PINCH: Self-Organized Context Neighborhoods for Smart Environments

Chenguang Liu
The University of Texas at Austin
Austin, Texas, USA
Email: liuchg@utexas.edu

Christine Julien
The University of Texas at Austin
Austin, Texas, USA
Email: c.julien@utexas.edu

Amy L. Murphy
Fondazione Bruno Kessler
Trento, Italy
Email: murphy@fbk.eu

***Abstract*—Today's "smart" domains are driven by lightweight battery operated devices carried by people and embedded in environments. Many applications rely on *continuous neighbor discovery*, i.e., the ability to detect other nearby devices. Application uses for neighbor discovery are widely varying, but they all rely on a protocol in which devices exchange periodic *beacons* containing device identifiers. Many applications also ultimately involve assessing and adapting to *context* information sensed about the physical world and the device's situation in that world (e.g., its location or speed, the ambient temperature or sound, etc.). In this paper, we define Proactive Implicit Neighborhood Context Heuristics (PINCH), which leverages unused payload in periodic neighbor discovery beacons to opportunistically distribute context information in a local area. PINCH's self-organizing algorithms use limited local views of the state of a one-hop network neighborhood to determine the most useful type of context information for a device to sense and share. In this paper, we develop the algorithms, integrate an implementation of PINCH with a smart city simulator, and benchmark the tradeoffs of self-organized local context sharing with 2.4GHz neighbor discovery beacons.**

***Keywords*-Context sharing; Device-to-device coordination**

I. Introduction

In applications for smart cities, smart homes, smart transportation, etc., mobile and situated devices often need to discover other devices in the surroundings. A user entering a smart building needs to discover what embedded devices are available to be accessed or controlled. A tourist in a smart city may need to connect to other tourist's devices [1] or to situated beacons in the city that give information about interesting sights [2], [3]. Devices in smart cars may coordinate with other cars, roadside kiosks, or pedestrians. In smart wildlife applications, devices on wild animals discover one another to monitor behavior patterns [4]. Enabling these applications requires *continuous neighbor discovery*, and protocols for neighbor discovery have received significant attention, from wireless sensor networks to smart-* applications [5]–[13].

These neighbor discovery protocols simply exchange identifiers of one-hop neighbors; how to coordinate with discovered neighbors is left to applications. Some application-specific approaches use the beacon payload to carry additional information, for instance to give a multi-hop view of a *group* [1], [14]. However, many applications that rely on continuous neighbor discovery also demand a view of local *context information* that goes beyond identities of neighboring devices. For instance, many applications use physical location. Smart building applications may use ambient temperature or lighting to adapt behavior; smart wildlife applications may correlate ambient information to animal contacts; smart tourism applications may use information about crowds or nearby available services.

The lightweight devices common in these applications often lack on-board sensors. Even if these capabilities are present, leveraging them continuously can be energy intensive. However, given the presence of other devices in the surroundings and the fact that context is often correlated to a device's physical location, the burden of sensing context could be shared within a network neighborhood. This can reduce the energy burden of context sensing for individual devices as well as extend the capabilities of sensor-limited devices. We explore allowing devices to use available payload in periodic neighbor discovery beacons to opportunistically share sensed context in the local network.

Many approaches support resource-efficient context sensing by intelligently tasking on-board sensors based on applications' high-level needs [15], [16]. Using co-located devices to extend a device's sensing capabilities has also been explored. ChitChat [17] supports lightweight sharing of complete snapshots of devices' contextual situations. Other approaches allow devices to discover and leverage sensors and I/O capabilities on neighboring devices [18], [19]. As described in more detail in Section II, these approaches are all driven explicitly by applications' requests for context information. Instead, we take a self-organizing approach to proactively infer what context information might be useful to others in the network neighborhood.

We develop PINCH (Proactive Implicit Neighborhood Context Heuristics), which assumes that devices in smart-* deployments participate in continuous neighbor discovery. In these protocols, detailed in Section II, there is often unused payload in the periodic neighbor discovery beacons, and PINCH packs valuable context information into this unused payload, constructing a sort of neighborhood-wide sensor. We build PINCH on the BLEnd protocol, directly considering how aspects of BLEnd influence our self-organizing algorithms for context sharing. Section III describes these algorithms in detail, incrementally constructing algorithms that rely on the minimal information that can be shared in the periodic beacons. As such, PINCH is: (1) *self-organizing*, i.e., individual devices make individual decisions in a heuristic approach that attempts to optimize for the local neighborhood; (2) *implicit*,

i.e., devices do not explicitly request or respond to requests for particular sensed values; and (3) *inexpensive*, i.e., the approach entails no additional communication overhead beyond what is already consumed by neighbor discovery. Our contributions are:

- We describe PINCH, a self-organizing heuristic for implicitly sharing context using neighbor discovery beacons.
- We derive algorithms for deciding what context type a device should share, given the state of the neighborhood.
- We evaluate PINCH using an expressive smart city simulator in with real-world application scenarios.

Our evaluation demonstrates that PINCH increases the coverage of context information in the local network neighborhood when devices have limited sensing capabilities and reduces the overall cost of context sensing in cases when devices have overlapping redundant sensing capabilities.

II. Background and Related Work

We begin with a concise survey of related work on local and collaborative context sensing, showing the gap that PINCH addresses. We then overview *continuous neighbor discovery*, detailing BLEnd, which PINCH relies on.

A. Related Work

Efficiently acquiring context is essential in smart-* applications that rely on lightweight battery-operated devices. A wealth of approaches solve this problem for a single device, e.g., leveraging on-device sensors to monitor high-level context *changes* rather than blindly sensing raw values [15], [16], [20]–[22]. While these approaches consider how best to use on-board resources to determine the lcoal context state, we broaden the perspective and ask how wirelessly connected devices can leverage their resources in aggregate.

Several early works in wireless sensor networks offered the ability to share context information among co-located devices [23]–[26]. However, these systems explicitly spread context sensing capabilities to neighboring devices, then responded to one-time queries or established persistent queries to receive changes in context values. In contrast, PINCH incurs no cost above that of continuous neighbor detection and works seamlessly with changing neighbor sets.

Work addressing user-facing applications has promoted the use of device-to-device links to exchange context. ChitChat [17] packages an application-specified view of a device's context in a lightweight data structure to disseminate locally. It focuses on compact data packaging and does not consider how or when that information is disseminated, nor what context may be most useful in the neighborhood. Other approaches allow devices to discover sensing or actuaion resources on neighboring devices then to request those discovered resources [18], [19]. Our work is complementary; instead of assuming an application-directed request and response, we opportunistically self-organize the network neighborhood into an aggregate context sensor based on presumptions of needed context information. The key enabler of our approach is that the communication of the context values comes virtually *for free*, under the assumption that devices are already participating in a local continuous neighbor discovery protocol.

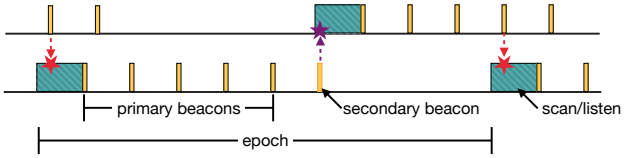


Figure 1: BLEnd Schedule of two devices. Solid rectangles are scanning (listening) periods; skinny rectangles are beacons. Discovery occurs when one device receives another's beacon. Beacons are scheduled in the second half of an epoch as a result of receiving a *primary beacon*; these *secondary beacons* enable bi-directional discovery.

B. Neighbor Discovery and BLEnd

Neighbor discovery enables devices to discover other devices within communication range. We focus on an entirely infrastructureless solution in which each device plays both sides of the discovery role: announcing its presence and scanning for the presence of others. Many neighbor discovery approaches emerged from wireless sensor network research; early efforts provided detection probabilities with a long-tailed distribution [10] but did not *guarantee* discovery, even in the absence of collisions or other channel-related unreliability. More recent efforts [5], [6], [8], [11]–[13] provide deterministic detection when communication is perfect. These approaches rely on communication occurring in fixed-length slots, but this assumption is incompatible with many off-the-shelf beacon technologies, which introduce randomness to mitigate the impact of collisions. As a result, when considering collisions, discovery is not guaranteed. Recent protocols provide slotless behavior [9], compatible with modern technologies e.g., Bluetooth Low Energy (BLE).

BLEnd [7] is a continuous neighbor discovery protocol designed to work within the constraints of BLE and to offer a clear service-level agreement for probabilistic discovery latency guarantees with minimal radio active time. In BLEnd, time is divided into repeating epochs. At the beginning of each epoch, the device listens for a specified period. It then sends a sequence of beacons, enabling other devices to discover it. The BLEnd schedule allows the radio to remain inactive for a significant period of time, dramatically reducing energy consumption. Figure 1 shows an example of two devices' BLEnd schedules.

The optimal BLEnd schedule is based on the desired service-level agreement, including: (1) desired discovery latency (Λ); (2) desired probability of discovery (p_d); and (3) beacon technology details (e.g., energy costs of beaconing and listening, etc.). The generated schedule includes the length of an epoch, the length of a listen, and the time between beacons. It minimizes the protocol's

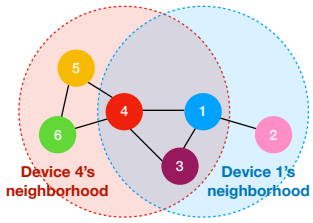


Figure 2: Asymmetric neighborhoods defined by BLEnd beacon exchange. Device 1 exchanges beacons with 2, 3, and 4; device 4 exchanges beacons with 1, 5, and 6.

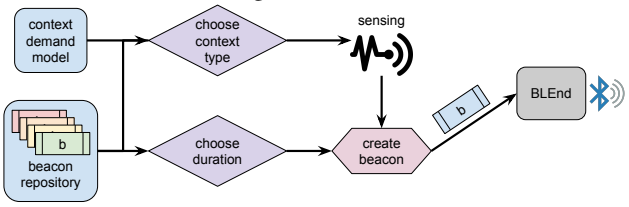


Figure 3: Overall PINCH Operation

energy consumption while guaranteeing that a device will discover any neighbor within Λ time with a probability of p_d.

Devices employing BLEnd receive beacons of neighboring devices; i.e., devices in one-hop communication range, identifying the device's one-hop neighborhood. Figure 2 shows two, asymmetric neighborhoods where devices 1 and 4 are both in each other's neighborhood but each neighborhood contains devices that the other's does not.

III. The PINCH Approach

PINCH supports applications that rely on context information sensed about the device's environment. We assume each piece of context information has a *type* and that these types are generic. Example context types include physical location, temperature, speed, sound level, crowd levels, etc.. PINCH is predicated on the assumption that the sensors on co-located devices commonly have correlated values for a given sensor type.

PINCH exploits the fact that neighbor discovery must use fixed-length beacons that are larger than required for neighbor discovery. It leaverages the unused space to allow a neighborhood of devices to *self-organize* into an aggregate context sensor in which each device independently determines a context sharing task that supplies situational context to other devices in the neighborhood. Devices thereby share the burden of context sensing, resulting in lower aggregate energy usage and higher coverage of hard-to-sense context types. In PINCH, individual devices make individual decisions to provide aggregate benefit to the network neighborhood. These decisions are informed by limited information shared in neighbor discovery beacons. Figure 3 shows the overall operation of PINCH.

We assume that the set of all possible context types is of size k and known *a priori* to all devices. A device's sensing capabilities is a subset of these types represented as a k-bit vector in which each bit indicates whether the device can provide the context type indicated by that index. We refer to this as a device's *context capability vector* (*cap*). We assume that each device is participating in BLEnd neighbor discovery.

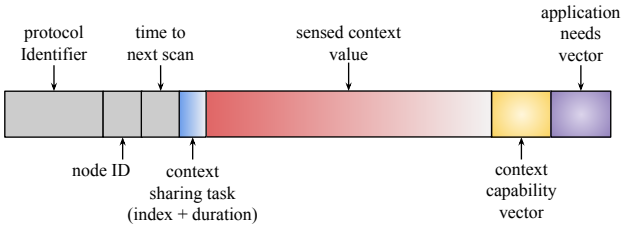


Figure 4: Basic Beacon Payload

A device stores the most recently received beacon from each neighboring device in a *beacon repository*; PINCH removes a device's beacon from this repository when the device moves out of range. Information from received beacons is combined with a *context demand model* that captures neighboring devices' needs for context. The context demand model is either static and known *a priori* or built from information in beacons. Together the beacon repository and context demand model serve as inputs to two algorithms. The first determines what *type* of context this device should sense and share, while the second determines the length of time for this sensing and sharing task, after which the device reevaluates its context sharing activity. Based on the outputs of these algorithms, the device constructs the BLEnd beacon.

A. Beacons for Self-Organized Context Sharing

In the BLEnd beacon of Figure 4, gray elements are used by the BLEnd protocol and treated as *header* data. We assume PINCH can access to data but may not modify it as doing so would change neighbor discovery behavior.

To support self-organized context dissemination in PINCH, we add four components to each beacon b, stored in the beacon repository $\mathcal{B}$. The first new component describes the sending device's selected context sensing task (i.e., the result of the algorithms in Figure 3), which itself has two components: the type of context shared in this beacon, $b.type$ (an index into the context capability vector, which requires $\lceil \log k \rceil$ bits) and a *duration* of this selected task, counted in BLEnd discovery periods (i.e., increments of Λ), $b.tts$. The latter indicates how much longer this device will make this type of context available to its neighborhood, making it a kind of *countdown* timer, decreasing each Λ. When the value reaches 0, the device reevaluates its context sharing task. The beacon also contains two k-bit vectors: the sending device's context capability vector $b.cap$, and (optionally) the context needs of the applications on the device, $b.a$. The latter, if included, can contribute to the construction of the neighborhood's context demand model, described in Section III-B. The remaining payload (the red portion in Figure 4) is used for the value of the sensed context. This can be a simple or composite value, e.g., carrying additional data such as error or units.

B. Neighborhood Context Demand Model

Our algorithms use a *neighborhood context demand model*, which expresses the importance of each of the k context types. We start our investigation with a demand model that is fixed and known to all devices *a priori*. We then build a dynamic model that captures the instantaneous requirements of neighboring devices by examining the information that those neighboring devices include in their beacons. Finally, we specify an *egocentric* model in which each device assumes others have needs similar to its own.

In all cases, we assume that the model assigns each context type $c \in [1..k]$ a *weight* w_c. A higher value of w_c indicates a higher (relative) importance of context type c. For simplicity, we constrain w_c to the interval $[0, 1]$, and require the sum of all k w_c values to be one, i.e.: $\sum_{c=1}^{k} w_c = 1$.

1) A Static Context Demand Model: In the simplest case, all devices can assume a context model in which all context types are equally important. To achieve this, we simply set $w_c = 1/k$ for all c. An alternate static context demand model could use a look-up table shared *a priori* among all devices, mapping each context type to a fixed weight. In any case, sharing the context demand model with all devices makes neighbor behavior more predictable.

2) A Dynamic Context Demand Model: Because PINCH beacons can include the sending device's application needs vector, the context demand model can respond to the network neighborhood's changing context demands. For instance, to express a context demand model in which the weight of a type is proportional to the number of neighboring devices that require that type, we use:

$$w_c = \frac{\eta_c}{\sum_{i=1}^{k} \eta_i} \tag{1}$$

where η_i is the number of devices in the neighborhood that require context i. We compute η_c using the application needs carried in beacons (Figure 4) as $b.a$. Specifically:

$$\eta_c = \sum_{j=1}^{n} |j.b.a \ \& \ 2^c| \tag{2}$$

We use the notation $|\mathcal{V}|$ to denote the *Hamming weight* of the bit vector $\mathcal{V}$, i.e., the number of bits in $\mathcal{V}$ set to one.

Eq. 1 is a general form of the basic static context demand model in which all types are equally important. In fact, this context demand model is quite flexible. For instance, if one wishes to define a more tailored *static* model, this can be done simply by assigning integer importance values to each of the k context types, then using those values in place of η_c in Eq. 1 to generate normalized context weights w_c.

3) An Egocentric Context Demand Model: An alternative dynamic model that can be employed even without sending $b.a$ in the beacons uses a local devices' context needs as a proxy for the neighborhood's needs. This reserves additional space in the beacon for context data. More importantly, however, it ensures that a device does not share context values that its applications' will not directly use. This reduces the *altruism* of the device (i.e., it does not perform context sensing that is not also locally useful) but also decreases the energy burden for the device.

C. Selecting a Context Type to Share

We now turn our attention to the context selection algorithms, which deal with selecting a context type to share using information received in other devices' beacons.

1) Basic Greedy Algorithm: To begin, consider an algorithm that relies on just the first two components of all of the other devices' beacon payloads: the sharing task description and the sensed context value.

This algorithm first looks the sharing tasks selected by neighbors to determine uncovered context types in the local neighborhood. By examining the $type$ in each received beacon, a device determines the aggregate of its neighborhood's context tasks using a multi-way bit-wise OR:

$$\mathcal{T} = \bigvee_{j=1}^{n} 2^{j.b.type} \tag{3}$$

where $\mathcal{T}$ is a bit vector, indexed in the same way as the beacons' cap vectors. Recall that neighborhoods are asymmetric; the context neighborhood of a device contains the devices from which beacons are received. To compute $\mathcal{T}$, we use each beacon's $type$ as an index into the capability types (i.e., $2^{j.b.type}$). Intuitively, $\mathcal{T}$ represents the context types that are "covered" within the device's local neighborhood.

We start by computing the negation of this aggregate neighborhood context to generate a bit vector indicating context types that are *not* currently shared by any neighboring device. When ANDed with this device's capability vector, cap, we are left with types this device could sense and share that would add to the neighborhood's covered types:

$$\mathcal{S} = \neg\mathcal{T} \ \& \ cap \tag{4}$$

Conceptually, $\mathcal{S}$ identifies sensing *gaps* in the local neighborhood. The device chooses the uncovered type of greatest importance as indicated by the context demand model, i.e., $s = \langle \max c : c \in \mathcal{S} :: w_c \rangle$[1]. The device then senses and shares the value of type s for the next Λ time, i.e., for a period of time equal to the BLEnd discovery latency.

Choosing to sense and share the value for Λ ensures that all devices within discovery range will receive the context value with a probability equal to BLEnd's discovery probability, i.e., p_d. Concretely, in the device's created beacon, the sharing task description's first value will indicate the index s (the selected type) and a task duration of 1Λ.

As an alternative equivalent definition of s, we specify a value $P(s)$ for each context type s:

$$P(s) = \begin{cases} 1, & \text{if } \langle \max c : c \in \mathcal{S} :: w_c \rangle = w_s \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where $P(s)$ indicates the probability that the selected context sensing task is to share context type s. In this

[1]We use the shorthand $c \in \mathcal{S}$ to denote the fact that the index of c carries a one in the bit vector $\mathcal{S}$; i.e., $c \in \mathcal{S} \leftrightarrow (\mathcal{S} \ \& \ 2^c = 2^c)$.

simple case, only the uncovered context type with the max weight will have a value $P(s) = 1$; all other probabilities will be 0. However, this generic formulation of the problem enables straightforward extensions of the selection algorithm.

2) Randomizing the Choice: When all devices are working from the same (or similar) context demand model, two devices with the ability to provide the same (important) context type may choose to cover the same thing. This is especially likely for two devices that are not in each other's neighborhoods but are connected to a shared neighbor. Our next refinement adds a small amount of randomness to the choice of type to share, while still giving a weighted preference to more important types according to the context-demand model. In particular, we use a function for $P(s)$ that assigns a probability of selecting s proportional to the weight w_s from the context demand model:

$$P(s) = \begin{cases} \frac{w_s}{\sum_{c \in \mathcal{S}} w_c}, & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

where uncovered types are assigned a non-zero probability; context types that are either covered in the neighborhood or that the device cannot sense are assigned a probability of 0.

This strategy can also be employed when there are no sensing gaps in the neighborhood (i.e., $|\mathcal{S}|$ is 0). Because the device will send neighbor discovery beacons anyway, it is productive to include some context information. Such a device can choose a context type according to the context demand model, considering all of its capabilities:

$$P(s) = \begin{cases} \frac{w_s}{\sum_{c \in cap} w_c}, & \text{if } s \in cap \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

3) Rarity-Weighted Algorithms: When devices have widely varying sensing capabilities, some capabilities might be much more rare than others. In these situations, a device with a rare capability should favor it over others with higher weights in the context demand model, especially when those more common types can be covered by other neighbors.

Using the capability vectors in received beacons, each device can compute how common each of its capabilities is in the neighborhood. We refer to this as a type's prevalence:

$$prev(c) = \frac{\sum_{j=1}^{n} |(j.b.cap \,\&\, 2^c)|}{n} \quad (8)$$

The value of $prev(c)$ is between 0 and 1 and captures the fraction of neighborhood devices capable of sensing type c. To consider only the rarity of a context type in selecting the device's sensing task, we compute $P(s)$ as:

$$P(s) = \begin{cases} \frac{1-prev(s)}{\sum_{c \in \mathcal{S}} (1-prev(c))}, & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

Combining the prevalence and weight from the previous refinement, $P(s)$ becomes:

$$P(s) = \begin{cases} (1-\alpha)\left(\frac{w_s}{\sum_{c \in \mathcal{S}} w_c}\right) \\ \quad +\alpha\left(\frac{1-prev(s)}{\sum_{c \in \mathcal{S}} (1-prev(c))}\right), & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

where α balances the degree to which the algorithm favors rarity vs. simply selecting an uncovered type. This strategy can also be employed even when there are no sensing gaps by replacing $\mathcal{S}$ with cap in the definitions of $P(s)$.

D. Selecting a Duration for Sharing

The above selection algorithms assume that a device selects the type of context to share and then shares that value for exactly one Λ time, after which the device reevaluates the situation, potentially selecting a different context sharing task for the next Λ period. However, Λ might be relatively short (especially relative to a neighborhood's contextual dynamics), and it may be reasonable for a device to choose a context type to share for a longer duration. Further, reevaluating the context sensing choice frequently could lead pairs or groups of devices to swap sensing tasks every Λ time; because context neighborhoods are asymmetric, these changes then ripple through the adjacent neighborhoods. Further, initializing a particular sensor (e.g., a GPS unit) may have overhead, so this switching may be quite expensive. We therefore allow devices to select a *multiple* m of Λ as the duration of a sharing task.

Intuitively, a device should reevaluate its sensing task less frequently if the capabilities and needs in the neighborhood change infrequently. To account for this, we make the selected task duration proportional to the neighborhood's dynamics. That is, we compare the neighborhood's capabilities at time t to those that were available at time $t - \Lambda$.

Similar to the computation of $\mathcal{T}$, we compute the aggregate (unweighted) capabilities of a device's neighborhood:

$$\mathcal{C} = \bigvee_{j=1}^{n} j.b.cap \quad (11)$$

where $\mathcal{C}$ is a bit vector of length k, indexed in the same way as $\mathcal{T}$ and the devices' cap vectors. We extend this notation slightly to account for time; $\mathcal{C}(t)$ indicates the aggregate capabilities in the neighborhood at time t. We define the *distance* between $\mathcal{C}(t)$ and $\mathcal{C}(t - \Lambda)$ to be the Hamming weight of $\mathcal{C}(t) \oplus \mathcal{C}(t - \Lambda)$:

$$distance = |(\mathcal{C}(t) \oplus \mathcal{C}(t - \Lambda)| \quad (12)$$

We then use $distance$ to determine the duration multiple for the selected context task. Because we want smaller values of *distance* to result in larger values of m; in particular, we could, for example define m as: $m = \max(2^d - distance, 1)$, where d is the number of bits allocated to the *duration* field in each beacon. This is conservative, favoring $m = 1$ in cases with any significant change in the neighborhood's capabilities. Alternatively,

we can more evenly distribute the duration selection among the possible choices:

$$m = \begin{cases} (2^d - 1), & \text{if } distance = 0 \\ \lfloor (2^d - 1) \times \left(1 - \frac{distance}{k}\right) \rfloor + 1, & \text{otherwise} \end{cases} \quad (13)$$

E. Collision-Aware Context Task Selection

Because BLEnd is probabilistic, devices receive beacons from neighbors every Λ interval only with a probability equal to the p_d in the BLEnd service-level agreement. Up to this point, our algorithms have not considered that this directly implies that a beacon only covers the selected context task (for a given neighbor) with probability $(1 - p_d)$.

In this section, we examine a final selection algorithm that accounts for this. We rely on a target *probability of coverage* (p_c) for each context type c. If $p_c = p_d$ (i.e., the target probability of coverage is the same as BLEnd's probability of discovery), then we probabilistically achieve the target in every Λ interval. However, when $p_c > p_d$, multiple devices in the neighborhood must share the same context type in order to achieve the target probability.

First, we redefine $\mathcal{T}$. Instead of computing a boolean coverage for each type, we determine *how many* neighbors are providing each type. We define $\mathcal{T}_{coll}$ to be a vector whose digits are base 2^n, where n is the number of neighbors. This prevents carries when summing bit vectors representing each beacon's context type. With this formulation, we compute:

$$\mathcal{T}_{coll} = \sum_{j=1}^{n} 2^{j.b.type} \quad (14)$$

and we refer to the count for a particular context type c as $\mathcal{T}_{coll}[c]$. Next, we estimate the neighborhood's achieved coverage probability using the counts in $\mathcal{T}_{coll}$:

$$p'_c = 1 - (1 - p_d)^{\mathcal{T}_{coll}[c]} \quad (15)$$

where p_d is the BLEnd neighbor discovery probability.

We compute the intermediate $\mathcal{T}^*_{coll}$ as:

$$\mathcal{T}^*_{coll}[c] = \begin{cases} 1, & \text{if } p'_c < p_c \\ 0, & \text{otherwise} \end{cases} \quad (16)$$

Finally, $\mathcal{S}_{coll}$ captures the context types that this device is capable of sensing and whose estimated probability of coverage (p'_c) has not reached the target (p_c):

$$\mathcal{S}_{coll} = \neg \mathcal{T}^*_{coll} \,\&\, cap \quad (17)$$

and used in the algorithms in Section III-C in place of $\mathcal{S}$.

IV. PINCH Implementation

PINCH builds directly on BLEnd, whose existing implementation relies on Bluetooth Low Energy (BLE) for beacon exchange. We are therefore constrained by the BLE advertisement PDU for the beacon, which has 31 octets of application-writeable data [27]. BLEnd [7] uses five octets for identifying the BLEnd protocol, two octets for carrying a unique node identifier, and two octets to announce the time to the start of the node's next epoch. This leaves 22 octets of writeable data that is unused by BLEnd.

We assume a maximum of 32 context types (i.e., $k = 32$) and allocate four octets to a beacon's context capability vector (and to its application needs vector, when it is included). We use another octet for the context sharing task: five bits as an index into the capability vector and three bits for the duration countdown, yielding a maximum duration of 7Λ. Finally, we allocate the remaining space (ten octets when the application needs vector is included) to the context data. The context data is formatted in a type-specific way, but this space is sufficient for commonly encountered context types in smart-* applications. For instance, while many representations of location (i.e., latitude/longitude pairs) use 16 octets, more recent lightweight devices use just six octets, resulting in $\sim 2m$ accuracy. In other cases, some of the space allocated to the context data may be used to provide meta-data, e.g., sensor precision or data freshness.

V. Evaluation

To benchmark PINCH and demonstrate its applicability, we use a custom smart-city simulator, based on the OMNeT++ v.5.2 discrete event simulator[2]. The wireless physical and MAC layers are based on the INET Framework v.3.6.2[3]. We integrated support for geographic coordinates and 3D graphics using Open Scene Graph[4] and Open Street Map[5]. The algorithms are implemented using the Eigen3 library[6]. We used a map of pedestrian-friendly areas of Trento, Italy, and modeled devices moving on streets at walking speeds (i.e., two meters per second) as if carried by people. To realistically model the impact of the urban environment on wireless communication, the simulator computes the influence of building material properties on radio signals.

We configure the BLEnd parameters in Section II to target one-hop network neighborhoods with around 10 devices, i.e., $n = 10$ in BLEnd. We set a target discovery probability of $p_d = 0.9$ and a target discovery latency of $\Lambda = 10s$. We use the beacon specifications of the TI SensorTag (www.ti.com/sensortag), an inexpensive sensing device with a complete BLE radio and networking stack that is representative of many IoT devices. The resulting optimal settings for BLEnd dictate an epoch length of 9.995s, with a listen duration of 217ms at the start of every epoch and a beacon interval of 204ms (Figure 1).

A. Benchmarking the Algorithms

We measure the *coverage percentage*, i.e., the percentage of types a device receives relative to its needs:

$$\text{coverage percentage(c)} = \frac{\#\text{ covered demands for } c}{\text{total } \#\text{ demands for } c} \quad (18)$$

[2]https://www.omnetpp.org/
[3]https://inet.omnetpp.org/
[4]http://www.openscenegraph.org/
[5]http://www.openstreetmap.org/
[6]http://eigen.tuxfamily.org/

Because some types have more value in the neighborhood than others, i.e., they have higher w_c values, we also compute a weighted *coverage quality*:

$$\text{coverage quality} = \sum_{c=1}^{k} \left[\left(\frac{\#\text{ covered demands for } c}{\text{total }\#\text{ demands for } c} \right) w_c \right] \quad (19)$$

We define a baseline scenario that controls the size of each device's one-hop neighborhood to isolate the impact of parameters to our algorithms. A group of (ten) devices starts together at one side of the city and follows a shared trajectory through the city streets[7]. As the devices navigate the urban space, they can become transiently disconnected, e.g., because of building obstructions near corners. This scenario is representative of many smart-city applications, e.g., those that support groups of tourists [1], school children, family members, or friends moving together.

1) Context Sharing for Improved Context Coverage: Figure 5 shows the coverage percentage for each context type with various parameters. Figures 5(a) and (b) use relatively few possible context types (i.e., $k = 10$). All devices demanded values for all types (i.e., a *demand ratio* (DR) of 100), and every device could sense a randomly selected 20% of the types (i.e., a *capability ratio* (CR) of 20). We show the coverage percentage for both greedy (Eq. 5) and randomized selection (Eq. 6). Figure 5(a) employs the static context demand model in which all context types are equally important (Eq. 1); Figure 5(b) employs a static demand model in which weights decay linearly for higher indexed types (i.e., $w_1 = 0.22$, $w_2 = 0.19$, ... $w_8 = 0.03$). The results are nearly identical because a device's neighborhood almost always has ten devices (including the device itself), and each device can choose a different one of the ten sensing tasks; PINCH implicitly fosters this diversity since a device uses the neighbors' sensing tasks to inform its own selection.

In Figures 5(c) and (d), we increase k to 20. In Figure 5(c), all context types are equally important. The greedy algorithm favors lower indexed types (it chooses greedily), while the randomized algorithm spreads coverage more evenly. In Figure 5(d), where we employ a linearly decaying demand, the demand model is apparent in the trend for the randomized algorithm, whose selection of types to share follows the weights in the model.

In Figures 5(e) and (f), we employ a dynamic context demand model. In Figure 5(e), every device selects a random subset of 20% of the context types it needs (i.e., DR $= 20$); in Figure 5(f), we increase this demand to 60% of the types. In both cases, CR $= 20$. In Figure 5(e), the simple greedy algorithm is unable to adjust when the capabilities are constrained; this is further highlighted in Figure 5(f), where the context demands are even higher, and the greedy algorithm favors lower indexed context types, while the randomized selection algorithm provides more uniform coverage.

2) Increasing Context Types: We next evaluated PINCH's ability to handle a growing number of context types, since smart environments may have many types of sensors or high-level context abstractions. Using the same set of ten devices moving together and the same demand and capability models as in Figure 5(f), we varied the number of context types (k) from 8 to 32. Figure 6 shows the average *coverage quality* for the greedy and randomized algorithms. PINCH maintains high coverage quality, even with many more context types than sensing-capable neighbors. The randomized algorithm performs better because it chooses from across all possible types instead of focusing greedily on (only) the most important.

3) Adapting to Dynamic Needs: PINCH is designed to adapt to the neighborhood's changing needs and capabilities. To assess this adaptation, we used our group mobility model with ten devices, with $k = 16$, and with capability and demand ratios both of 20%, where the demanded types were not necessarily the same as the capabilities. Periodically, in this case, every 30 seconds, every device randomly re-selects its needed context types. We also statically set the duration of the context to either the minimum possible (1Λ) or the maximum possible (7Λ). Figure 7 shows the coverage percentage in both cases. With a shorter duration, PINCH is able to achieve a slightly higher coverage, but, as evidenced by the sharp peaks, the devices are switching context sensing tasks more frequently. This figure demonstrates the tradeoff between sensing stability and sensing coverage; while a higher coverage is obviously preferred, as described previously, there is overhead to activating a context sensor that also increases the energy cost of sensing.[8]

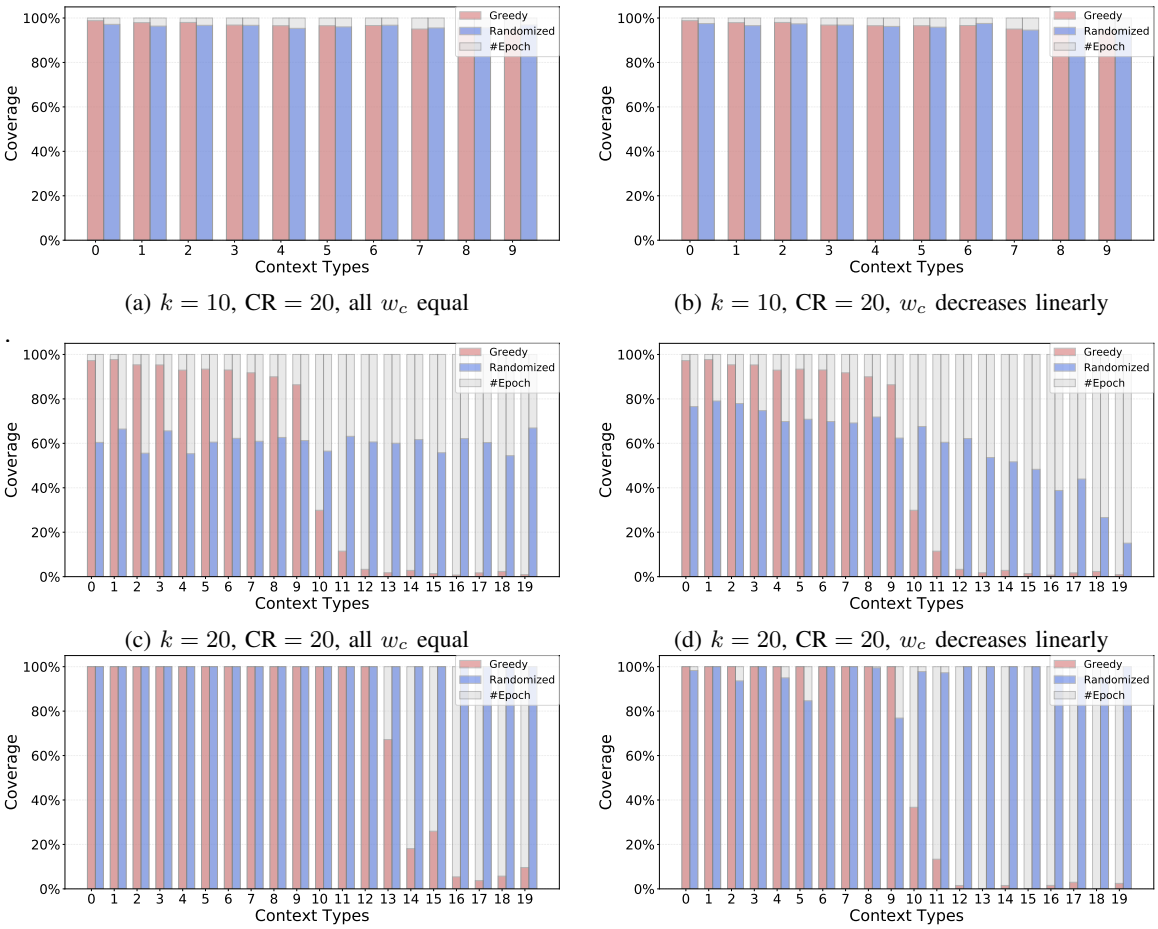
4) Benefits and Costs: PINCH uses the neighborhood's sensing capabilities to fulfill sensing tasks a device is incapable of or simply to leverage energy used for sensing on nearby devices. We measured the *benefit* and *cost* of sharing the sensing task. For the benefit, we compute the percentage of a device's needed types that were covered by some other device. For the cost, we compute the percentage of the time that the context value a device was sharing was something the node itself did not need.

Figure 8 shows the benefit devices received from participating in PINCH as we varied the capability ratio from 100% down to 10%. Even when the device *can* sense all of the context types on its own, it still garners a significant benefit from PINCH. With higher values of CR, the benefit to the device is a reduced context sensing and therefore energy burden. Instead, when the device's context sensing capabilities decrease (lower values of CR), the device gains access to context values that it cannot sense itself.

Figure 9 shows the additional cost incurred by devices as their own demands for context types increase from 10% to 100%. When a device is capable of sensing all of

[7]https://youtu.be/TOVtlOpgbNg

[8]The coverage transiently rises when devices change their sensing task as a result of a device receiving different beacons (with different context types) from the same neighbor in a given epoch. This is an artifact of our measurement approach, but it indicates a potential avenue of exploration: if devices change their sensing task *within* an epoch, it is possible that BLEnd can be manipulated to achieve even higher levels of context coverage. This investigation is left for future work.



(a) $k = 10$, CR $= 20$, all w_c equal

(b) $k = 10$, CR $= 20$, w_c decreases linearly

(c) $k = 20$, CR $= 20$, all w_c equal

(d) $k = 20$, CR $= 20$, w_c decreases linearly

(e) $k = 20$, CR $= 20$, DR $= 20$

(f) $k = 20$, CR $= 20$, DR $= 60$

Figure 5: Benchmarking coverage of greedy and randomized selection.

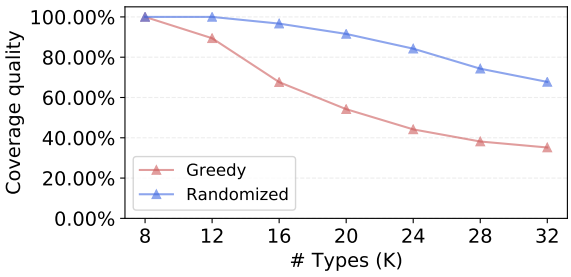


Figure 6: The pressure of increasing numbers of context types (CR $= 20$, DR $= 60$)

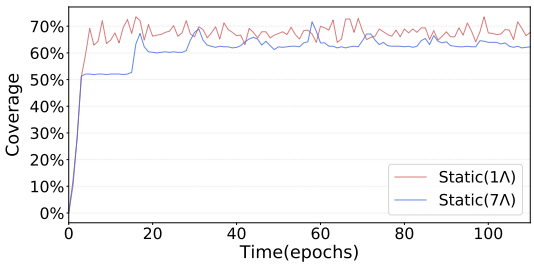


Figure 7: Adapting to changing context demands ($k = 16$, CR $= 20$, DR $= 20$)

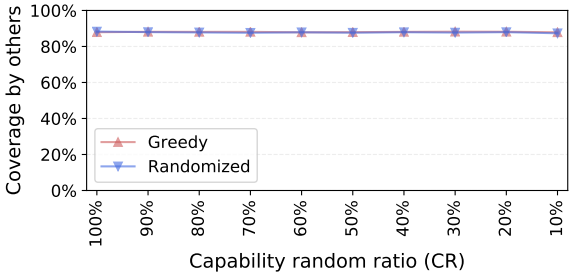


Figure 8: Benefits of PINCH ($k = 10$, DR $= 100$)

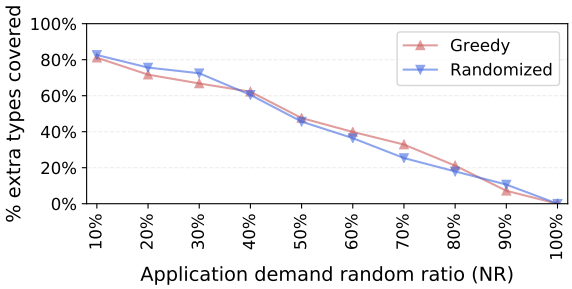


Figure 9: Costs of PINCH ($k = 10$, CR $= 100$)

the context types but only needs a random 10% of those types, the device is often contributing something to the neighborhood that is not useful to itself. However, as the device's needs grow, the context type that PINCH chooses for the device to sense and share is more likely to be useful to the device.

5) Considering the Rarity of Context Types: The rarity-weighted algorithm in Eq. 10 allows a device to favor selecting a context type that it alone in the neighborhood

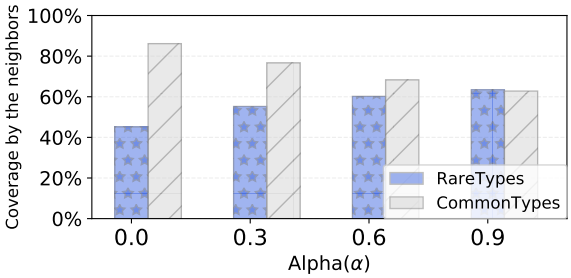


Figure 10: Rarity Weighting

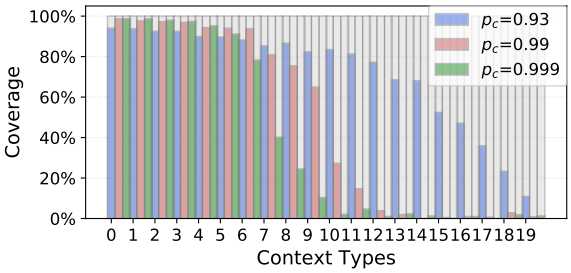


Figure 11: Collision Awareness (CR $= 60$, $n = 20$)

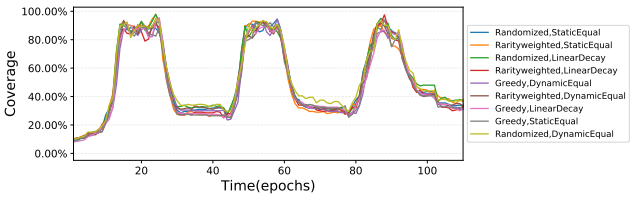


Figure 12: Trace of coverage percentage for congregating devices (CR $= 20$, DR $= 100$, $k = 10$).

can provide, even if it is not the most important according to the context demand model. We evaluate this with the group mobility scenario and assume every device needs every type. We vary the distribution of capabilities: for $k = 16$, we make every device capable of sensing the first eight context types plus, randomly, one of the remaining types. Figure 10 shows the coverage percentage of the rare (colored bars) and common (gray bars) types while varying α. We consider only the coverage contributed to by *other* devices (and not the device itself). Coverage of the rare context types increases with increasing α; this does come at some cost–devices that are sharing rare types cannot also share common types. Because all devices were outfitted with all eight of the common sensors, the rarity weighted algorithm allows the device to use the neighborhood's sensing resources to fill in the types that it *cannot* sense on its own. Some of the remaining common types are filled in by neighborhood sharing (e.g., because a second device in the neighborhood can provide the same rare type as another), and the device can use its own sensors to complete its coverage.

6) Collision Awareness: The prior experiments simply use BLEnd off-the-shelf without considering the implications of its probabilistic discovery guarantees. To evaluate the collision-aware algorithm of Section III-E we use the same core settings as those in Figure 5(d) but with increased capabilities (CR=60) and an increased number of nodes (20) to make collisions even more likely. Instead of assuming that the probability of coverage is the same as BLEnd's discovery probability, we set $p_c = \{0.93, 0.99, 0.999\}$[9]. Figure 11 shows that PINCH achieves the higher target p_c for the more important context types (i.e., those with lower indices); this comes at a cost of lower coverage for the less important types. In especially dense networks or when BLEnd is parameterized with a low target discovery probability, applications can use this collision-aware protocol as another option for achieving high coverage of important context types.

B. Realistic Smart-City Scenarios

While the previous evaluations attempted to benchmark the various parameters and settings for PINCH, the remainder of our evaluation explores more dynamic scenarios, both in terms of application mobility setting and device capabilities.

1) A Central Meeting Point: PINCH is most useful when devices are co-located for a sufficiently long period of time to share sensed context information. To assess how this benefit might fluctuate for realistic behaviors, we defined a model in which devices periodically transition between (1) randomly moving in a space and experiencing only transient encounters and (2) spending a longer time in a central meeting point (e.g., a market, store, restaurant, etc.). In particular, we selected a central point in the city as a meeting place; devices travel from random starting positions toward this meeting point, remain close together for a period of time, then move away before repeating the process. Figure 12 shows the average coverage percentage for all of the devices. The figure shows three "meeting times"; when the devices have congregated, their coverage is high; when the devices are isolated, their coverage drops to only what they can individually sense.

2) Leveraging Situated Beacons: PINCH can also enable mobile devices to collect context from fixed devices. To demonstrate this potential, we present a scenario in which a single mobile device connects opportunistically to stationary beacons in a city. In this scenario, the situated sensors can provide *all* context types. The mobile device has *no* sensing capabilities, yet, as Figure 13 shows, the device is able to achieve some coverage of context as it moves. Figure 13 shows the randomized algorithm's context coverage quality at the mobile device as we vary a, the number of context types the mobile device demands. The x-axis plots the average number of neighbors in a run (i.e., density of 1 indicates that the mobile device is connected to, on average, a single beacon at any given time). Note that, although the number of situated beacons the device can contact is determined by the density, because the device is connected to a beacon for multiple epochs as it passes by, the beacon can change its contents over time, improving the coverage for the mobile device.

[9]Though we used a target discovery probability $p_d = 0.9$, the optimal settings actually have a theoretical discovery rate of 0.92; in the simulator, we achieved a BLEnd beacon reception rate of 0.93.

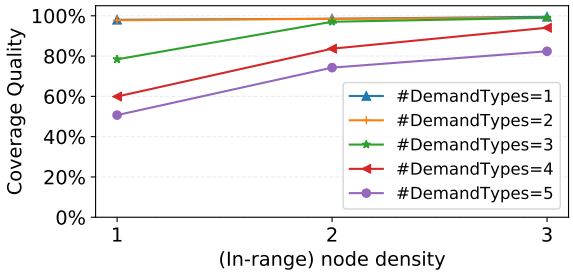


Figure 13: PINCH with situated beacons

VI. Conclusions

This paper introduces PINCH, a system that embodies a suite of heuristics for devices to opportunistically share sensed context within their local network neighborhoods. Such sharing has significant potential when context values are physically correlated. PINCH's heuristics leverage empty space in the periodic beacons of the BLEnd neighbor discovery protocol. As such, PINCH entails no additional energy costs beyond context sensing itself. We show that devices participating in PINCH can self-organize to provide very high degrees of coverage of context types. This sharing has multiple benefits in smart-* applications, e.g., allowing a device to collect context information it cannot sense on its own, or sharing the energy burden of context sensing across a neighborhood.

This initial work opens several avenues of exploration. We can change the beacon contents, for instance, to include more than just a single context type (if the data types themselves can be represented compactly). We could also explore exposing additional information about devices' needs, e.g., explicitly exposing *uncovered* types or sharing the needs of neighboring devices. The latter would allow PINCH to extend beyond a one-hop network neighborhood to provide larger coverage. Importantly, the algorithms we derived so far do not explicitly account for the cost of context sensing itself. Incorporating this information into the selection algorithm might change the decisions made. In summary, PINCH enables distributed cooperative context sensing without explicit collaboration between devices; such an approach has wide potential applicability as smart-* applications become commonplace.

References

[1] M. Saloni, C. Julien, A. Murphy, and G. Picco, "LASSO: A device-to-device group monitoring service for smart cities," in *Proc. of ISC2*, 2017.

[2] U. Gretzel, M. Sigala, Z. Xiang, and C. Koo, "Smart tourism: foundations and developments," *Electronic Markets*, vol. 25, no. 3, pp. 179–188, 2015.

[3] C. Martella, A. Miraglia, M. Cattani, and M. van Steen, "Leveraging proximity sensing to mine the behavior of museum visitors," in *Proc. of PerCom*, 2016.

[4] G. Picco, D. Molteni, A. Murphy, F. Ossi, F. Cagnacci, M. Corra, and S. Nicoloso, "Geo-referenced proximity detection of wildlife with WildScope: Design and characterization," in *Proc. of IPSN*, 2015.

[5] M. Bakht and R. Kravets, "Searchlight: Won't you be my neighbor?" in *Proc. of Mobicom*, 2012.

[6] P. Dutta and D. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *Proc. of SenSys*, 2008.

[7] C. Julien, C. Liu, A. Murphy, and G. Picco, "Blend: practical continuous neighbor discovery for bluetooth low energy," in *Proc. of IPSN*, 2017.

[8] A. Kandhalu, K. Lakshmanan, and R. Rajkumar, "U-Connect: A low-latency energy-efficient asynchronous neighbor discovery protocol," in *Proc. of IPSN*, 2010.

[9] P. Kindt, D. Yunge, G. Reinerth, and S. Chakraborty, "Griassdi: Mutually assisted slotless neighbor discovery," in *Proc. of IPSN*, 2017.

[10] M. McGlynn and S. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proc. of MobiHoc*, 2001.

[11] Y. Qiu, S. Li, X. Xu, and Z. Li, "Talk more listen less: Energy efficient neighbor discovery in wireless sensor networks," in *Proc. of Infocom*, 2016.

[12] K. Wang, X. Mao, and Y. Liu, "BlindDate: A neighbor discovery protocol," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 949–959, 2015.

[13] L. Wei, B. Zhou, X. Ma, D. Chen, J. Zhang, J. Peng, Q. Luo, L. Sun, D. Li, and L. Chen, "Lightning: A high-efficient neighbor discovery protocol for low duty cycle wsns," *IEEE Communications Letters*, vol. 20, no. 5, pp. 966–969, 2016.

[14] D. Zhang, T. He, Y. Liu, Y. Gu, F. Ye, R. G. H., and Lei, "ACC: generic on-demand accelerations for neighbor discovery in mobile applications," in *Proc. of SenSys*, 2012.

[15] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song, "Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments," in *Proc. of MobiSys*, 2008.

[16] S. Nath, "Ace: exploiting correlation for energy-efficient and continuous context sensing," in *Proc. of MobiSys*, 2012.

[17] S. Cho and C. Julien, "Chitchat: Navigating tradeoffs in device-to-device context sharing," in *Proc. of PerCom*, 2016.

[18] A. Amiri Sani, K. Boos, M. Yun, and L. Zhong, "Rio: a system solution for sharing i/o between mobile systems," in *Proc. of MobiSys*, 2014.

[19] X. Zheng, D. Perry, and C. Julien, "Braceforce: A middleware to enable sensing integration in mobile applications for novice programmers," in *Proc. of MobileSoft*, 2014.

[20] A. Kansal, S. Saponas, A. Brush, T. Mytkiwocz, and R. Ziola, "The latency, accuracy, and battery (LAB) abstraction: Programmer productivity and energy efficiency for mobile context sensing," in *Proc. of OOPSLA*, 2013.

[21] K. Rachuri, M. Musolesi, and C. Mascolo, "Energy-accuracy tradeoffs of sensor sampling in smart phone based sensing systems," in *Mobile Context Awareness*, 2012.

[22] Z. Zhuang, K.-H. Kim, and J. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proc. of MobiSys*, 2010.

[23] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Programming wireless sensor networks with the teenylime middleware," in *Proc. of Middleware*, 2007.

[24] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: A neighborhood abstraction for sensor networks," in *Proc. of MobiSys*, 2004.

[25] M. Jonsson, "Supporting Context Awareness with the Context Shadow Infrastructure," in *Wkshp. on Affordable Wireless Services and Infrastructure*, June 2003.

[26] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proc. of ICDCS*, 2005.

[27] Texas Instruments, "Bluetooth low energy beacons (application report)," http://www.ti.com/lit/an/swra475a/swra475a.pdf, October 2016.