynchronous neighbor discovery [30]. Their solution addresses the symmetric duty cycle problem, when all node duty cycles are uniform throughout the network. They conclude that for asymmetric duty cycles, their approach reduces to an NP-complete minimum vertex cover problem requiring a centralized solution. These limitations are at odds with the requirements of our problem.

Herman et al. explore the temporal partition problem in sensornets where two or more groups of nodes with differing schedules become unaware of each other [8]. The paper presents several self-stabilizing protocols to solve the problem of temporal partition; starting from an arbitrary temporally partitioned state, these protocols lead the network to a state in which all nodes have aligned sleep schedules. Their approach uses randomly chosen relatively prime sleep periods and occasional, and possibly random, probing of extra time slots. This approach constrains a node to only two duty cycle choices and the paper further states that since deterministically guaranteeing that two groups make distinct choices is difficult, the protocol resorts to randomization. Disco addresses a more general set of neighbor discovery problems and avoids the need for a randomized protocol by using *pairs* of primes.

# 3. DESIGN

This section presents the design of the Disco neighbor discovery and rendezvous protocol. Neighbor discovery (or rediscovery) allows two nodes with independent duty cycles and no prior (or current) synchronization information to discover each other in bounded time when the nodes are within radio range of each other. Rendezvous allows nodes to deliver messages to previously discovered neighbors with predictable and controllable latencies. We begin with a simplified version of the Disco algorithm that makes proving Disco's correctness straightforward. We then relax the simplifying assumptions to flesh out a protocol that works in practice.

## 3.1 Simplified Algorithm

Discovery is the process by which nodes learn about their current one-hop neighbors. The idea behind the discovery algorithm is simple. Two nodes, $i$ and $j$, pick two numbers, $m_i$ and $m_j$, such that $m_i$ and $m_j$ are relatively prime (coprimes) and $1/m_i$ and $1/m_j$ are approximately equal to $i$ and $j$'s desired duty cycles, respectively. Time is divided into fixed-width *reference periods* and consecutive periods are labeled with consecutive integers. Nodes $i$ and $j$ start counting the passage of these periods at times $a_i$ and $a_j$, with their respective counters, $c_i$ and $c_j$, initialized to zero, and with $i$ and $j$ counts synchronized to the reference period (we will relax this last assumption in later sections). If $c_i|m_i$ ($c_i$ is divisible by $m_i$), then $i$ turns on its radio for one period and beacons (or listens, or does both, depending on application requirements). Similarly, if $c_j|m_j$, then $j$ turns on its radio for one period and beacons. When both $i$ and $j$ turn on their radios during the same period, they can exchange beacons and discover each other.

It is easy to see that there is exactly one such overlapping period every $m = m_i m_j$ periods. Letting $x$ represent the reference period number, we have

$$\begin{aligned} c_i &= x - a_i \\ c_j &= x - a_j \end{aligned}$$

Our goal is to find an $x$ such that $c_i|m_i$ and $c_j|m_j$. We can express this as a pair of simultaneous congruences

$$\begin{aligned} x &\equiv a_i \pmod{m_i} \\ x &\equiv a_j \pmod{m_j} \end{aligned}$$

Such a set of congruences are known to have a common solution by the Chinese Remainder Theorem [18]. This theorem states that if $x_0$ is one such solution, then an integer $x$ satisfies the congruences if and only if $x$ is of the form $x = x_0 + km$ for some integer $k$. One $x_0$ is

$$x_0 = a_i b_i m_j + a_j b_j m_i$$

where the solution is unique $\pmod{m}$ for $m = m_i m_j$, and where $b_i$ and $b_j$ must satisfy the congruences

$$\begin{aligned} b_i m_j &\equiv 1 \pmod{m_i} \\ b_j m_i &\equiv 1 \pmod{m_j} \end{aligned}$$

Let us consider a concrete example. Let node $i$ select $m_i = 3$ (so $i$'s duty cycle is: $DC \approx 33\%$), start counting at reference period $x = 1$ (so that $a_i = 1$), with counter values $c_i$. Similarly, let node $j$ select $m_j = 5$ (so $j$'s duty cycle is: $DC \approx 20\%$), start counting at reference period $x = 2$ (so that $a_j = 2$), with counter values $c_j$. Figure 1 illustrates this timeline and counter values. Dark entries in the $c_i$ and $c_j$ rows indicate $c_i|m_i$ and $c_j|m_j$, respectively. Columns where both $c_i$ and $c_j$ are dark indicate values of $x$ for which both $i$ and $j$ have overlapping on slots, and can therefore communicate. In this example, when $x = 7$ and $x = 22$, both $i$ and $j$ are turned on and can discover each other.

We can express this example as the following simultaneous congruences

$$\begin{aligned} x &\equiv 1 \pmod{3} \\ x &\equiv 2 \pmod{5} \end{aligned}$$

and see that when $x = 7$, both congruences are solved

$$\begin{aligned} (1-7)|3 \\ (2-7)|5 \end{aligned}$$

An analytic solution requires finding $b_i$ and $b_j$

$$\begin{aligned} 5b_i &\equiv 1 \pmod{3} \\ 3b_j &\equiv 1 \pmod{5} \end{aligned}$$

We see that values of $b_i = 2$ and $b_j = 2$ satisfy these congruences and hence one solution $x_0$ is

$$\begin{aligned} x_0 &= a_i b_i m_j + a_j b_j m_i \\ x_0 &= 1 \cdot 2 \cdot 5 + 2 \cdot 2 \cdot 3 \\ x_0 &= 22 \end{aligned}$$

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| $c_j$ | - | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

**Figure 1: An example discovery timeline. Two nodes, $i$ and $j$, start their counters, $c_i$ and $c_j$, at times $x = 1$ and $x = 2$, with periods $m_i = 3$ and $m_j = 5$, and duty cycles of approximately 33% and 20%, respectively. The dark cells indicate times when the nodes $i$ and $j$ turn on their radio. Both nodes are awake at times $x = 7$ and $x = 22$. This pattern repeats when $x = 7 + 15k$, for all $k \in \mathbb{Z}^+$.**

Since all solutions are unique $\pmod{15}$, we have

$$x_0 = 22 \pmod{15} = 7$$

which agrees with our solution from Figure 1 and gives $x = 7 + 15k$, for all $k \in \mathbb{Z}^+$.

The preceding analysis sidesteps a number of practical considerations. Since, for example, the Chinese Remainder Theorem requires the moduli $m_i$ and $m_j$ be coprimes to guarantee a solution to the simultaneous congruences, these values cannot be independently chosen by the nodes, which is limiting. We also required that nodes be able to express their desired duty cycle as the reciprocal of a positive integer (e.g. $1, 1/2, 1/3, \ldots, 1/k$, where $k \in \mathbb{Z}^+$), which is restrictive. We assumed that nodes $i$ and $j$ synchronized their counting with the reference phase, which aids analysis but is unlikely to hold in practice. The preceding analysis also fails to explore the effect of clock drift on discovery, ignores radio startup time and energy overhead, and assumes that communications jitter is negligible. In the remainder of this section, we progressively relax these assumptions, and evolve our protocol, to flesh out one that works in practice.

## 3.2 Coprimes are not Enough

The Chinese Remainder Theorem requires the moduli $m_i$ and $m_j$ be coprimes to guarantee a solution to the simultaneous congruences. This restriction raises some challenges. First, the moduli cannot be chosen independently by the nodes since such choices could lead to values of $m_i$ and $m_j$ that are not coprimes. It would, of course, be preferable to let nodes choose the moduli that best satisfy their individual duty cycle requirements rather than require a static or central assignment. Second, restricting the moduli to coprimes is not scalable since there are only a handful of numbers that can satisfy both the target duty cycle (typically 1-5%) and coprime requirement. Third, if $m_i = m_j$, then nodes $i$ and $j$ may never discover each if they wake up with the same period but different phase.

One way to allow each node to select the best duty cycle for itself while still ensuring discovery occurs is to require each node $i$ to pick two primes, $p_{i_1}$ and $p_{i_2}$, such that $p_{i_1} \neq p_{i_2}$ and the sum of their reciprocals (approximately) equals the desired duty cycle

$$DC \approx \frac{1}{p_{i_1}} + \frac{1}{p_{i_2}}$$

Each node increments a local counter and if a node's local counter is divisible by either of its primes, the node turns on its radio for a single interval, whose length is the only global parameter. This approach ensures that no matter what duty cycles are independently selected at different nodes, for every pair of nodes $i$ and $j$, there will be at least one pair in the set $\{(p_{i_1}, p_{j_1}), (p_{i_1}, p_{j_2}), (p_{i_2}, p_{j_1}), (p_{i_2}, p_{j_2})\}$ that are relatively prime, satisfying the Chinese Remainder Theorem.

Note, however, that simply requiring $p_{i_1}$ and $p_{i_2}$ to be coprimes does not satisfy the requirements of the Theorem, and therefore cannot ensure discovery. For example, letting $(p_{i_1}, p_{i_2}) = (30, 77)$ and $(p_{j_1}, p_{j_2}) = (35, 66)$, ensures that intra-node pairs are coprime

$$\begin{aligned} \gcd(p_{i_1}, p_{i_2}) = \gcd(30, 77) &= 1 \\ \gcd(p_{j_1}, p_{j_2}) = \gcd(35, 66) &= 1 \end{aligned}$$

however, the inter-node pairs are not

$$\begin{aligned} \gcd(p_{i_1}, p_{j_1}) = \gcd(30, 35) &= 5 \\ \gcd(p_{i_1}, p_{j_2}) = \gcd(30, 66) &= 6 \\ \gcd(p_{i_2}, p_{j_1}) = \gcd(77, 35) &= 7 \\ \gcd(p_{i_2}, p_{j_2}) = \gcd(77, 66) &= 11 \end{aligned}$$

which means discovery may fail. Consider the case in which node $i$ starts counting at time $x = 0$, so node $i$'s radio is on at times $x = 30k$ and $x = 77k$, for all $k \in \mathbb{Z}^+$, and node $j$ starts counting at time $x = 1$, so node $j$'s radio is on at times $x = 35k + 1$ and $x = 66k + 1$, for all $k \in \mathbb{Z}^+$. There is no $x$ for which both $i$ and $j$ have their radios turned on simultaneously, ensuring discovery will fail. Therefore, to ensure correctness, Disco uses prime pairs rather than coprimes.

## 3.3 Choosing Primes

The choice of primes can have a large impact on discovery latency. For example, a target duty cycle of 2% can be achieved in several ways. One combination of primes that achieves this duty cycle is 97 and 103 ($1/97 + 1/103 = 2\%$) but another combination is 53 and 883 ($1/53 + 1/883 = 2\%$). Which combination is better? Assume that both nodes picked the same pair of primes. In that case, with high probability, the worst-case discovery latency will be $97 \times 103 = 9,991$ periods vs $53 \times 883 = 46,799$ periods, more than a factor of four difference. Now, assume that one node picked 53 and 883 while the other node picked 57 and 409. In this case, the worst-case discovery latency becomes $53 \times 57 = 3,201$, *fifteen times faster* than $53 \times 883 = 46,799$.

The ratio of the worst-case discovery latency between an auspicious and an unfortunate set of prime pairs is bounded by the duty cycle. For example, a 10% duty cycle results in no worse than a 1:10 ratio, a 2% duty cycle is bounded by a 1:50 ratio, and a 1% duty cycle results in at worse a 1:100 ratio. If we let $c = 1/DC$, we have

$$\frac{1}{c} \approx \frac{1}{p_{i_1}} + \frac{1}{p_{i_2}} \tag{1}$$

rearranging and solving for $p_{i_2}$, we have

$$p_{i_2} \approx \frac{p_{i_1} c}{p_{i_1} - c} \tag{2}$$

The limit of the ratio between the auspicious and unfortunate worst-case latencies is

$$\lim_{p_{i_1} \to c+1} \frac{p_{i_1}^2}{p_{i_1} p_{i_2}} = \frac{(c+1)^2(c+1-c)}{(c+1)(c+1)c} = \frac{1}{c} = DC \quad (3)$$

These observations suggest that picking the prime pairs requires care: a good choice can result in low discovery latency but a poor choice can result in much longer worst-case discovery latency. Low discovery times are possible if one of the primes is very close to the reciprocal of the duty cycle while the other prime is a much larger number. If this approach is taken, then it becomes important to randomize the choice of prime pairs to reduce the chance that two nodes will have picked the same pair if they both select the same duty cycle.

If nodes can be assigned to different classes such that members of a class do not need to discover or communicate with each other, then it is easy to ensure good pairs are selected. For each possible duty cycle, every node running Disco employs a deterministic algorithm to generate an ordered list of prime pairs that can satisfy the duty cycle. A prime pair's position in the ordered list, taken modulo the number of distinct classes defined by the application, dictates the particular class to which a pair is assigned. A node chooses at random one of the prime pairs assigned to its class. This algorithm ensures that nodes in different classes are assigned distinct pairs, which, as we will show later can greatly improve discovery latency. The policy of class label assignment is left to the application, but the mechanism to ensure good inter-class pairs are chosen is handled by Disco.

## 3.4 Slot Non-Alignment

We now relax the assumption that slots are aligned and delve into the details of slot construction. In practice, slots will rarely be aligned since nodes are run independently and do not adjust clock skews or set up a global time reference. Since we now assume slots are not aligned, we need to ensure that two nodes will still discover each other regardless of how their slots overlap. *To maximize the likelihood that overlapping slots result in discovery, Disco transmits a beacon at both the beginning and end of a slot when beaconing.*

Even if slots are generally non-aligned, nodes that are in-phase may come into contact with each other from time to time. When this happens, both nodes may attempt to transmit their beacons at the same time, causing collisions or receiving each others beacons during every slot. Although this situation can occur, Disco leaves it to the application to decide how to respond at a coarse grain by, for example, changing the duty cycle or changing to a listen-only mode. Part of the reason to leave this to the application is that if nodes have even small variations in the clocks, nodes are likely to fall out of phase naturally. One API design question is whether there should be explicit support for letting the application shift phase to explicitly deal with synchronization?

An important question is what a node should do if the channel is busy. There are a handful of options. A node may blindly transmit regardless of channel contention, it could enter a channel contention phase, it could forgo transmission altogether, or it could wait until the channel is clear and then transmit. Blindly transmitting when the channel is busy is hardly scalable and would lead to channel contention from colliding beacons. Entering channel contention could introduce a long delay, effectively throwing off the timing of the slot. Forgoing transmission also throws off the timing of the slot. Disco currently transmits as soon as the channel is clear, provided it does not expect any of its neighbors to do so.

## 3.5 Duty Cycle from Discovery Latency

In many applications, it will be necessary to compute the duty cycle or beacon rate required to satisfy a particular discovery latency. The application will specify the worst-case discovery latency tolerable and the minimum duty cycle will need to be computed. The procedure to convert maximum discovery latency, $t_{disco}$, to duty cycle is relatively simple. Two nodes operating with primes $p_{i_1}$ and $p_{j_1}$ will discover each other in at most $p_{i_1} p_{j_1}$ counter periods, where each counter period is of length $t_{slot}$. For discovery to occur in the required time, the following inequality must hold

$$p_{i_1} p_{j_1} t_{slot} \leq t_{disco}$$

Without loss of generality, we assume the primes are equal so that $p = p_{i_1} = p_{j_1}$, giving us the following constraint for choosing the primes

$$p \leq \sqrt{\frac{t_{disco}}{t_{slot}}}$$

Recall, however, that Disco requires a *pair* of primes to ensure discovery when duty cycles are independently chosen. Therefore the minimum duty cycle, $DC$, must satisfy the following inequality

$$DC \geq \frac{1}{p} + \frac{1}{p} = \frac{2}{p}$$

Since each prime $p$ results in a beacon slot every $p$ slots, the minimum required beacon rate is given by

$$f_{beacon} \geq \frac{2}{p \cdot t_{slot}} = \frac{2}{\sqrt{t_{disco} t_{slot}}} \text{ Hz}$$

Note that although the beacon rate *increases* with smaller $t_{slot}$ values, the effective duty cycle *decreases*

$$DC \geq \frac{2}{p} = 2\sqrt{\frac{t_{slot}}{t_{disco}}}$$

## 3.6 Duty Cycle Granularity

A side effect of allowing only those duty cycles that can be expressed as the sum of the reciprocal of two primes is that many large duty cycles cannot be specified with fine granularity. For example, the only legal duty cycles between 57.6% and 100% are shown in Table 1.

| $p_{i_1}$ | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|
| $p_{i_2}$ | 0 | 3 | 5 | 7 | 9 | 11 | 4 | 13 |
| DC (%) | 100 | 83.3 | 70 | 64.3 | 61.1 | 59 | 58.3 | 57.6 |

**Table 1: Legal duty cycles (DC) between 57.6% and 100%. Many duty cycles cannot be realized and the distribution of duty cycles is not uniform.**

To allow a more flexible and fine-grained assignment of duty cycles, Disco supports a third parameter, $p_{i_3}$ that can assume any prime number. With this addition, the duty cycle becomes

$$DC \approx \frac{1}{p_{i_1}} + \frac{1}{p_{i_2}} + \frac{1}{p_{i_3}} \quad (4)$$

# 4. IMPLEMENTATION

To evaluate the feasibility and performance of our design, we implemented Disco using the nesC programming language [6], TinyOS operating system [9], and Telos wireless sensor node [21]. Figure 2 shows the radio on time, beacon transmissions, and current draw profile during a 25 ms slot. We experimented with a range of $t_{slot}$ values and found that discovery performance degrades when $t_{slot} < 5$ ms due to the jitter introduced by the TinyOS timer library and radio stack. Therefore, we use $t_{slot} = 10$ ms in the rest of this paper.

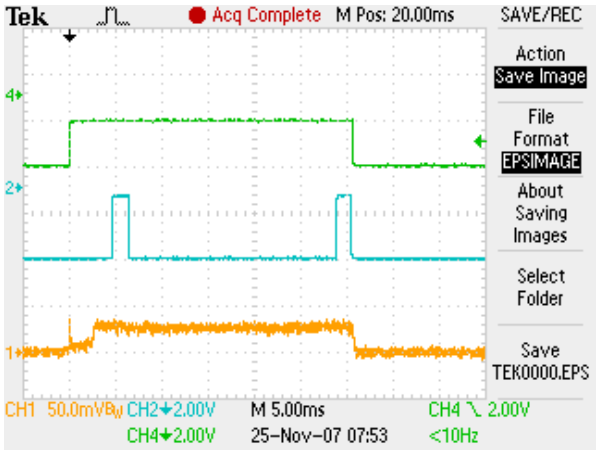


**Figure 2: Beacon slot details ($t_{slot} = 25$ ms). The green line (top) indicates the on-time envelope of the radio. The blue line (middle) shows beacon transmissions at the beginning and end of the slot, and the orange line (bottom) shows the current draw in mA (not mV) using a 1 Ω sense resistor in series with the power supply.**

The `Discovery` interface, shown in Figure 3, allows an application to control discovery parameters, policy, and traffic tunneling. A duty cycle between 0 and 100% can be requested and the service will indicate the duty cycle actually used. The worst-case discovery latency can be limited by assigning different nodes to different classes. The application can control the beaconing policy and piggy-back packets over the discovery channel.

```
interface Discovery {
  // Request a duty cycle between 0 and 100 percent
  command uint8_t setDutyCycle(uint8_t dutycycle);
  command uint8_t getDutyCycle();

  // Set the node class to reduce inter-class latency
  command error_t setNodeClass(uint8_t classid);
  command uint_t  getNodeClass();

  // Select beacon-and-listen or listen-only mode
  command error_t setBeaconMode(bool beacon);
  command bool    getBeaconMode();

  // Request, event, callback for app-specific payload
  command error_t requestBroadcast();
  event error_t   fetchPayload(void *buf, uint8_t *len);
  event message_t received(message_t* msg, void* buf,
    uint8_t len);
}
```

**Figure 3: The discovery programming interface.**

# 5. SIMULATION STUDY

In this section, we evaluate the performance of Disco with earlier work through simulation and we study the relationship between slot length, beacon rate, discovery latency, discovery rate, and duty cycle. In particular, we evaluate the sensitivity of the protocol to the choice of primes.

We use the term *balanced primes* to refer to the case in which the intra-node primes are approximately equal (e.g. 37 and 43). The term *unbalanced primes* refers to the case in which the intra-node primes are significantly different (e.g. 23 and 157). We use the term *symmetric pairs* to refer to the case in which both nodes choose the *identical* pair of primes. The term *asymmetric pairs* refers to the case in which both nodes choose a *different* pair of primes.

## 5.1 Simulation Models

Table 2 shows the three most closely related asynchronous neighbor discovery services. We developed the closed form expression of latency CDFs to speed up simulation runs but also verified their output against a random set of brute force simulations. Note that even though Disco is compared against both Birthday and Quorum in this section, neither of the two actually meets application needs. Birthday is based on a randomized algorithm that does not provide predictable rendezvous times and exhibits a long tail for discovery while Quorum specifies a global constant that all nodes use for their duty cycle.

## 5.2 Discovery Latency Comparison

Discovery latency refers to the delay between the moment two nodes are within communications range to the moment when they first discover each other. The distribution of discovery latencies, as well as the tail of this distribution (*i.e.* worst-case discovery latency), are both important metrics for a neighbor discovery protocol. The worst-case discovery latency determines the minimum amount of time two nodes need to be in communications range to ensure discovery. The distribution provides insight into the average case, or median, behavior.

In this section, we compare the discovery latency of the Disco approach with those of the probabilistic [17] and quorum [26] approaches. We do not compare Disco with the combinatoric approach that uses difference sets because that approach addresses the symmetric problem, when node duty cycles are the same, but concludes that for asymmetric duty cycles, the approach reduces to an NP-complete minimum vertex cover problem requiring a centralized solution [30]. These limitations are at odds with many of the requirements of our problem.

For Disco, the discovery latency is a function of the particular prime pairs being used as well as the offset in the node counters. For the grid quorum system, this latency is a function of the particular row and column choices being used as well as the group size, $m^2$, where these $m^2$ intervals or slots are arranged as a 2-dimensional $m \times m$ array in row-major manner [26]. For birthday protocols, the discovery latency is a function of the beacon/listen probability of each node [17].

Figure 4 shows the cumulative distribution of discovery latencies for Disco using the (37,43) balanced primes and symmetric pairs, Quorum using $m = 40$ (value of $m$ that makes $\frac{2m-1}{m^2} = 5\%$), and Birthday using probability $p_{tx} = p_{rx} = 0.05$, with all protocols operating at 5% duty cycle. The 50-th percentile discovery latency is 444 slots for Disco,

| Protocol | Cite | Parameters | Duty Cycle | Latency CDF($n$) | Asymm? |
|---|---|---|---|---|---|
| Disco | * | $(p_{i_1}, p_{i_2}), (p_{j_1}, p_{j_2})$ | $\frac{p_{i_1}+p_{i_2}-1}{p_{i_1} \cdot p_{i_2}}, \frac{p_{j_1}+p_{j_2}-1}{p_{j_1} \cdot p_{j_2}}$ | No closed form | Yes |
| Birthday | [17] | $0 \leq p_{tx}, p_{rx} \leq 1$ | $p_{tx}, p_{rx}$ | $1-(1-p_{tx} \cdot p_{rx})^n, \forall n \in \mathbb{Z}^+$ | Yes |
| Quorum | [26] | $m \in \mathbb{Z}^+$ | $\frac{2m-1}{m^2}$ | $1-(1-\frac{n}{m^2})^2, \forall n \leq m^2$ | No |
| Combinatoric | [30] | $k=p^q; p \in \mathbb{P}, q \in \mathbb{Z}^+$ | $\frac{k+1}{k^2+k+1}$ | $1-\frac{n}{k^2+k+1}, \forall n \leq k^2+k+1$ | No |

**Table 2: Comparison of asynchronous neighbor discovery techniques including Disco, Birthday, Quorum, and Combinatoric. The duty cycle and discovery latency of these techniques are parameterized by primes (Disco), transmit and receive probabilities (Birthday), the rank of a square matrix (Quorum), and powers of a prime (Combinatoric). The latency cumulative distribution function describes the probability of discovery after $n$ trials or slots as a function of the protocol parameters. An asymmetric protocol allows each node to choose a duty cycle independently of other nodes and still ensure discovery (with high probability in case of the Birthday protocol). $\mathbb{P}$ is the set of primes. $\mathbb{Z}^+$ is the set of positive integers.**

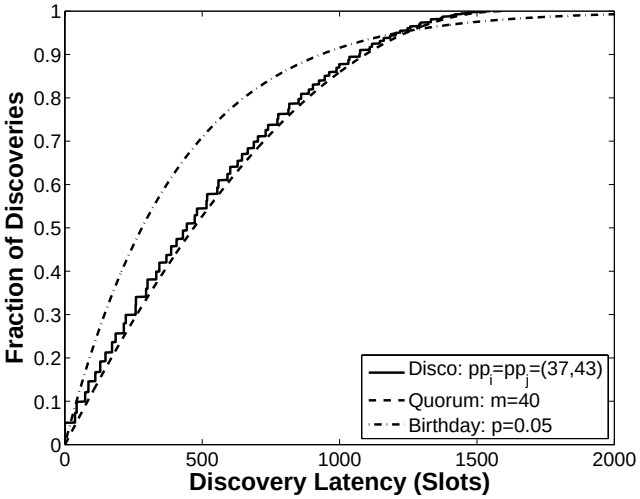


**Figure 4: The distribution and worst-case discovery latency of Disco closely matches the Quorum protocol. Both Disco and Quorum trail the Birthday protocol, which achieves the lowest latency 95% of the time, but its probabilistic nature leads to a long tail. The CDF of discovery latency for Disco, Quorum, and Birthday protocols operating at a 5% duty cycle is shown. The two Disco nodes use balanced primes and symmetric pairs (37,43); the Quorum system uses $m = 40$; and the Birthday protocol nodes both turn on their radio with probability $p = 0.05$.**

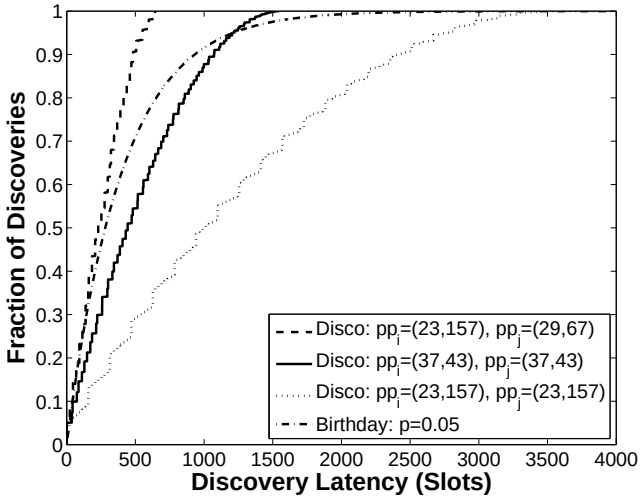


**Figure 5: The choice of prime pairs significantly affects the latency distribution and worst-case discovery latency. The CDF of discovery latency for three different Disco prime pairs as well as the Birthday protocol, all operating at a 5% duty cycle, is shown. Unbalanced primes in asymmetric pairs provide the best overall behavior and offer the lowest worst-case discovery latency – better than all other approaches. In contrast, unbalanced primes in symmetric pairs provide the worst average-case behavior and the highest worst-case discovery latency.**

470 slots for Quorum, and 281 slots for Birthday. The distribution and worst-case discovery latency of Disco closely matches the Quorum protocol, although Disco performs just slightly better. Both Disco and Quorum trail the Birthday protocol, which achieves the lowest discovery latency 95% of the time, but its underlying probabilistic nature leads to a long tail.

## 5.3 Discovery Latency: A Deeper Look

The discovery latencies in Figure 4 for Disco reflect a particular choice of prime pairs. In general, if two Disco peers select balanced primes and symmetric pairs, their discovery latency will closely track that of the Quorum protocol. If, on the other hand, nodes choose unbalanced primes and asymmetric pairs, then the discovery latency could be reduced significantly, or increased considerably, as Figure 5 shows. Three Disco cumulative distributions are plotted, showing the range of potential discovery latencies. The Birthday distribution for the same duty cycle is plotted as a baseline. Unbalanced primes in asymmetric pairs of (23,157) and (29,67) provide the best average-case behavior, 230 slots at the 50-th percentile mark, and they also offer the lowest worst-case discovery latency of 644 slots – better than all other approaches. In contrast, unbalanced primes in symmetric pairs of (23,157) and (23,157) provide the worst average-case behavior, 1012 slots at the 50-th percentile mark, and the highest worst-case discovery latency of 3454 slots. Balanced primes in symmetric pairs of (37,43) and (37,43) and Birthday using probability $p_{tx} = p_{rx} = 0.05$ is shown for comparison.

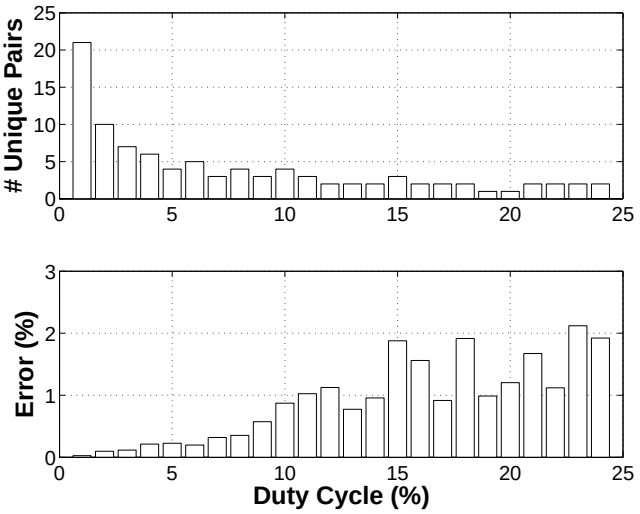


**Figure 6: The number of unique prime pairs that generate a particular duty cycle and largest error across all of the pairs for each duty cycle. Error is measured as the magnitude of the deviation from the desired duty cycle, e.g. for primes $a$ and $b$, and duty cycle $c$, the error is $\left|\frac{1}{c} - \frac{a+b-1}{a \times b}\right|$.**

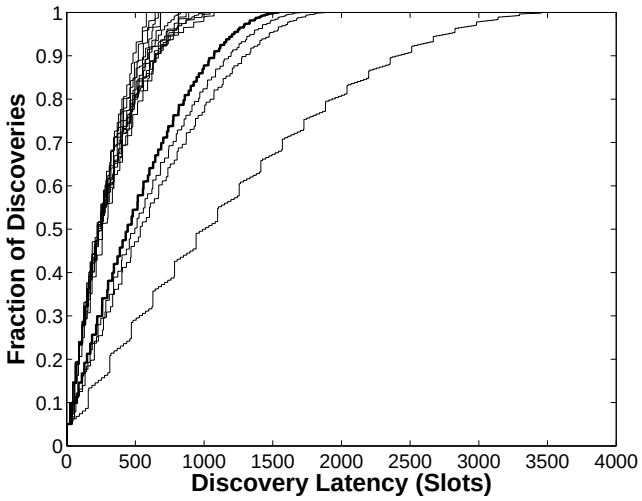


**Figure 7: The cumulative distribution of discovery latency across all 16 possible prime pair values for a 5% duty cycle. Although the worst-case discovery latency is 3,611 slots when both nodes choose the (23,157) pair, the median discovery time is much lower.**

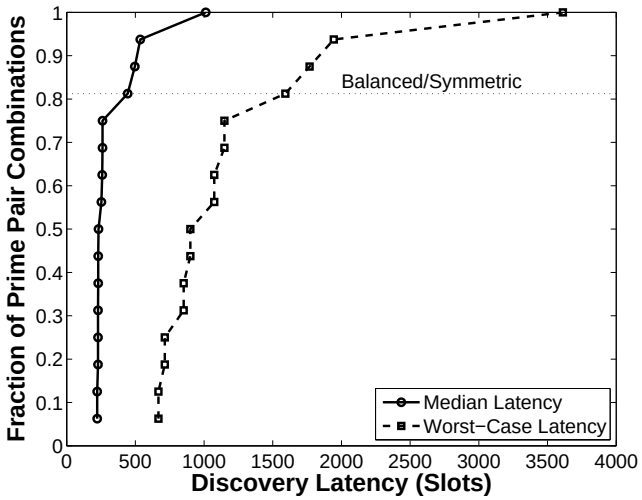


**Figure 8: The cumulative distribution of the median (50-th percentile) and worst-case (100-th percentile) discovery latency values across all 16 possible prime pair combinations for a 5% duty cycle. The dotted line intersects the CDF of the balanced primes (37,43) in symmetric pairs.**

The ratio of the worst-case discovery latency between an a good and bad pairing is bounded by the duty cycle. For example, a 10% duty cycle results in no worse than a 1:10 ratio, a 5% duty cycle is bounded by a 1:20 ratio, and a 1% duty cycle results in at worse a 1:100 ratio. While these numbers paint a grim picture of the worst-case downside, they fail to capture *how likely* that downside is to occur. Since unbalanced primes in symmetric pairs result in the worst discovery latency, it is worth exploring how often such bad pairs occur and what is lost if more conservative pairings are used.

Bad pairings occur whenever two nodes pick the same prime pairs. The key question is how often this happens. Assume that two nodes operate at the same duty cycle and they choose their prime pairs independently and uniformly randomly from a set of $k$ prime pair choices. Then, the chance that they pick the same pair is $1/k$. If $k$ is large, then this chance is small. Figure 6 shows the number of unique prime pairs for each possible integral duty cycle from 1% to 25% as well as the largest duty cycle error across all pairs for a given duty cycle. The number of unique pairs possible for each integral duty cycle value grows quickly as the duty cycle falls below 5%.

Since 5% appears to be near the elbow of the curve in the number of unique pairs in Figure 6, we next explore the cumulative distribution of discovery latencies across all 16 prime pair combinations for a 5% duty cycle. Figure 7 shows the distribution of discovery latencies of all sixteen pairings possible with each node choosing one of the following prime pairs: (23,157), (29,67), (31,59), (37,43). The dark line with a worst-case latency of 1,591 slots highlights the case when both nodes select the (37,43) pair. All lines to the right of the (37,43) line are cases when both nodes choose the same pair while all lines to the left of the (37,43) line are cases when nodes choose dissimilar pairs. The conclusion is clear: asymmetric pairs dramatically reduce discovery latency, by 30 – 50%.

Figure 8 shows the cumulative distribution of the median (50-th percentile) and worst-case (100-th percentile) discovery latency *across all possible prime pair combinations* for a 5% duty cycle. These values are taken directly from the 50-th and 100-th percentile data points from the 16 CDFs shown in Figure 7. The horizontal line labeled "Balanced/Symmetric" identifies the median and worst-case discovery latencies when both nodes select the (37,43) pair.

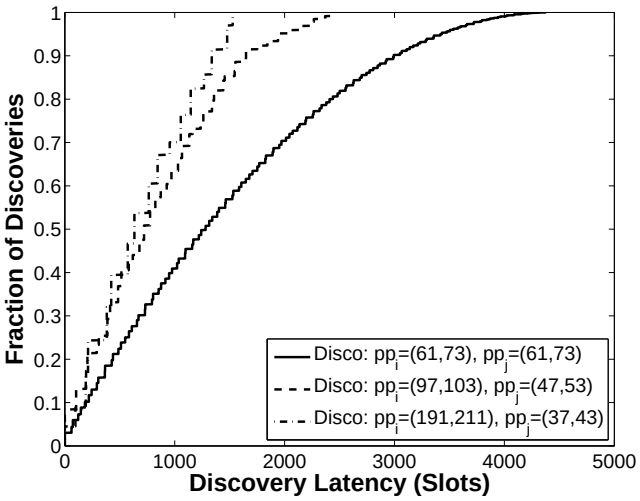


**Figure 9: Discovery latency decreases with increasing asymmetry in pairwise duty cycles for a fixed average duty cycle. The CDF of discovery latency for an average duty cycle of 3% is shown. This 3% average is achieved in three ways: (3%+3%)/2 using prime pairs (61,73) and (61,73), (2%+4%)/2 using prime pairs (97,103) and (47,53), and (1%+5%)/2 using prime pairs (191,211) and (37,43).**

The key observation is that the data show excellent average-case performance: 75% of all pairwise combinations result in median discovery latency of less than 261 slots and over 93% of all combination result in a median discovery latency of less than 536 slots, 2.61 seconds and 5.36 seconds, respectively, using our implementation slot length of 10 ms.

## 5.4 Impact of Duty Cycle Asymmetry

In some docking applications, discovery occurs between nodes with dissimilar energy supplies [28]. In other applications, a fixed amount of energy may be allocated between two or more nodes [19]. And, in a network of equal-energy nodes, operating nodes at different duty cycles makes sense [7]. Figure 9 shows that inter-pair asymmetry reduces discovery latency for a fixed pairwise-average duty cycle. This suggests that more powerful beacons combined with less powerful mobile tags is feasible and beneficial, and well supported by the algorithm.

## 5.5 Latency-Driven Discovery

In some applications, the encounter window of two nodes is short, and it becomes necessary to configure the duty cycle, $DC$, or beacon rate, $f_{beacon}$, to ensure that discovery occurs within this short window [3, 10]. The duty cycle and beacon rate depend on the maximum tolerable discovery latency, $t_{disco}$, and the length of a beaconing slot, $t_{slot}$, as derived in Section 3.5. The duty cycle required to ensure discovery in time $t_{disco}$ is

$$DC \geq 2\sqrt{\frac{t_{slot}}{t_{disco}}}$$

and beacon slot rate required to ensure discovery is

$$f_{beacon} \geq \frac{2}{\sqrt{t_{disco}t_{slot}}} \text{ Hz}$$

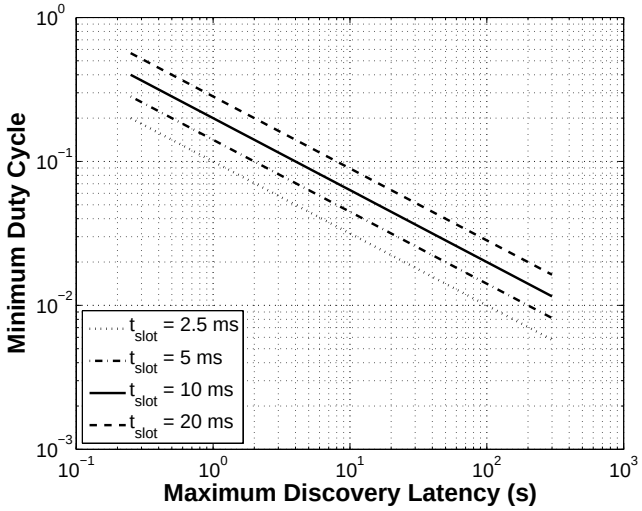


**Figure 10: The minimum duty cycle required to ensure a maximum discovery latency.**

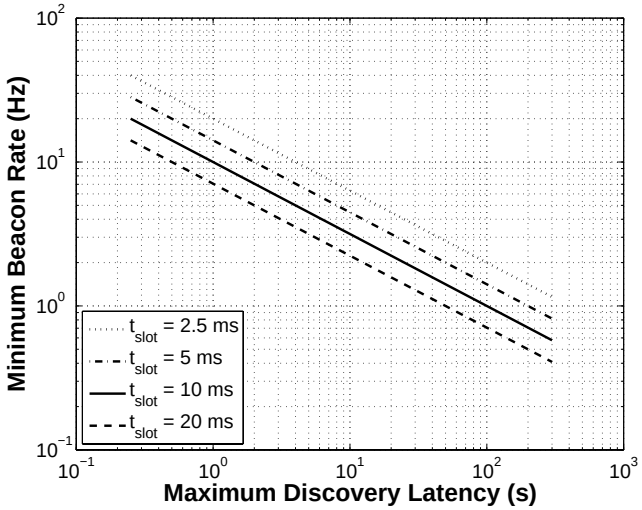


**Figure 11: The minimum beacon rate required to ensure a maximum discovery latency.**

Figures 10 and 11 show the minimum duty cycle and beacon slot rate, respectively, required to ensure a maximum discovery latency across a range of $t_{slot}$ values.

# 6. EMPIRICAL EVALUATION

In this section, we evaluate the performance of Disco empirically, based on our TinyOS implementation that runs on Telos motes. We study the sensitivity of Disco to slot length, real-world effects like clock skew and jitter, and node density.

## 6.1 Discovery Rate

Discovery rate refers to the number of discovery beacons received per unit time. Figure 12 shows an empirical timeline of discoveries between a pair of Telos nodes for two values of the slot period, $t_{slot}$, collected over two one hour periods. For a $t_{slot}$ value of 10 ms, 150 discoveries are observed over a one hour period, which translates to an average discovery period

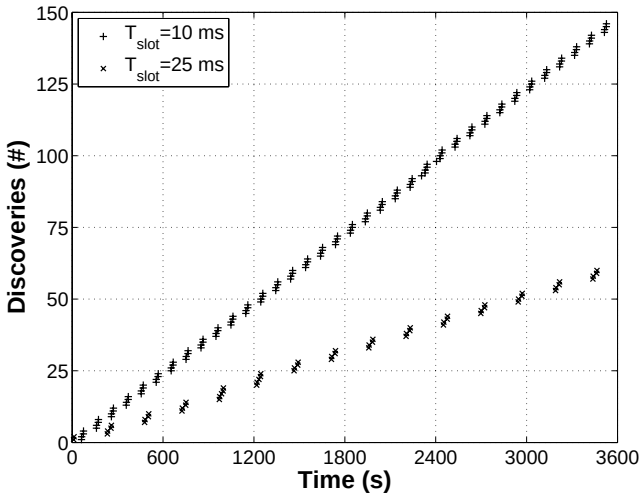


**Figure 12: Empirical discovery timeline for two different $t_{slot}$ values (10 ms and 25 ms) using balanced primes and symmetric pairs (97, 103). Rendezvous is stable and predictable over the one hour experiment.**

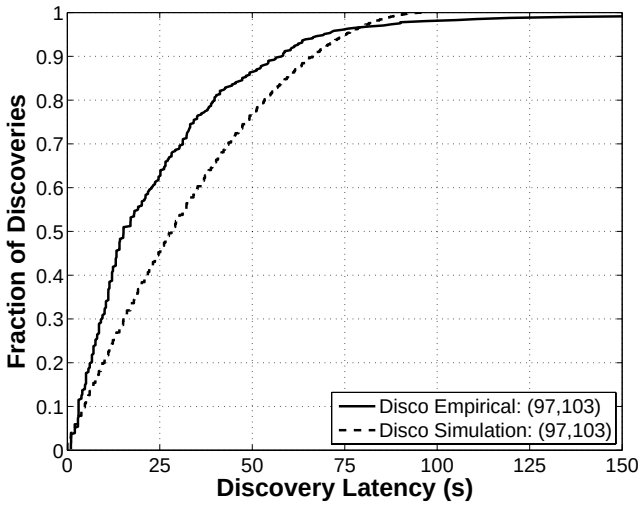


**Figure 13: The empirical discovery latency is lower than the simulated discovery latency in 95% of trials ($N = 408$) but in 2% of the trials, the empirical discovery latency exceeds the worst-case discovery latency obtained through simulation. The nodes are operating with a $t_{slot} = 10$ ms and at a 2% duty cycle using balanced primes and symmetric pairs (97, 103) for both empirical and simulation results.**

of about 24 seconds. For a $t_{slot}$ value of 25 ms, 60 discoveries are observed over a one hour period, which translates to an average discovery period of about 60 seconds. The ratio of these two numbers matches the ratio of the two different $t_{slot}$ values, as expected, showing that the average discovery rate scales linearly with slot length, even though duty cycle and beacon rate do not. Note that with a 2% duty cycle using *symmetric* pairs (97,103), the worst-case discovery latency is 9,991 slots or about 100 seconds for the 10 ms slot and 250 seconds for the 25 ms slot.

## 6.2 Discovery Latency in Clusters

Figures 13 and 14 compare how simulated and empirical results compare in clusters. These two figures present the same underlying data in different ways. To collect this data, seven nodes were programmed to operating at a 2% duty cycle using balanced primes and symmetric pairs (97, 103). The unique neighbor discoveries of one particular node (the "test" node) were logged. Each time the test node discovered all of its neighbors, it was held in reset. Each of the other nodes were randomly reset while the test node was held in reset. The test node started running (i.e. was released from reset) a random amount of time after the other nodes were reset. The discovery latency of each neighbor (time to first discovery from reset) was logged.

The distribution of empirical discovery latencies is shown in Figure 13 along with the simulated latencies. The empirical discovery latency is lower than the simulated discovery latency in 95% of trials ($N = 408$) but in 2% of the trials, the empirical discovery latency exceeds the worst-case discovery latency obtained through simulation. This difference is principally due to the fact that slots are not aligned in practice while in simulation they are aligned. It may also be due to clock skew and jitter, both of which contribute to small variations in slot times.

Figure 14 shows the time to discover the 1st, 2nd, 3rd, 4th, and 5th neighbor when a node joins a cluster (simulated by being reset). Note that the discovery latencies are for the first through fifth neighbors discovered regardless of the actual neighbor identifier. Discovery latency times for the sixth neighbor is not shown because of a long tail (the tail is, however, shown in Figure 13. The long tail may be an artifact of channel contention or collisions, especially since all nodes were using the same prime pairs. This data suggest that beaconing rate adaptation may be necessary for node clusters of modest density.

Despite the long tail, the median discovery latency for the first (of six) neighbors is far lower than the median discovery across all the neighbors shown in Figure 13. This suggests that sharing neighbor table information may decrease neighbor discovery latency.

# 7. DISCUSSION

Having presented and evaluated the Disco design both empirically and through simulation, we now revisit some open issues that merit further study and discuss some possible extensions to this work.

## 7.1 Beacon Rate Adaptation

As node density increases, beaconing consumes an increasing fraction of channel activity. For Disco to be able to scale to high densities, the beaconing rate must be reduced when nodes are in high density clusters. Disco currently allows the application to control duty cycle, beaconing mode (beacon and listen, or listen only), and other service parameters. Factoring out these policies from their underlying mechanisms makes sense, especially since we do not have enough experience with applications to craft an appropriate adaptation policy nor do we understand how such a policy would affect the predictability of discovery.

Still, we can envision some approaches to beacon rate adaptation. In one scenario, nodes could track how often

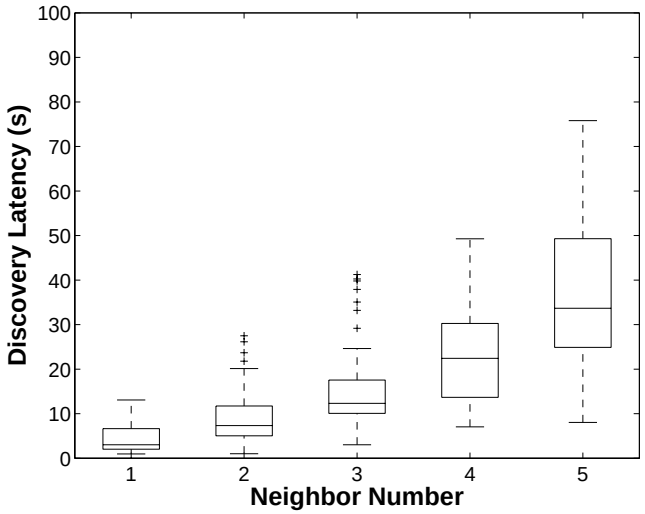


**Figure 14: The discovery latency of the 1st, 2nd, 3rd, 4th, and 5th neighbors (regardless of their actual ids) when a node joins a cluster with six nodes.. The empirical discovery latency is lower than the simulated discovery latency in 95% of trials ($N = 408$) but in 2% of the trials, the empirical discovery latency exceeds the worst-case discovery latency obtained through simulation. The nodes are operating with a $t_{slot} = 10$ ms and at a 2% duty cycle using balanced primes and symmetric pairs (97, 103) for both empirical and simulation results. The sixth node is not shown because of a long tail, suggesting beaconing adaptation may be needed in dense node clusters.**

they hear neighbors and they probabilistically adjust whether to both beacon and listen during their on slots or just to listen. If nodes are just listening, then they would send beacons during their normally scheduled rendezvous slots with nodes in their neighbor table to ensure that routing links remain connected and synchronized, but they would not send other beacons. In this scenario, a neighborless mobile node would beacon while nodes in the cluster would (largely) listen. Upon discovering a new node, members of the cluster would gossip about their own neighbors with the new node, which perhaps would help the new node discover its neighbors more quickly.

## 7.2 Robustness to Clock Skew

Clock skew presents a challenge for many synchronized protocols and requires nodes to estimate and adjust for neighbors' clock drift to maintain synchronization. Some time synchronization protocols use linear regression to perform this function [16, 22] but since Disco generally operates in an unsynchronized manner, it only updates information about clock offsets and does not compute skews. Over short timescales, this approach works well, but over longer timescales, nodes that have significantly different skews are likely to have difficulty with rendezvous, but asynchronous discovery should still work. Whether or not skew compensation is needed is unclear but one way to explore this question is to assume clock skew is $r_{skew} = \Delta f/f$. Under normal circumstances, a cycle repeats every $m = p_{i_x} \cdot p_{j_y}$ slots. If the skew during a cycle must be less than some slot fraction $\alpha$, to ensure overlapping slots, then we must ensure

$$t_{slot} \cdot p_{i_x} \cdot p_{j_y} \cdot r_{skew} \leq \alpha \cdot t_{slot} \quad (5)$$

For typical values of $p_{i_x} = 97$, $p_{j_y} = 103$ (a 2% duty cycle), and a conservative skew assumption $r_{skew} = 50$ ppm, $\alpha \approx 0.5$, suggesting that with typical crystals operating at the very extremes of their temperature specification, there could be up to 1/2 of a slot-width's phase shift every $m$ slots when running at a 2% duty cycle. This may be the case for some of the nodes used in our earlier experiments. Because of the design of our slots – a beacon at the beginning and one at the end – a phase shift simply means that a *different* pair of slots will overlap, quite possibly sooner than predicted by the Chinese Remainder Theorem, since every offset between zero and $max(p_{i_x}, p_{j_y})$ occurs between overlapping slots.

For example, in Figure 1, we see that between time $x = 7$ and time $x = 22$, node $i$ and node $j$ have awake slots that are offset from each other by every value between zero and $max(3, 5) = 5$. Offsets of zero ($x = 7$), one ($x = 12$ and $x = 13$; $x = 16$ and $x = 17$), two ($x = 10$ and $x = 12$; $x = 17$ and $x = 19$), three ($x = 7$ and $x = 10$), four ($x = 12$ and $x = 16$; $x = 13$ and $x = 17$) and five ($x = 7$ and $x = 12$; $x = 17$ and $x = 22$) all appear. This suggests that even with clock skew, there may be overlaps that continue to occur. Understanding and characterizing this phenomenon may allow use of discovery at duty cycles below 1% and even at the extremes of the temperature operating range.

## 7.3 Gossip

The current implementation does not make effective use of gossip even though beacons do include a neighbor count field (which indicates the number of one-hop neighbors in the neighbor table). We envision that in the future, if a node $i$ receives a beacon from a node $j$ with a neighbor count less than some threshold (e.g. two), then node $i$ could send a longer beacon with (a random subset of) the entries of $i$'s own neighbor table to help node $j$ probabilistically speed up its own neighbor discovery attempts. We suggest that the speedup may be probabilistic in nature because $i$'s neighborhood may or may not overlap $j$'s neighborhood. The rationale for keeping beacons small normally, and sending long beacons infrequently, is that it optimizes for the common case: sending long beacons most of the time wastes bandwidth and energy as most beacons would not be heard in mobile, low-power, or low-density networks.

## 7.4 Combining Discovery and MAC

The current Disco implementation runs as a standalone client of the TinyOS radio stack and uses the standard link layer message abstraction, `ActiveMessage`, to directly access and control the radio. Since Disco is sensitive to timing jitter, it requires exclusive control of the radio to meet its timing constraints, and it therefore does not play well with others today. Since Disco performs some of the functions of a MAC, it may make sense for the Disco service itself to present the same application programming interface as the current stack does, and just layer on top of it.

For this approach to work, Disco would have to provide the standard TinyOS `Packet`, `AMPacket`, `Send`, `AMSend`, and `Receive` interfaces through a set of generic components that virtualize the communication services. These services can be layered above the standard TinyOS Hardware Interface Layer (HIL), a platform independent abstraction over the

hardware. Since the HIL hides hardware differences, porting Disco to other platforms would be easy. In addition to these TinyOS interfaces, providing additional interfaces for neighbor table management, phase adjustment, or neighbor schedule tracking may be useful.

### 7.5 Secure Discovery

In some applications, it may not be advisable for mobile nodes to normally beacon (i.e. they only listen), and only respond if they first receive a peer's message that can be authenticated. A motivating application would be wireless sensors mounted on military vehicles or personnel. Beaconing would be undesirable because opposing forces might be able to detect and track these transmissions. Peer authentication, however, is tricky. It is not enough for a peer to digitally sign a beacon because such a beacon could be captured and replayed across an entire city using a high power transmitter. Including a timestamp in the beacon doesn't protect against a replay attack either. Identity, time, and location must be authenticated to ensure that a node only replies when it is verifiably close in both space and time to an authenticated peer. The Disco application programming interface needed to support secure discovery may require a tighter coupling or greater application-layer influence over the beacons.

## 8. CONCLUSION

This paper presents a practical solution to the low-power, asynchronous neighbor discovery problem. Our solution is conceptually simple and easy to implement: nodes pick a pair of dissimilar primes such that the sum of their reciprocals is equal to the desired radio duty cycle. Each node increments a local counter and if a node's counter is divisible by either of its primes, the node turns on its radio for a single interval, whose length is the only global parameter. This simple protocol achieves discovery faster than other discovery protocols for a given duty cycle, allows nodes to independently select their own duty cycle, offers a provable upper bound on discovery latency, and performs better than expected in practice.

Going forward, we envision several new directions for this work. Comparing the differences between the empirical and simulated results shows that the simulation models are too conservative. An obvious extension to this work would be to develop richer models that can more faithfully simulate the real-world factors like jitter, clock skew, and radio interactions that occur during neighbor discovery, and more importantly, which tend to improve discovery performance. Since Disco performs many of the operations of a MAC including neighbor discovery, neighbor table management, rendezvous management, and media access arbitration, providing a complete set of MAC primitives would make Disco more useful. Specifically, wrapping the discovery service in a standard TinyOS interface, integrating it into a MAC layer, and adding better gossip support are key. Extending the ideas presented in this paper to support neighbor discovery in a multi-channel MAC could also be fruitful and timely.

The asynchronous neighbor discovery problem principally arises when two systems that are awake infrequently must discover each other without any prior knowledge of their potential encounters, and without external assistance. However, we observe that efficient, asynchronous discovery is needed in many other contexts as well. Static networks, for example, can experience link churn and may need to rediscover neighbors. Other applications with mobility patterns like docking and flocking can also benefit from discovery. These observations lead us to conclude that a continuous, asynchronous, neighbor discovery process should be a basic communications service integrated into the medium access control function for all wireless networks.

## 9. ACKNOWLEDGMENTS

Special thanks to Rodrigo Fonseca, Igor Ganichev, Timothy Wark, Richard Han, and the anonymous reviewers for their insightful and constructive comments. This material is based upon work supported by the National Science Foundation under grants #0435454 ("NeTS-NR") and #0454432 ("CNS-CRI"), a grant from the Keck Foundation, an NSF Graduate Fellowship, a Microsoft Graduate Fellowship, and generous gifts from Aginova, HP, Intel, Microsoft, and Sharp.

## 10. REFERENCES

[1] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney. Reminding about tagged objects using passive rfids. In *Ubicomp*, pages 36–53, 2004.

[2] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble mac protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 307–320, New York, NY, USA, 2006. ACM.

[3] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, and R. Kumar. Brimon: a sensor network system for railway bridge monitoring. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 2–14, 2008.

[4] T. K. Choudhury. *Sensing and modeling human networks*. PhD thesis, Massachusetts Institute of Technology, 2004.

[5] P. Dutta, D. Culler, and S. Shenker. Procrastination Might Lead to a Longer and More Useful Life. In *The 6th Workshop on Hot Topics in Networks (HotNets VI)*, 2007.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI'03: Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*, June 2003.

[7] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283, 2004.

[8] T. Herman, S. V. Pemmaraju, L. Pilard, and M. Mjelde. Temporal partition in sensor networks. In *SSS '07: 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 325–339, 2007.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. In *ASPLOS-IX: Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[10] J.-H. Huang, S. Amjad, and S. Mishra. Cenwits: a sensor-based loosely coupled search and rescue system using witnesses. In *SenSys '05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 180–191, New York, NY, USA, 2005. ACM.

[11] J.-R. Jiang, Y.-C. Tseng, C.-S. Hsu, and T.-H. Lai. Quorum-based asynchronous power-saving protocols for IEEE 802.11 ad hoc networks. *Mobile Networks and Applications*, 10(1-2):169–181, 2005.

[12] J. Leguay, T. Friedman, and V. Conan. Evaluating mobility pattern space routing. In *INFOCOM'06: Proceedings of the 25th IEEE Conference on Computer Communications*, 2006.

[13] Y. Li, W. Ye, and J. Heidemann. Energy and latency control in low duty cycle MAC protocols. In *IEEE WCNC '05: Proceedings of the IEEE Wireless Communications and Networking Conference*, 2005.

[14] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with Impala and ZebraNet. In *MobiSys '04: Proceedings of the 2nd International conference on Mobile Systems, Applications, and Services*, pages 256–269, New York, NY, USA, 2004. ACM.

[15] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. A. Paradiso. Cargonet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *SenSys '07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, pages 145–159, New York, NY, USA, 2007. ACM.

[16] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, 2004.

[17] M. J. McGlynn and S. 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, 2001.

[18] I. Niven, H. S. Zuckerman, and H. L. Mongomery. *An Introduction to the Theory of Numbers*. John Wiley & Sons, 1991.

[19] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. H. Growdon, M. Welsh, and P. Bonato. Analysis of feature space for monitoring persons with parkinson's disease with application to a wireless wearable sensor system. In *Proceedings of the 29th IEEE EMBS Annual International Conference*, Aug. 2007.

[20] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Sensys'04: Proceedings of the Second ACM Conferences on Embedded Networked Sensor Systems*, 2004.

[21] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN '05: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, page 48, 2005.

[22] H.-S. W. So, G. Nguyen, and J. Walrand. Practical synchronization techniques for multi-channel mac. In *MobiCom '06: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, pages 134–145, 2006.

[23] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 161–174, 2007.

[24] J. Taneja, J. Jeong, and D. E. Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *IPSN '08: Proceedings of the 7th international Conference on Information Processing in Sensor Networks*, pages 407–418, 2008.

[25] G. Tolle, J. Polastre, R. Szewczyk, D. E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Sensys'05: Proceedings of the Second ACM Conferences on Embedded Networked Sensor Systems*, pages 51–63, 2005.

[26] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. In *INFOCOM'02: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Comm. Soc.*, 2002.

[27] R. Want, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10:91–102, 1992.

[28] T. Wark, W. Hu, P. Sikka, L. Klingbeil, P. Corke, C. Crossman, and G. Bishop-Hurley. A model-based routing protocol for a mobile, delay tolerant network. In *SenSys '07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, pages 421–422, 2007.

[29] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM'02: Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, June 2002.

[30] R. Zheng, J. C. Hou, and L. Sha. Asynchronous wakeup for ad hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 35–45, 2003.