2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 979-8-3503-2433-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASS58611.2023.00048

CANDor: Continuous Adaptive Neighbor Discovery

Evan King and Christine Julien
Department of Electrical and Computer Engineering, University of Texas at Austin
{e.king, c.julien}@utexas.edu

***Abstract*—Many applications require continuous awareness of the set of surrounding devices. A user entering a smart space benefits from the ability to quickly discover which devices they can control nearby. This ability to know "who is around" is commonly provided by *continuous neighbor discovery* protocols, which entail a *schedule* on which devices alternate between *beaconing* to advertise their presence and *listening* to detect the presence of neighbors. In existing protocols, these schedules are carefully configured to achieve application-level objectives—a target discovery latency, discovery probability, and energy consumption. These configurations are determined statically based on assumptions about network conditions (e.g., the number of expected neighboring devices, or nodes). However, the conditions that a given node typically experiences are dynamic. To handle these dynamics, existing protocols tend to be configured for the worst case, which results in schedules that waste energy. In this paper, we identify a *signal* for adaptation of continuous neighbor discovery and show how existing protocols can incorporate this signal to adapt their behavior to achieve consistent discovery probability and latency. The signal we adapt to is changing node density, and our novel insight is that *we can extract this signal directly from the performance of neighbor discovery itself*. We show that our approach effectively senses and adapts even when using initially suboptimal schedules in dynamic mobile environments, allowing neighbor discovery protocols to maintain their performance guarantees without added sensing overhead.**

***Index Terms*—continuous neighbor discovery, mobile computing, adaptive networking**

I. Introduction

Many mobile computing technologies rely on continuous discovery and awareness of surrounding devices. These technologies run the gamut, from user interfaces that provide control of nearby smart devices [1], to contact tracing during a pandemic [2]. This is enabled by *continuous neighbor discovery*, which relies on a *schedule* of when devices *advertise* their presence and *listen* for the presence of neighbors. The schedule determines how likely it is for devices to discover each other within a window after initial contact, as well as how much energy is consumed doing so. Suboptimal schedules result in slow discovery or consume excess energy.

As the number of deployed devices grows, an increasingly important factor in neighbor discovery performance are *beacon collisions* that occur when devices advertise at the same time. This effect is particularly significant in areas with a high density. Crowded spaces, for instance, where digital contact tracing applications might be expected to provide an accurate assessment of disease risk are ironically the places where neighbor discovery is *least effective* due to collisions [3].

Reducing collisions by configuring schedules to account for dynamics of node density can provide increased confidence that a target discovery probability and latency can be met with minimal energy consumption. Prior work has shown that such schedules can be determined using analytical models [4] [5] [6], but with a notable limitation: *these methods assume the number of surrounding devices is static*. This is far from realistic, especially in mobile applications: users often divide time between diverse environments throughout the day—from home, to busy urban spaces, to the office, to nature retreats. In spite of this, application developers must choose a fixed estimate of the number of neighbors at pre-deployment time. Over-estimates and under-estimates both negatively impact performance. Overestimation can shave meaningful time off a device's battery life due to missed opportunities for energy conservation; in the case of underestimation, lofty expectations about discovery probability and latency fail to be met—discovery and control of nearby smart devices therefore becomes frustrating for users due to lag, and applications that rely on accurate discovery fail to perform as expected.

We address these challenges with CANDor, a method for Continuous Adaptive Neighbor DiscOveRy that uses an analytical model to continuously adapt devices' schedules to a sensed estimate of the true node density. CANDor centers on the intuition that *the performance of neighbor discovery is a signal* that can be analyzed to glean information about the state of the dynamic network. By assessing and adapting to the real state of the network environment, mobile applications can make use of neighbor discovery capabilities with higher assurance that pre-deployment promises about energy use, discovery probability, and discovery latency will be kept.

Our contributions are summarized as follows:

- We formalize an adaptive approach that *does not require additional sensing hardware* or contextual information to adapt neighbor discovery to the number of neighbors.
- We apply our approach to two exemplars, showing that it generalizes to existing neighbor discovery protocols.
- We show that CANDor is capable of sensing surrounding node density and adapting a protocol, *even in unstable and dynamic mobile environments*.

II. Background & Related Work

Slotted protocols model fixed-length "slots" during which a device is either advertising, listening, or sleeping (e.g., Nihao [7], U-Connect [8], and Birthday [6]). Our approach draws inspiration from Birthday in particular since it is one of the earliest works to investigate partial adaptation (nodes switch between schedule modes depending on discovery state) and because it explicitly models collisions. Slotless approaches

2155-6814/23/$31.00 ©2023 IEEE
DOI 10.1109/MASS58611.2023.00048

place fewer constraints on the respective lengths of advertising, listening, and sleeping (e.g., SingleInt and MultiInt [5], BLEnd [4], and Griassdi [9]). Griassdi is an interesting example of partial adaptation, where nodes perform "assisted two-way discovery" that adapts advertising and scanning to minimize latency; similarly BLEnd adaptively transmits beacons to enable two-way discovery based on beacons received.

Adaptive approaches improve neighbor discovery performance often using additional sensing capabilities to choose schedules that are sensitive to context, e.g., based on patterns at a location or based on past encounters [10]. Such approaches assume that devices have positioning hardware (e.g., GPS) and additional storage to log the location of previous contacts. Renzler et al. use information about encounters—specifically, trends in user interaction with an immobile smart device—to choose schedules that are more optimal for different times of day [1]. Hess et al. use mobility as input to adaptation, choosing to beacon only at periods with low movement (and thus a higher probability of prolonged encounters) [11]. Both approaches are similar to ours in that they exploit information already available to adapt. Both cases leverage context for narrowly-focused applications: either an immobile smart device with no consideration of mobility, or *requiring* devices to be mobile to provide adaptation. Our approach differs from existing ones in that it generalizes to a variety of application scenarios—mobile or immobile—by relying only on contextual signals in the behavior of neighbor discovery itself. We do not rely on additional hardware, providing adaptation *for free*. While we focus on the tradeoffs in employing a particular discovery signal for adaptation, these approaches could be combined in applications where multiple signals are available.

III. Best Guesses & Static Schedules

Continuous neighbor discovery protocols switch between *advertising* to announce one's presence and *listening* to hear neighbors advertising their presence. A protocol's schedule Ω determines when to *advertise*, *listen*, and sleep. Practical applications typically require that the protocol operate within an energy budget while providing a bound on *discovery probability* P and *discovery latency* Λ. P is the probability of a node discovering all of its neighbors with the latency given by Λ. The protocol's ability to achieve the target P and Λ within an energy budget is a function of the schedule Ω and the *estimated number of neighbors* within communication range, N_e. In existing protocols, schedules are established *statically*. However, in real world deployments, the number of neighboring devices can vary dramatically over time [12], making static schedules perform sub-optimally.

We motivate adaptive neighbor discovery using an exemplar protocol [4] to empirically show two things: (1) when a protocol *over-estimates* node density, energy resources are wasted and (2) when a protocol *under-estimates* node density, it fails to achieve target performance guarantees.

Schedules that achieve P within Λ while considering collisions tend to listen more and advertise less as N_e grows [4], [6]. More frequent (continuous) listening, however, comes at a cost. Ideally, less talkative schedules are used only when the actual node density (which we term N_a) is high so that energy is not needlessly consumed by frequent listening. Using a statically-configured schedule in different-than-expected node densities, however, results in missed opportunities for energy savings, as we illustrate in Fig. 1.

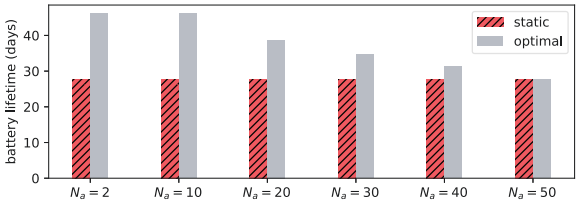

Fig. 1: Battery lifetime comparison ($N_e = 50$).

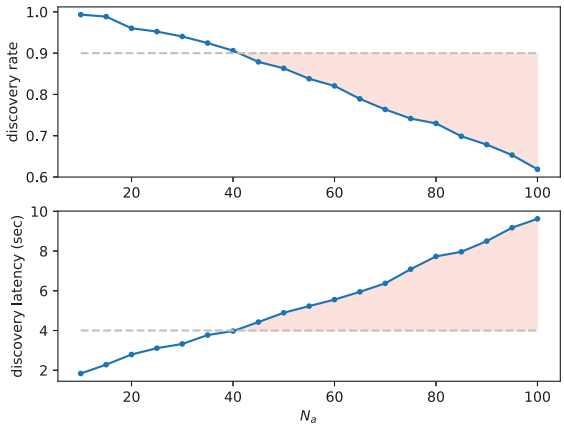

Fig. 2: Performance of a static schedule set for $N_e = 20$. As N_a exceeds N_e, discovery rate and latency miss the target.

Small differences between schedules compound. For example, if a schedule on a low-power sensing device (e.g., a TI SensorTag) with a 320 mAh battery is configured for $N_e = 50$, yet has $N_a = 20$ for its entire lifetime, its battery is expended about *12 days sooner* than if the schedule reflected the actual number of neighbors.

If the potential for wasted energy is a concern, applications can underestimate N_e for a target P and Λ. However, schedules that assume a lower N_e than reality will harm P and Λ due to an increase in beacon collisions caused by more talkative schedules. The top of Fig. 2 depicts the actual discovery rate relative to the promised P as N_a grows past an underestimated value N_e. The bottom portrays the inverse effect on discovery latency—as N_a grows past N_e, the time it takes for devices to discover one another grows. The consequences of broken discovery probability and latency promises are degraded application performance: this can frustrate users attempting to control nearby devices, or make it difficult to provide an honest assessment of disease risk [3].

Application developers tasked with configuring discovery schedules must choose between two evils: overestimate N_e at the expense of battery life, or underestimate it at the cost of broken promises about performance. There is thus a strong justification for adaptive neighbor discovery.

IV. Approach

We first formalize discovery performance as a signal then describe how we use this to adapt protocol schedules.

A. Neighbor Discovery as a Signal

Our framing principle is that observing the real-time performance of neighbor discovery provides insight into the true state of the surrounding network—even when that state is sampled using a suboptimal schedule. We sense the *actual* surrounding node density, N_a, by analyzing the observed performance of a schedule Ω optimized for an *estimate* of node density, N_e. This estimate enables us to adapt Ω such that N_e better approximates N_a. To estimate N_a, we compare the actual performance of Ω to its expected performance, as captured by an analytical model of the protocol. In short, we compare the number of nodes we *expect* to discover with the number of nodes the protocol *actually* discovers.

We formalize our approach under a few assumptions:

- all N_a nodes have the same schedule
- for a window in which we explore adapting, N_a is stable
- the underlying neighbor discovery protocol models the fraction of N neighbors expected to be discovered given a schedule Ω over a given window of time

The first of these simplifies our formalization—it allows us to model a single value for the schedule parameters across all nodes. Clearly, it is impossible for all nodes to adapt independently and achieve this goal; *our evaluation shows that this assumption is unnecessary in practice*. We also later demonstrate that the second assumption can be relaxed.

Consider a function $F_d(\Omega, N, w)$ that computes for any schedule Ω and node density N what fraction of N are expected to be discovered during a window w. The units and value of w vary based on the protocol—for slotted protocols, it is typically a number of slots, while for slotless protocols it is a number of full periods of listening and advertising.

We extend F_d to define the expected number of nodes discovered in w, $D(\Omega, N, w) = F_d(\Omega, N, w) \times N$. Choosing the hypothetical value of N input to D provides insight into how Ω is expected to perform in different contexts. If we input N_e, the *estimated* number of neighbors, we get the number of neighbors we expect to discover when our estimate is correct. Consider $N_e = 10$ and we know from a model of the protocol that some specific schedule $\hat{\Omega}$ is expected to discover 90% of those nodes in the window of time $\hat{w}$. In other words, $F_d(\hat{\Omega}, 10, \hat{w}) = 90\%$, meaning $D(\hat{\Omega}, 10, \hat{w}) = 0.9 \times 10 = 9$ nodes. If the protocol discovers 9 neighbors using the schedule $\hat{\Omega}$ during $\hat{w}$, the actual number of neighbors N_a *may* be equal to our estimate N_e. There are indeed situations where this is not the case, which we address in the following.

To compare these expectations with reality, we define an observed quantity N_{da}, the number of neighbors *actually* discovered in the window w. We can compute $D(\Omega, N_e, w)$ *a priori* since we know Ω and N_e. We must *measure* N_{da} at run time. However, a single sample of N_{da} over one window w does not reveal what fraction of N_a has been discovered in w. Consider a situation where $N_e = 10$ but $N_a = 100$, i.e., $\hat{\Omega}$ is constructed with parameters that exceedingly *underestimate* the true number of nearby devices. This will result in talkative schedules that produce a large number of collisions and thus degrade performance. It is possible, then, that $D(\hat{\Omega}, 10, \hat{w}) = 9$ nodes and N_{da} *also equals* 9, not because there are actually $N_a = N_e = 10$ neighbors, but because *performance is so degraded by collisions* that $\hat{\Omega}$ severely underperforms.

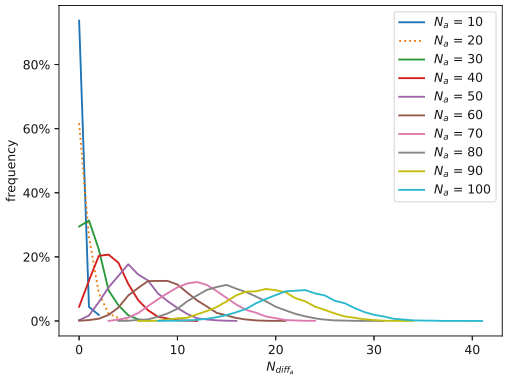

Fig. 3: Empirical distributions of N_{diff_a} for $N_e = 20$.

We gain more information by measuring the *difference in the number of unique neighbors discovered between unequal time windows*. We define this as N_{diff_a}, a sample of the difference in actual neighbors discovered between a window w and a larger w' ($N_{diff_a} = N'_{da} - N_{da}$). The chosen windows w and w' must share the same start time, ensuring that $N'_{da} \geq N_{da}$. While values of $w' \neq 2w$ may be worth consideration, we choose $2w$ for simplicity.

Samples of N_{diff_a} obey a probability distribution with mean and variance based on Ω and N_a (Fig. 3). When $N_a = 10$ and $N_e = 20$, the schedule is parameterized to work for more neighbors than it needs to, and neighbor discovery is likely to discover almost all neighbors in the first window, at the expense of extra energy (e.g., consider the line $N_a = 10$ in Fig. 3,with mean and variance close to zero). Alternatively, e.g., when $N_a = 100$, higher node density increases the mean of the samples since extending the window from w to w' results in novel discoveries. Furthermore, many collisions are unaccounted for, leading to less stable discovery performance and a dramatic increase in the variance of N_{diff_a}.

Sampling N_{diff_a} provides valuable information about the surrounding node density because different schedules met with different surrounding node densities result in uniquely-identifiable distributions of samples in N_{diff_a}. Since multiple such samples are required to characterize the distribution, we capture a sequence s of n samples of N_{diff_a}:

$$(s_i)_{i=0}^{n} = (N_{diff_{a0}}, N_{diff_{a1}}, ..., N_{diff_{an}}) \quad (1)$$

We are concerned primarily with the mean μ and variance σ^2 of (s_i), which indicate whether or not a schedule Ω is accurately estimated, overestimated, or underestimated *in situ*. The last step in deriving a numerical estimate of N_a is to compare the observed distribution to an analytical model that

tells us which values of N_a result in which distributions for a given schedule. If the model suggests that, given the current schedule $\hat{\Omega}$, some value $\hat{N}$ produces a distribution that matches the observed distribution of (s_i), it is likely that $\hat{N} = N_a$. With minor modification, we use D to define an empirical value N_{diff}, the computed difference in number of neighbors discovered between time windows w and w':

$$N_{diff} = D(\Omega, N, w') - D(\Omega, N, w) \quad (2)$$

Since D is derived from a closed-form equation given by the protocol, N_{diff} tells us what the difference in neighbors discovered between windows will be for any schedule Ω (optimal or not) surrounded by any hypothetical node density N. In essence, we can compare the observed difference in (s_i) with the computed difference given by N_{diff} for the current schedule across a range of N to determine what N_a is.

N_{diff} is a powerful analytical tool that connects the observed performance in (s_i) of *any* schedule Ω surrounded by *any* true node density N. We introduce a "schedule topography" which relates Ω, N_a, and N_{diff}, allowing us to derive a numerical estimate for N_a using the current performance of the schedule. The schedule topography reduces the computational complexity of estimating actual node density when implemented on resource-constrained devices since it can be pre-computed and stored in a small table rather than calculated at runtime. It provides a map of the landscape given any combination of schedule Ω and node density N_a.

Detecting Accurate Estimates. Once a node collects a sequence of N_{diff_a} samples in (s_i), we compute the mean μ of (s_i), which represents N_{diff_a} overall. If $\mu \approx N_{diff}$ then $N_e \approx N_a$ and our schedule is accurately parameterized.

Detecting Overestimates. Next we consider the case when $N_e >> N_a$. The schedule quickly discovers a large fraction of the neighbors, overshooting the target discovery probability (and wasting energy). The variance of (s_i), σ^2, will be low since it is unlikely that additional neighbors will be discovered in w'. We introduce a threshold, τ, which varies based on the schedule's N_e. τ can be determined empirically by analyzing the distribution of N_{diff_a} when $N_e = N_a$. We use a threshold τ_{σ^2} for the variance and a separate threshold τ_μ for the mean. When $\tau_\mu > \mu \geq 0$ and $\tau_{\sigma^2} > \sigma^2 \geq 0$, we assume that the actual number of neighbors N'_{da} discovered in w' is equal to N_a and our new estimate is $N_e = N'_{da} = N_a$.

Detecting Underestimates. Values of μ and σ^2 beyond their thresholds signal that Ω is underestimating the number of neighbors. Higher μ implies the protocol is discovering a larger number of neighbors than expected. Higher σ^2 suggests an increase in collisions that makes discovery intermittent. We use N_{diff} to analytically determine a new estimate by inputting Ω into N_{diff} for different N using the w and w' used to collect the samples in (s_i) and note the value $\hat{N}$ that results in a value $N_{diff} \approx \mu$. We use $\hat{N}$ as our new estimate, $N_e = \hat{N} \approx N_a$.

B. Adapting Schedules to Node Density in Existing Protocols

Any neighbor discovery protocol that models collisions can compute a new set of parameters given the application's requirements (e.g., P, Λ, energy budget) and the node density estimated by CANDor. To examine how to adapt a protocol's behavior, we use two exemplar protocols: a slotted protocol, Birthday [6] and a slotless protocol, BLEnd [4]. These choices are partially of convenience, as both analytical models directly reference the surrounding node density.

Birthday. In the slotted Birthday protocol, each node chooses in each slot whether to listen, transmit, or sleep with probabilities p_l, p_t, and p_s, respectively, i.e., $\Omega_{Bday} = [p_l, p_t, p_s]$. The schedule depends on which "mode" the protocol is in; for simplicity, we use the probabilistic round robin (PRR) mode since its schedule is derived using an estimated node density. The three parameters of a PRR schedule are:

$$p_t = \frac{1}{N_e}, p_l = 1 - \frac{1}{N_e}, p_s = 0 \quad (3)$$

The fraction of N nodes discovered over n slots is given as a Poisson distribution dependent on the schedule:

$$F_{d_{Bday}}(\Omega_{Bday}, N, n) = 1 - e^{-np_t p_l^{N-1}} \quad p_t, p_l \in \Omega_{Bday} \quad (4)$$

We can define D_{Bday} using $F_{d_{Bday}}$. Since Birthday is slotted, we express w in terms of slots;we set w to n slotsSsince $F_{d_{Bday}}$ expresses P in terms of n, N_e, and Ω_{Bday}, we solve for n to determine the number of slots required to achieve P:

$$n \geq \left\lfloor \frac{-log(1-P)}{p_t p_l^{(N_e-1)}} \right\rfloor \quad p_l, p_l \in \Omega_{Bday} \quad (5)$$

Adapting Birthday to sensed node density is now straightforward. Using a sequence of samples in (s_i) gathered over n and $n' = 2n$ slots, we sense a new estimate of N_a and compute new schedule parameters using Eqn. 3.

BLEnd. BLEnd uses a slotless scheme in which schedules are parameterized using an epoch length E (the total duration of one period of advertising and listening) and an advertising interval A (the time between beacons). The probability $P_d(\Omega_{BLEnd}, N, k)$ of discovering N neighbors with a schedule $\Omega_{BLEnd} = [E, A]$ over k epochs is given in [4]. Since P_d is intuitively an expression of the fraction of neighbors we expect to discover, we treat it as directly interchangeable with $F_{d_{BLEnd}}$. P_d is expressed in terms of a number of epochs k, so we define k in terms of E and w to derive our window:

$$k = \left\lfloor \frac{w}{E} \right\rfloor \quad (6)$$

We substitute $F_{d_{BLEnd}}$ into D to derive the expected number of neighbors discovered. We sample N_{diff_a} using $w = k$ epochs and $k' = 2k$. Using D_{BLEnd}, we can also compute N_{diff}.

BLEnd uses a brute-force optimization that derives a Ω_{BLEnd} with minimal energy consumption given a target P, Λ, and N_e. By creating such a brute force implementation of the optimization approach in [4], we can simply input the new estimate of N_a sensed by our approach, along with the application's required P and Λ to derive an optimal Ω_{BLEnd}.

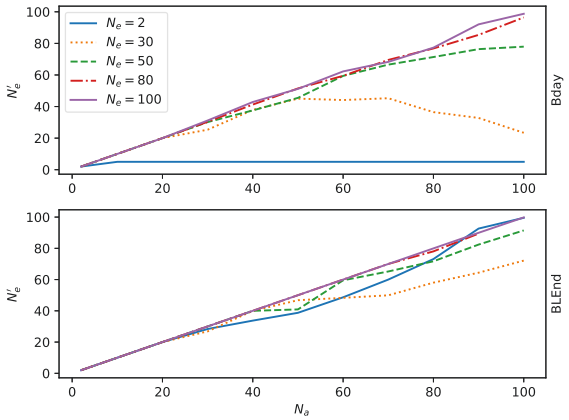

Fig. 4: Sensed node density N'_e versus actual node density N_a using schedules configured for different N_e.

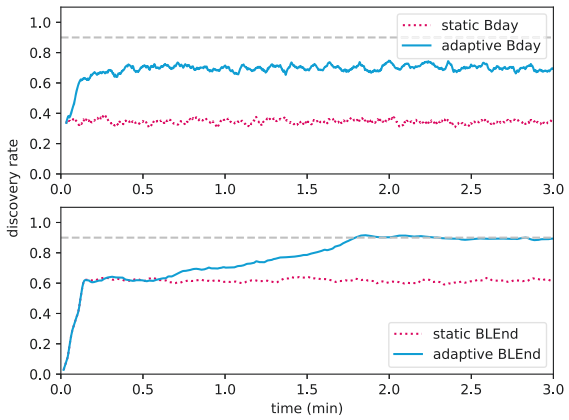

Fig. 5: Static vs. adaptive with initially underestimated schedules and stable surrounding node density ($N_a > N_e$)

V. Evaluation

We evaluate our approach using a simulator implemented in Python using SimPy[1]. Each node has a position in 2D space and implements a neighbor discovery protocol that issues "beacon", "scan", or "sleep" events according to its schedule—discoveries occur if exactly one beacon overlaps with at least one scan, and a collision (i.e., failed discovery) occurs if multiple beacon events overlap. Each node maintains a set of discovery events that include the time and the neighbor discovered. We leverage protocol-specific models to aggregate samples of N_{diff_a}, analyze them, and periodically update the estimate of surrounding node density. CANDor then invokes a protocol-specific function to derive a new schedule.

A. Sensing Node Density

We first evaluate how well CANDor estimates N'_e using schedules parameterized for different N_e. For each (N_e, N_a), we use the protocol's analytical model to determine the optimal schedule. We execute that schedule in a context where the actual number of nodes is N_a and measure a new estimate N'_e. In Fig. 4, we plot N_a vs. N_e. We wait until the sensed estimate N'_e stabilizes and report the average across all nodes. In Birthday, CANDor can estimate N_a with high accuracy given schedules that overestimate or slightly underestimate N_e (Fig. 4). Exceedingly underestimated schedules, however, are less capable of providing accurate estimates because schedules in Birthday never sleep, which produces a catastrophically high number of collisions when N_a grows far beyond N_e. This could likely be addressed by occasionally switching nodes out of PRR and into *birthday-listen* mode [6], which would reduce collisions. In BLEnd, CANDor can estimate N_a with good accuracy given any schedule, overestimated or underestimated. Even for a very low initial estimate of $N_e = 2$, however, we arrive at accurate N'_e across the full range of N_a. Since CANDor's ability to sense node density incurs no additional overhead, sensing and adapting to *any* estimate improves performance over a static schedule.

[1]https://github.com/UT-MPC/dulcet

B. Adaptive Schedules

We next compare static BLEnd and Birthday to adaptive versions, measuring the average discovery rate of all nodes. Both runs start with schedules optimized for highly-inaccurate underestimates ($N_e = 2$ for BLEnd and $N_e = 30$ for Birthday, selected based on Fig. 4 to test the limits of each protocol to adapt from a suboptimal initial schedule) and a target discovery probability of 90%, with an actual node density of $N_a = 80$. Fig. 5 shows that *even with suboptimal initial schedules*, CANDor enables both protocols to adapt to the true node density and achieve superior discovery probability. In Birthday, adaptive performance plateaus at about 70% relative to 40% for the static schedule. Adaptive BLEnd also outperforms its static counterpart, ultimately converging on the target discovery probability of 90% while the static schedule plateaus at around 60%. Note that each node senses the surrounding node density and changes its schedule independently; *even when nodes no longer have the same schedule, all of them still converge on higher-performing configurations.*

C. Dynamic Node Density

We next evaluate CANDor in the presence of node densities that change over time. We first model a scenario where additional nodes are gradually introduced to the collision domain, as may be true in a sensor network or smart space. We start with two schedules—static and adaptive—both perfectly-estimated to the actual initial number of nodes (i.e., $N_e = N_a = 20$) with a target discovery probability of 90%. Every minute, we introduce 10 additional nodes until the final number of nodes is $N_a = 100$. From Fig. 6, we can see that the discovery rate of the static schedule for both protocols suffers a drop every time new nodes enter. In Birthday, the adaptive schedule maintains significantly superior performance relative to the static schedule, which quickly degrades. For BLEnd, the adaptive schedule's performance tracks with that of the static schedule until about the 3 minute mark, at which point it pulls away. The difference in responsiveness between protocols is due to the window of time required to gather

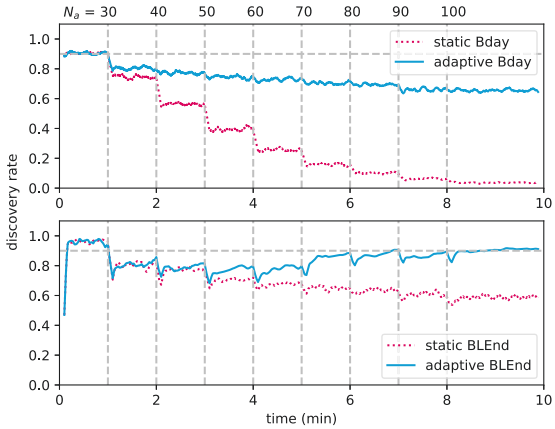

Fig. 6: Static vs. adaptive with accurate initial schedules ($N_e = N_a = 20$) as nodes are added to collision domain

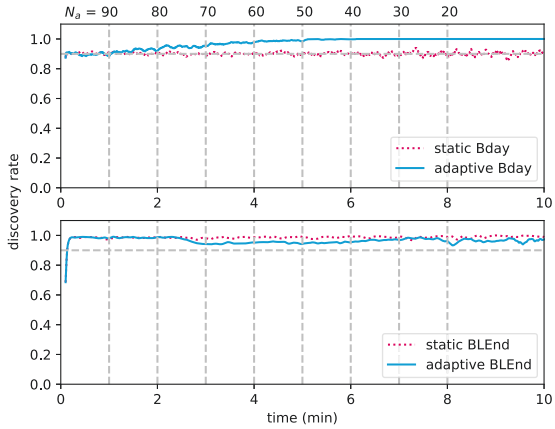

Fig. 7: Static vs. adaptive with accurate initial schedules ($N_e = N_a = 20$) as nodes are removed from collision domain

samples—hundreds of milliseconds for Birthday versus several seconds for BLEnd—in conjunction with the time required for all nodes to independently adapt their schedules.

Our next evaluation demonstrates the effect of node density *decrease* over time, as we iteratively remove nodes rather than adding them. Fig. 7 shows the results. For Birthday, as the node density decreases, the adaptive schedule reconfigures to provide performance above the target. In BLEnd, the static schedule performance stays constant while the discovery rate with adaptation decreases slightly to approach the target. Perhaps counter-intuitively, this is desirable: over-estimated schedules *waste energy*; CANDor adapts to conserve energy.

D. Node Mobility

We next evaluate CANDor's performance in the presence of realistic node mobility using a scenario based on Levy walk mobility [13], which models the dynamics of real human mobility [14]. Each of $N_a = 100$ nodes in our scenario moves for 30 minutes according to this model, resulting in a highly variable number of neighbors over time. We first analyze the results of this scenario from the perspective of a single node before providing a view of aggregate performance.

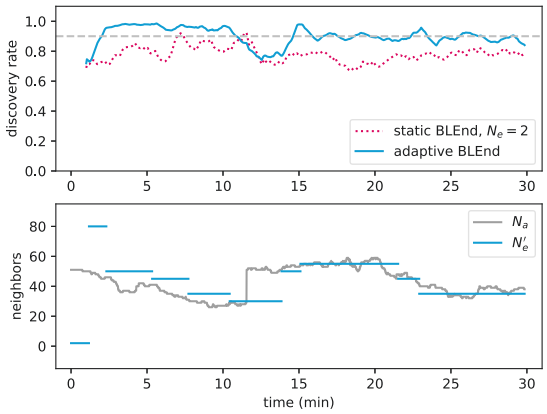

Fig. 8: Static vs. adaptive BLEnd, one node. Top: discovery performance. Bottom: actual neighbors vs. sensed estimate.

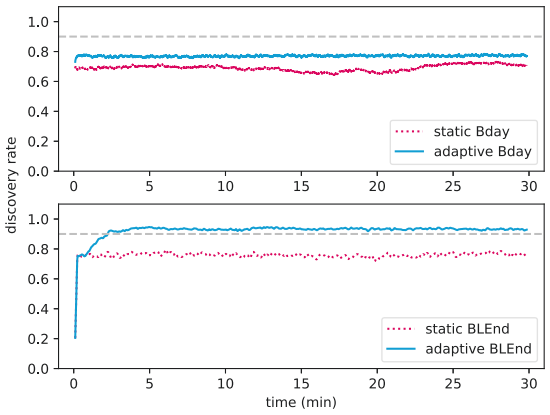

Fig. 9: Static vs. adaptive Birthday and BLEnd with underestimated initial schedules ($N_e = 2$ and $N_e = 30$, respectively)

Adaptation is resilient to small changes in node density. In Fig. 8, for the first 10 minutes, adaptive BLEnd maintains superior performance as the number of neighbors gradually—but noisily—declines from around 50 to 20. Recall that while all nodes *begin* with the same default schedule, their schedules gradually diverge as they independently move in different contexts. Even when nodes have dissimilar schedules, adaptation successfully maintains performance. While we formalized CANDor under the assumption that all nodes have the same schedule, *it is resilient in practice to less-ideal circumstances.*

Adaptation stabilizes after unpredictable leaps in the number of neighbors, as may be the case when an individual in a new location is surrounded by a new "neighborhood" of devices. Around the 12 minute mark in Fig. 8, the adaptive version of BLEnd converges on a better-performing schedule than its static counterpart. A user could expect performance consistent with a designated target within only a few minutes of arriving at a new destination. The Birthday protocol shows a similar performance (figure omitted for brevity).

Adaptation behavior varies with the underlying protocol. CANDor adapts more frequently to node density in Birthday than in BLEnd due to a shorter sensing window. While this

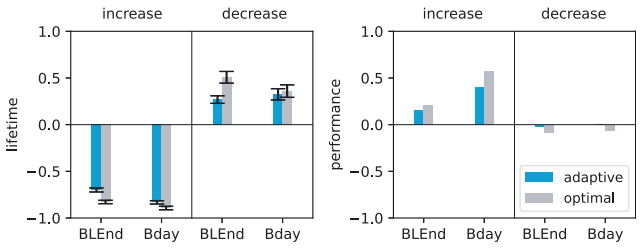

Fig. 10: Battery lifetime and performance of adaptive and optimal schedules, normalized to the values for a static schedule.

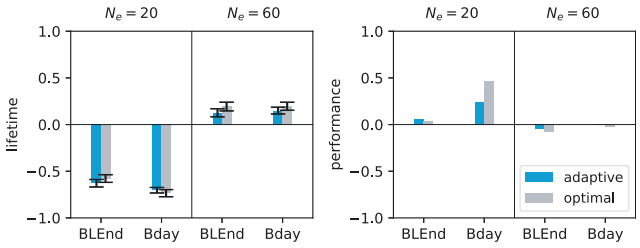

Fig. 11: Battery lifetime and performance of adaptive and optimal schedules, normalized to static during mobility.

may appear ideal for a single node, it causes instability in the broader neighborhood since devices update schedules more frequently. However, across all nodes, adaptation consistently outperforms static schedules for both protocols (Fig. 9).

E. Cost and Benefit Tradeoffs

The application benefits of adaptation come at a cost—schedules able to handle a higher number of collisions also require more energy to ensure a high rate of discovery. Conversely, when the number of neighbors is lower, there is an opportunity to adapt to conserve energy.

We assign an energy cost E, in mAh of instantaneous current draw of a schedule period. We use empirical measurements from a reference device [4] to set the current draw I of transmitting t, listening l, and sleeping s. We find E_{BLEnd} by integrating over samples of I for the radio state at each time step in a schedule period. We define E_{Bday} by weighting the current draw of each radio state by the probability that a slot is spent in that state, $E_{Bday} = I_t p_t + I_l p_l + I_s p_s$.We log every node's actual number of neighbors and the number estimated by CANDor, allowing us to compare the overall energy consumption of the adaptive schedule to an "optimal" one that always perfectly estimates the number of neighbors. We report the normalized difference in battery lifetime between a static schedule and the adaptive and "optimal" ones.

Fig. 10 depicts the results of both the density increase and decrease evaluations from Section V-C. When density increases, the improved performance of adaptation comes at the cost of increased energy (i.e., shorter lifetime). Lifetime is improved over the static schedule with negligible impact on performance when density decreases. If we consider these results in aggregate, they suggest that adaptation in mobile contexts (where node density varies from high to low over time) can make smarter use of available energy, taxing the battery only when necessary to provide the target performance, and opportunistically conserving energy when possible.

Fig. 11 depicts the energy cost and performance benefit comparison for the mobility scenario in Section V-D, relative to static schedules for different N_e. While CANDor *does* successfully exploit opportunities for energy savings, these savings do not fully offset the added cost of adaptation in more dense environments. On the whole, CANDor provides consistently superior discovery performance in dynamic contexts, with a small net increase in energy cost.

VI. Conclusion

We presented CANDor, an approach to continuous adaptive neighbor discovery that leverages signals within a neighbor discovery protocol to adapt schedules to sensed estimates of the surrounding number of devices. Our approach enables neighbor discovery to provide more reliable performance through adaptation *without a need for additional sensing overhead*. We formalized our approach, applied it to two exemplar protocols—slotted Birthday and slotless BLEnd—then showed that CANDor is capable of estimating and adapting to sensed node densities in dynamic operating environments.

Acknowledgements

This work was funded in part by the National Science Foundation under grant CNS-1909221. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] T. Renzler *et al.*, "Improving the efficiency and responsiveness of smart objects using adaptive ble device discovery," in *Proc. of SmartObjects*, 2018, pp. 1–10.
[2] A. Trivedi and D. Vasisht, "Digital contact tracing: technologies, shortcomings, and the path forward," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 4, pp. 75–81, 2020.
[3] P. H. Kindt, T. Chakraborty, and S. Chakraborty, "How reliable is smartphone-based electronic contact tracing for covid-19?" *Communications of the ACM*, vol. 65, no. 1, pp. 56–67, 2021.
[4] C. Julien *et al.*, "Blend: practical continuous neighbor discovery for bluetooth low energy," in *Proc. of IPSN*, 2017, pp. 105–116.
[5] P. H. Kindt *et al.*, "Optimizing ble-like neighbor discovery," *IEEE Transactions on Mobile Computing*, 2020.
[6] M. J. McGlynn and S. A. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proc. of MobiCom*, 2001, pp. 137–145.
[7] Y. Qiu *et al.*, "Talk more listen less: Energy-efficient neighbor discovery in wireless sensor networks," in *Proc. of INFOCOM*, 2016, pp. 1–9.
[8] A. Kandhalu, K. Lakshmanan, and R. Rajkumar, "U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol," in *Proc. of IPSN*, 2010, pp. 350–361.
[9] P. H. Kindt *et al.*, "Griassdi: Mutually assisted slotless neighbor discovery," in *Proc. of IPSN*, 2017, pp. 93–104.
[10] C. Drula *et al.*, "Adaptive energy conserving algorithms for neighbor discovery in opportunistic bluetooth networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 96–107, 2007.
[11] A. Hess, E. Hyytiä, and J. Ott, "Efficient neighbor discovery in mobile opportunistic networking using mobility awareness," in *Proc. of COMSNETS*. IEEE, 2014, pp. 1–8.
[12] S. Firdose *et al.*, "CRAWDAD dataset copelabs/usense," Downloaded from https://crawdad.org/copelabs/usense/20170127, Jan. 2017.
[13] I. Rhee *et al.*, "On the levy-walk nature of human mobility," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 630–643, 2011.
[14] Z. Cheng *et al.*, "Exploring millions of footprints in location sharing services," in *Proc. of ICWSM*, vol. 5, no. 1, 2011, pp. 81–88.