The Dissertation Committee for Chenguang Liu
certifies that this is the approved version of the following dissertation:

Facilitate collaboration in Internet of things (IoT) proximity networks

Committee:

Christine Julien, Supervisor

Vijay Garg

Sarfraz Khurshid

Amy L. Murphy

Mohit Tiwari

Facilitate collaboration in Internet of things (IoT) proximity networks

by

CHENGUANG LIU

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2020

to Chloe

Acknowledgments

It has been a long and amazing journey. Throughout this PhD study I have met a lot of great people who have helped me grow as a person and as a researcher. In particular, I'd like to thank my advisor Dr. Christine Julien, for her generous guidance and support. She has taught me to always have the spirit to explore and supported me through all my decisions, both inside and outside our lab, and I am really grateful for that.

I would also like to thank Dr. Amy Murphy, Dr. Vijay Garg, Dr. Sarfraz Khurshid and Dr. Mohit Tiwari for serving as my committee members. I have benefited greatly from my interactions with them. Their helpful feedback have contributed a lot to my doctoral research and this dissertation.

As a member of the mobile and pervasive computing group and a student at UT, I have received enormous support from many friends and lab mates. Here I would like to acknowledge my debt to these incredible people.

My deepest gratitude goes to my parents Shaokai Liu, Ling Wang and uncle Jing Wang. It is their unconditional love and support make it possible for me to pursue this degree.

Finally, I want to thank my wife Qian and our daughter Chloe for supporting and encouraging me throughout this adventure. They are the source of my strength.

Facilitate collaboration in Internet of things (IoT) proximity networks

Chenguang Liu, Ph.D.
The University of Texas at Austin, 2020

Supervisor: Christine Julien

The popularity and reduced cost of low-power multi-sensor Internet of Things (IoT) devices provide many opportunities for human-centered ubiquitous computing. These personal and environmental embedded systems *continuously* collect the context information about the user activity or the environment, and provide digital assistance (e.g., turn up the A/C, unlock a door).

While many existing *always-connected* IoT solutions rely on the success of cloud computing, there is a growing interest in discovering the benefits of utilizing local resources and on-device processing. To make the IoT applications work without global connectivity, we need to overcome the limitation of the IoT devices in terms of sensor equipment and energy consumption. On the other hand, the manners of device interactions have changed with advances in wireless communication. Today nearly all mobile devices are capable of short-range wireless communication (e.g., via Bluetooth, Zigbee, etc). This provides

a great opportunity for allowing the co-located IoT devices to communicate and interact free of user intrusion.

In this thesis, I advocate that we utilize the pervasive and opportunistic connections around us for facilitating collaboration in the internet of things. With the help from IoT proximity networks, the costly sensing tasks can be distributed to improve sustainability. Heterogeneous devices can leverage each other's capability in the spontaneous context neighborhoods. This dissertation details the technical solutions to the following challenges:

- A key enabler of collaboration in IoT proximity networks is the ability to continuously identify nearby resources. We develop a low duty cycle protocol BLEnd to let the IoT hosts automatically discover neighboring devices in range of wireless communication.

- To effectively coordinate context sensing tasks among the IoT devices via device-to-device (D2D) communication, we propose a generic collaborative sensing framework SCENTS which enables application-transparent collaboration for context sharing. SCENTS incorporates an active request-response model to let devices in proximity sense context information as a fleet, while balancing sensing fulfillment and the fairness of energy consumption.

- In complement to the request-response approach of SCENTS we architect the PINCH framework, which *proactively* distributes the context information in local network based on various context demand models.

To further optimize the sensing task assignment in an ad hoc grouping of devices, we develop the Stacon system. It entails a distributed algorithm that quickly adjusts the neighborhood's sensing task assignments based on the heterogeneity and dynamics of resources in proximity.

- To exhibit the real world implications of the proposed solutions, we deploy a IoT sensor testbed and acquire a data collection from user-carried mobile devices. In the end, we evaluate the sensing collaboration with this rich-context, real-life dataset.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent decade has seen the rise of highly integrated IoT devices with multiple low-powered commodity sensors and the ability to communicate wirelessly with other nearby devices. These emerging computing systems can be found in nearly every application domain, from wearables (e.g., sport watches and medical wearables) to home automation (e.g., thermostats and smart cameras), to smart city infrastructure (e.g., providing environmental monitoring or civic services).

The IoT devices use their on-device electronics to *sense* the object attributes or user activities from the physical world and provide personal or industrial assistance digitally. From a sensing perspective, how an IoT device operates is typically simple: it periodically queries the embedded sensors and sends readings to an on-device application, to a paired device, or to the "cloud" via a nearby gateway. This design works unsurprisingly well for enterprise applications and smart home settings. However, the full potential of personal IoT applications are constrained by this computing model. The tight binding between the application and the hardware forms a star topology and blocks the embedded sensor kit from being utilized by *multiple* applications [73]. Besides,

cloud-based context processing and service delivery pose an strong dependency on the hub or global connectivity. According to Gartner[1], the number of such IoT devices surpassed the human population in 2017 and is expected to grow to 20.4 billion by 2020. This means there will be a massive deluge of sensor data generated by the IoT devices and we need an alternative to offloading everything to the cloud.

Because of all these problems, there is a growing interest in discovering the benefits of utilizing local resources and on-device processing [22, 117]. However, the global connectivity and cloud-based coordination have to be substituted properly [26, 113]. Otherwise the usefulness of the IoT application will be damaged by the extreme resource limitations in terms of *hardware equipment* and *energy consumption*. Unlike Android or iOS applications, which typically have a sensor equipment known *a priori* to the developers, IoT applications often meet the problem of *missing needed sensing capability*, simply because a user may have a device that cannot perform every sensing task. For example, if a smart office application needs both accelerometer and microphone for detecting the user's working situation, then it would not be able to run on a device that lacks either of the physical sensors. Applications designed for user carried devices (e.g. smart phones, wearables) face the constraint of energy consumption. For instance, querying a costly sensor frequently on a battery powered system-on-chip (SoC) will inevitably drain the battery. One promising alternative to the cloud coordination is the local area interactions [1, 26];

[1] `https://www.gartner.com/newsroom/id/3598917`

that is, coordinating the co-located IoT devices with pro-active interactions and providing the IoT service collectively.

With the advances in wireless communication, the manners of IoT device interactions have changed from tag based technologies (e.g., scanning QR codes [2], near-field communication cards [3]) to WPAN (e.g., ANT, Bluetooth, Zigbee). Physically nearby devices now have the chance to interact non-intrusively and securely [10, 88, 121]. They can also form proximity networks to further optimize the context collection and service delivery processes. Instead of offloading all sensed data to a third-party and relying on cloud services and remote service invocations (e.g. turning up the A/C through a Nest thermostat [4]), the whole computation flow can then be achieved by *local collaboration* since the sensing and consumption both occur in the same physical area. The key of this collaboration is sharing the context data among co-located IoT devices efficiently. Without the assumption of any-to-any reliable communication between the nodes or a fixed network topology, it is challenging for IoT devices to coordinate *asynchronously* at low duty cycles with the underlying *dynamic* and *unreliable* connections.

For all these reasons, the effectiveness of local area interactions is crucial to the collaborative IoT applications. Distributing the sensing cost can help the mobile and wearable systems to be sustainable. Heterogeneous devices can

[2] `https://en.wikipedia.org/wiki/QR_code`
[3] `https://en.wikipedia.org/wiki/Near-field_communication`
[4] `https://nest.com/thermostats/nest-learning-thermostat/overview/`

trade-off their sensors and connectivities in exchange for mitigating their own incapabilities and limitations.

In this dissertation, I present a series of core technical challenges in the opportunistic collaboration among IoT devices in proximity networks, along with their proposed solutions. The collective goal of these works is to identify the key building blocks caused by *hardware constraints* and *network dynamics*, to mitigate or solve the technical difficulties from a software perspective, and to encapsulate the solutions as generic, system-level software components which can be easily ported to the existing IoT embedded systems without encumbering hardware resources. To summarize, the research contributions include the following:

- To enable a mobile IoT device to communicate, compute and collaborate with other co-located peers, our first work along this path aimed to give them the ability to continuously discover what digital resources are located in the surrounding environment [58]. The primary challenge of identifying "who is around" is a neighbor discovery problem, i.e., to find a low duty cycle schedule of transmitting and receiving operations which can let the IoT node to automatically discover its neighboring devices (e.g. sensing peers, stationary sensors) in range of wireless communication. In addition to solving the problem on an abstraction level, we investigate a popular WPAN technology Bluetooth Low Energy(BLE) and developed a continuous neighbor discovery protocol BLEnd tailored

to the real constraints. We evaluate the its performance against state-of-the-art protocols on the TI CC2650 [5] platform. The ultimate goal of BLEnd is to directly empower developers with the ability of integrating resource discovery into their applications without mastering the domain knowledge of wireless communication. We open source the implementation of BLEnd scheduler for the Nordic nRF52 system-on-chip family [6].

- To effectively coordinate sensing and sharing tasks among mobile IoT device via D2D communication, we formulate a dynamic sensor selection problem and develop a generic collborative sensing framework SCENTS [76]. This framework is designed to use commonly available communication technologies to share sensing capabilities directly among co-located heterogeneous IoT devices. Its sensing partner selection algorithm accounts for sensing and communication costs, and predicting and adapting to mobility-induced failures.

- As a complement to the request-response model used in SCENTS, we explore the implicit approaches where the context information is *proactively* distributed in the network neighborhood. The PINCH framework exploits self-organizing features in the context sharing schemes to opportunistically disseminate the context of choice [80]. In PINCH, a set of self-organizing heuristics is derived to use limited local views of the state of a one-hop neighborhood to determine the most *useful* type of context

[5] `http://www.ti.com/product/cc2650`
[6] `https://www.nordicsemi.com/`

information for a device to sense and share. We build PINCH on the BLEnd protocol, directly considering how aspects of BLEnd influence our heuristic for context sharing.

- While developing the context sharing strategies, we model the sensor and communication characteristics of a state-of-the-art IoT sensor kit and evaluate the collaboration in several smart city settings. A systematic evaluation software is built to simulate the mobile IoT nodes in real 3D city-level environments with 2.4GHz wireless communication technologies [7]. It supports customizable mobility patterns and trajectories. The buildings, which serve as signal obstacles, are extracted and transformed from the real map data source [8].

- To exhibit the feasibility of context sharing in everyday environments, we explore the self-stabilizing nature of the context neighborhood with a low-power system-on-chip. With the BLEnd middleware we are able to showcase the resilience of distributed sensing collaboration with optimized task assignments using the Stacon system [75]. Further, we deploy a testbed with anchor nodes at 24 locations inside a university academic building and 7 human-carried beacons to collect a dataset with real spontaneous connections [78]. We use the acquired dataset to verify the usefulness and utilization of the sensing collaboration in IoT proximity networks.

[7] INeT framework: `https://inet.omnetpp.org/`
[8] OpenStreetMap: `https://www.openstreetmap.org`

Chapter 2

BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy

The ability to continuously discover neighboring devices in range of wireless communication is a key building block of many Internet of Things (IoT) scenarios. Smart spaces require the ability to detect the proximity of users to devices deployed in the space, to trigger interactions. Smart retail systems need to detect the time a person spends in each area of the store; information acquired via device discovery serves both to orchestrate interaction with the user and to analyze long-term shopping behavior to improve the retail experience. Smart cities take these capabilities to a larger geographical scale, enabling the automatic triggering of personalized services on a smartphone based on physical proximity to designated places (e.g., monuments or exhibits [84]) hosting fixed nodes.

Neighbor discovery can also be used as a stepping stone for services concerned with proximity among people, e.g., to ensure that tourists do not get separated from their tour group, to enable proximity-based authentication [88], or to ensure that children on the way to school are always close enough to at least one responsible adult [45, 111]. Studies on behavioral analytics are

fueled by the ability to non-invasively detect proximity among humans [29] or animals [98]. In general, the *continuous* and unsupervised (i.e., without explicit user interaction, such as pairing) ability to detect proximity to other nearby, potentially mobile, devices enables new interaction patterns, unlocking novel application domains.

Continuous Neighbor Discovery: State of the Art. Most state-of-the-art continuous neighbor discovery protocols divide time into equal-sized slots, during which a node is either active or inactive. When the active slots of two nodes overlap, discovery occurs. The protocols are evaluated by assessing the *discovery latency*, defined as the number of slots until a neighbor is detected; a protocol's *duty cycle*, defined as the number of active slots over a unit of time, serves as an indirect measure of energy consumption. Protocols are described relative to the mechanisms they use to determine which slots are active, with the result being either probabilistic or deterministic discovery. In the Birthday protocol [86], nodes randomly make a slot active with a given probability, offering good average case performance but not providing guarantees on discovery latency. Instead, Disco [37] and U-Connect [61] space active slots according to prime numbers, relying on the properties of the Chinese Remainder Theorem to guarantee discovery within a tight time bound. Searchlight [13] and BlindDate [124] offer hybrid approaches, placing some active slots for deterministic discovery, then adding more in a pseudo-random manner to improve performance.

Nihao [101] departs from these protocols by specifying that a slot can

either be for listening or transmitting, where the latter behavior is defined by a single, short beacon at the beginning of the slot. By observing that a single short beacon costs much less than listening for the entire slot, Nihao proposes to "talk more and listen less", resulting in a protocol with more active slots, but with competitive consumption as most slots are cheaper, transmission-only slots.

Theory vs. Practice. Research on continuous neighbor discovery has been hitherto characterized by a strong slant towards theory, with most protocols taking the slotted approach described earlier and assuming slots of arbitrary length. Much work has focused on relating protocols to one another at the model level, based entirely on a fixed size slot. As a consequence, details concerning the actual behavior within a slot are often abstracted away.

Unfortunately, these assumptions overlook system-level constraints that significantly change tradeoffs and may even prevent the use of a given protocol with a given network technology. For instance, an average discovery latency of 10,000 slots or more is common [101]. This is acceptable when slots are small; a common slot length is 10 ms, yielding a latency of minutes. However, the Bluetooth Low Energy (BLE) standard, available on many commodity devices, prescribes that advertisements are separated by at least 20ms (and even 100ms for some advertisement types). This places a hard lower bound on slot duration and can increase discovery latencies by up to an order of magnitude, rendering them unacceptable *in practice*. Further, existing protocols' models ignore the density of nearby nodes, a factor that can increase the potential for

beacon collisions that hinder discovery, as discussed next.

Our Perspective. We take a view motivated by a desire for a *practical* approach to continuous neighbor discovery.

First, we do not neglect system-level concerns; instead they are the starting point of our endeavor. We choose BLE as our reference platform as it is pervasive on many consumer electronics. On top of BLE, we devise a continuous neighbor discovery protocol, called BLEnd (BLE neighbor discovery), that is compatible with BLE's technological constraints and features, as outlined in Section 2.1.

Moreover, we emphasize metrics that impact the use of BLEnd in real environments. Specifically, we recognize that packet collisions reduce discovery rates, making it difficult if not impossible to reach 100% discovery. State-of-the-art protocols ignore this aspect, simultaneously aiming to reach this unattainable goal and failing to provide application designers with information about the concrete, negative effects of collisions. Instead, BLEnd enables designers to express requirements as a *service level agreement* that includes the target discovery latency plus two parameters related to collisions: expected node density and target discovery probability. An optimizer tool automatically derives the BLEnd parameters that meet these requirements with the lowest energy costs, allowing application designers to both tune BLEnd to their specific needs and to use it with a precise understanding of its actual performance.

Our Protocol: BLEnd. In a system without energy constraints, discov-

ery can be achieved with an always-on receiver and periodic broadcasts to announce presence. As long as messages do not collide, discovery is guaranteed. Alternatively, if nodes are synchronized, they can exchange discovery messages at predetermined times, allowing the radio to be otherwise turned off. Unfortunately, most real systems have tight power budgets and providing perfect synchronization is expensive. Therefore our goal is to make guarantees about discovery latency while minimizing power consumption, allowing neighbor discovery to be continuous. As with other protocols, the key is scheduling transmitting and listening, whose spatio-temporal overlaps enable discovery. However, our design departs from the state of the art in many respects.

Existing approaches focus exclusively on *bi-directional* discovery (i.e., node A discovers node B and vice versa). While this is intuitive, we observe that, for many applications, *uni-directional* discovery, in which only one node in a pair discovers the other, is sufficient. For example, when discovery serves as the core mechanism for recording proximity in the human or animal social studies above, uni-directional detection is sufficient to demonstrate that A and B were in range at some point in time. Offline analysis can later infer that a bi-directional contact has taken place from a single uni-directional discovery. Unidirectional detection can also serve as a cornerstone for *triggering communication*, with one device detecting the other then initiating a separate bi-directional communication.

Based on these motivations, and in contrast with the state of the art, we design the core of BLEnd to support uni-directional discovery. This

application-level choice has deep system-level implications, as it unlocks opportunities for energy optimization currently missed by the state of the art. As we discuss in Section 2.2, the focus on uni-directional discovery allows us to define a periodic schedule where each node can be duty-cycled during slightly more than half of the period and be *completely inactive* for the remaining portion, significantly reducing energy consumption. This is achieved without any assumptions about time synchronization and yet provides deterministic discovery latency guarantees at low duty cycles. This design decision does not affect BLEnd's generality or applicability; while uni-directional discovery is the fundamental building-block, it can be efficiently extended to bi-directional discovery.

We depart from dominant trends in the state of the art in two other respects. First, we *completely remove the concept of slot.* The primary advantage is in added flexibility for BLEnd to meet application requirements with the lowest possible energy consumption. Second, we *directly account for constraints from the BLE stack and the nature of communication in dense mobile environments.* We create an *optimizer* (Section 2.3) that determines the parameter settings that provide optimal continuous neighbor discovery w.r.t. a node's battery lifetime, given an application-desired service level agreement. This optimizer is built around a novel analytical model (Section 2.4) that accounts for *i)* idiosyncrasies of BLE, including accurate power consumption of BLE operations derived from laboratory experiments, and *ii)* packet collisions, which profoundly affect the behavior of neighbor discovery. Alongside

this mathematical model, we provide an accurate simulator that we exploit to show the protocol functionality in a variety of scenarios (Section 2.5) as well as its performance in comparison to two reference protocols (Section 2.6). Finally, we describe our implementation of BLEnd on a *standard, unmodified BLE stack* (Section 2.7), and analyze its performance (Section 2.8). Our experimental results confirm not only that BLEnd is more versatile and efficient than its competitors, but also that our model, simulator, and implementation are in good agreement, enabling the immediate use of BLEnd in applications. Section 2.9 ends the paper with brief concluding remarks.

2.1 Background and Motivation

One of our motivations for BLEnd is that existing *continuous* neighbor discovery approaches are incompatible with BLE in various ways. This section discusses elements of BLE that are critical when using it for neighbor discovery. We then describe how neighbor discovery is commonly done in BLE, notably *not* in a continuous manner. We discuss the possibilities of using BLE as the underlying technology for existing approaches, laying the foundation for our novel continuous neighbor discovery protocol, BLEnd.

BLE in the Abstract. BLE is an obvious candidate to support continuous neighbor discovery for commodity applications, due to its low power and wide availability [81, 114]. We take as a premise the need to work with and around the BLE standard [19]. Figure 2.1 overviews key elements of BLE discovery, in which one side acts as an *observer* and the other as a *broadcaster*. The

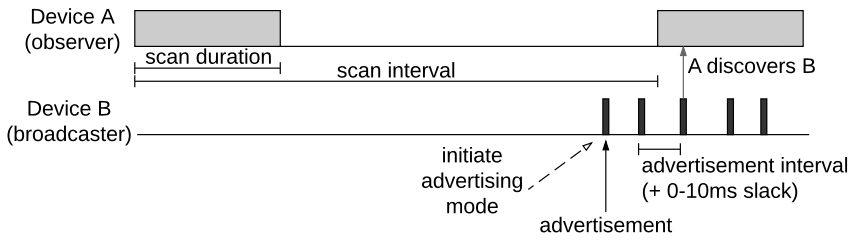


Figure 2.1: BLE discovery.

broadcaster emits an *advertisement event*, typically 1-3ms long depending on hardware, during which it sends a beacon on one of the three advertisement channels then waits briefly for a *scan response* on the same channel. Each advertisement event may perform this process on one, two, or all three of BLE's advertisement channels. Advertisement events are triggered periodically based on the *advertisement interval*. BLE also automatically adds a 0-10ms *random slack* to the advertisement interval. This slack is engineered into BLE to reduce the potential for simultaneous advertisers to collide many times in a row.

On the receiving side, a BLE observer listens continuously for a *scan duration*, repeating this *scan event* periodically based on a *scan interval*. The scan duration must be shorter than the scan interval; the advertisement and scan interval must be at least 20ms long; all three values are parameters from the application layer. Each scan event listens on exactly one advertisement channel; subsequent scan events are required to cycle through the three advertising channels. A scan event detects an advertising event only if the advertisement is sent on the matching channel; therefore, in mapping continuous

neighbor discovery onto BLE, we send every beacon on all three advertisement channels to guarantee that it is captured by any listening device, regardless of the listener's scan channel.

BLE in the Concrete. Our approach to continuous neighbor discovery is generic to devices supporting the BLE specification [19], including many modern Android devices. Our evaluation uses TI SensorTag (`www.ti.com/sensortag`), an inexpensive sensing device with a complete BLE radio and networking stack representative of many realizations of the BLE specification, especially on IoT-style devices. Here, we highlight elements of the SensorTag BLE implementation relevant to continuous neighbor discovery.

Basic "off-the-shelf" BLE discovery expects one or more nodes to act as broadcasters and another to act as observer. However, the specification allows a single device to assume both roles, if supported by the hardware. The SensorTag does support applications that take on the multiple roles, switching between them without restarting the BLE stack. This capability is common in many other BLE implementations, both on lightweight devices and on Android.

Figure 2.2 shows an oscilloscope trace of a segment of BLE operation on a CC2650STK SensorTag. Oscilloscope measurements in this paper were captured with a Picoscope 2204A acquisition device. Current traces result from measuring the voltage drop over a 10Ω low side shunt resistor placed in series with the SensorTag. In this trace, our device begins as an observer, scanning one advertisement channel. At time 994.84ms, the application instructs the

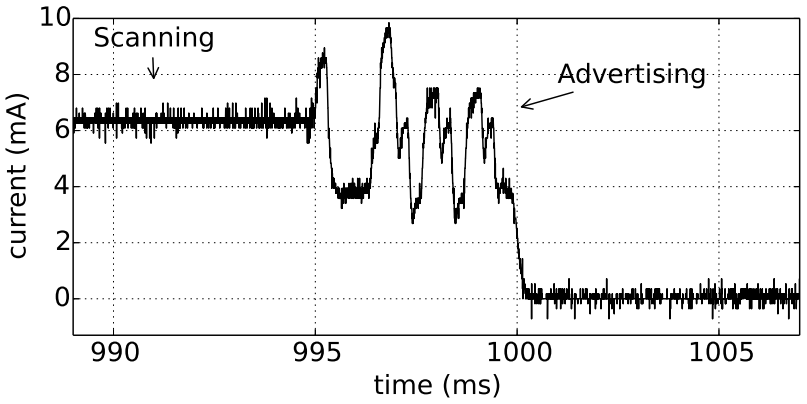


Figure 2.2: Oscilloscope trace showing scanning then advertising on the SensorTag.

device to cease observing and start advertising. Here and throughout the paper we use non-connectable advertisements, as the intent is simply to announce presence, not to initiate a connection for sending data. The advertisement event starts at 996.26ms and ends at 999.46ms, sending a single advertisement on each of BLE's advertising channels. This advertisement event lasts for $b = 3.2$ms. Using this measurement setup, we recover the instantaneous currents of scanning and advertising, respectively, as $I_{scan} = 6.329$mA and $I_{adv} = 5.725$mA. We also measured a stand-by current $I_{idle} = 80.64\mu$A, although the TI SensorTag datasheet indicates an expected value of 1μA.

2.2 BLEnd

We start from the premise that many applications require only one node of a pair to detect the other's presence and design support for unidirectional discovery (U-BLEnd). We then show how this is easily adapted

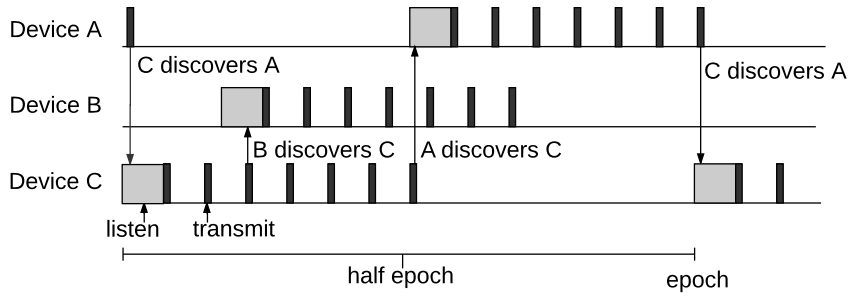


Figure 2.3: U-BLEnd: Uni-directional discovery.

to bi-directional discovery. In both cases, the goal is to achieve continuous neighbor discovery without the device's radio having to be continuously active.

Uni-directional Discovery: U-BLEnd. Our initial goal is to guarantee that one of every pair of devices will discover the other within a given time, the *discovery latency*, while simultaneously minimizing the energy consumed in discovery activities. The protocol behavior is a simple, repeating sequence of listening intervals (scans, in BLE) and beacon transmissions (advertisement events). We term the duration of this repeating sequence the *epoch*, E.

On a node, U-BLEnd exploits the first half of every epoch to attempt to discover or be discovered, then remains inactive for the second half, with the radio in stand-by. The radio is also switched to stand-by whenever the node is not listening or transmitting during the first half epoch, conserving as much energy as possible.

As shown in Figure 2.3, an epoch always begins with a listening interval whose length depends on application requirements, most critically the

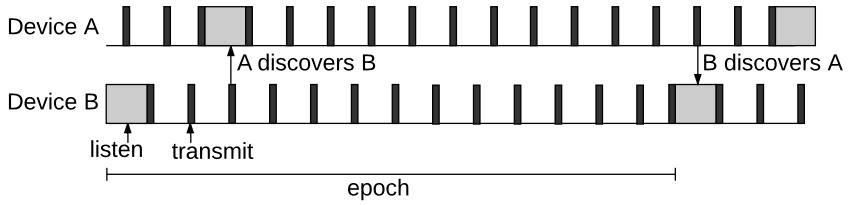


Figure 2.4: F-BLEnd: Full-epoch bi-directional discovery.

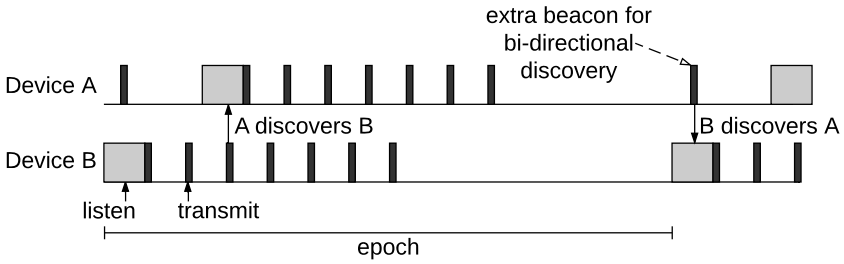


Figure 2.5: B-BLEnd: Efficient bi-directional discovery.

discovery latency. Immediately following the listen interval, BLEnd begins a sequence of beacon transmissions that lasts for the remainder of the first half of the epoch, with the last scheduled beacon falling just inside the second half of the epoch. The interval between the beginning of adjacent beacons is at most the length of the listen interval, thus ensuring that if the listening interval of another node overlaps with the active portion of this node, the listener receives at least one beacon and discovery occurs, as shown with arrows. Further, the active portion of the epoch, defined as the time from the beginning of the listening interval to the end of the last beacon, is greater than half of the epoch. This configuration *guarantees* that the active portions of two independent nodes overlap, resulting in detection. Interestingly, it *may* also result in bi-directional discovery, as shown in Figure 2.3 for A and C.

A key element of BLEnd relative to existing continuous neighbor discovery protocols is its direct consideration of the practical constraints of BLE. In the context of uni-directional discovery, there are two important elements. First, to provide guarantees on discovery latency, a node's beacon must be entirely contained within a listener's listening interval, and BLE beacons have non-negligible duration. Setting $A = L$, i.e., the beaconing interval equal to the listening interval does not consider the case in which the listener might receive a partial beacon at either the beginning or the end of its listening interval, a situation that may prevent discovery.

Second, BLE adds random slack to the application-specified advertising interval. Therefore, when U-BLEnd specifies an advertising interval of A, beacons may be spread as much as $A + s$ apart, where s is the maximum random slack added. Because this may make A greater than L, a listening interval may fall entirely between two beacons, preventing discovery.

We cater to these observations by defining $A = L - b - s$.

Full-epoch Bi-directional Discovery: F-BLEnd. To naïvely implement bi-directional discovery, one can simply continue beaconing throughout the entire epoch; we call this variant full-epoch BLEnd, F-BLEnd. In Figure 2.3, B would not discover A in U-BLEnd as B's listening interval falls inside the inactive portion of A's epoch. Instead, as shown in Figure 2.4, discovery happens in F-BLEnd, as B's listening interval overlaps one of A's additional beacons.

Efficient Bi-directional Discovery: B-BLEnd. Adding *all* of the beacons in the inactive half of the epoch is unnecessary; a more efficient strategy is employed by our last protocol variant, B-BLEnd. Consider discovery between two nodes A and B. If the U-BLEnd schedule allows A to detect B in one epoch, the goal of B-BLEnd is for B to detect A in the next epoch. To this end, any beacon added by B is unnecessary, as B has already been detected by A. Instead, it is sufficient that A adds exactly *one* beacon—the one falling in B's listening interval, as shown in Figure 2.5. This is possible if beacons include a small bit of information, i.e., the time between the start of the epoch and the beacon transmission. Combining this information with the knowledge about the duration of the epoch and of the advertisement interval enables a receiving node (A in our case) to compute the start time of the discovered node's next listening interval and selectively schedule only the one beacon, out of all those that F-BLEnd would add, that lands inside the other node's listening interval, thus allowing bi-directional discovery.

In scenarios with more than two nodes, the worst case requires a node to activate all of the beacons in the inactive half of the epoch, as in F-BLEnd. However, we observe that a single added beacon may allow discovery by more than one neighboring node if the listening intervals of multiple neighbors overlap with the beacon. Put another way, a new beacon is not necessarily needed for every neighbor. On the other hand, when a node is among few neighbors, correspondingly few beacons will be activated by B-BLEnd—none if the node is alone. It is this observation that makes BLEnd particularly suitable for con-

tinuous operation: *to reduce the energy cost of continuous neighbor discovery, BLEnd effectively adapts to the natural dynamics in the density of neighboring nodes.*

2.3 Optimizing Latency vs. Lifetime

Using BLEnd in real applications requires that trade-offs between latency and energy consumption are made explicit. Application developers often expect zero latency and infinite lifetime, which is clearly impossible. The tool we describe next, the *BLEnd optimizer*, enables developers to quickly explore these tradeoffs and select a protocol configuration most suited to the application requirements.

The latter may vary widely. An application that monitors proximity of visitors to museum exhibits [84] targets a discovery latency of seconds, which is more expensive in terms of energy but acceptable since devices can be easily recharged after each museum visit. On the other hand, wildlife monitoring [98] must accept a discovery latency of up to a minute, as devices are animal-borne and expected to last months, if not years. In our experience, neither latency nor lifetime requirements are cast in stone; they are selected as a compromise between application- and system-level concerns.

In this respect, our work has two assets relative to the state-of-the-art. First, the configuration space of existing protocols is severely limited by the coarse-grained discretization induced by slots and other constraints (e.g., choosing prime numbers [37, 61]). In contrast, BLEnd's slotless opera-

Table 2.1: Parameters used in model and optimizer.

Parameter	Description
Input parameters: Application requirements	
mode	uni-directional vs. bi-directional discovery
Λ	maximum discovery latency
P	minimum discovery probability
N	maximum number of nodes in a collision domain
Input parameters: BLE stack	
b	duration of an advertisement event (beacon)
s	maximum random slack for an advertisement event
Output parameters: BLEnd configuration	
E	duration of the epoch
A	duration of the advertising interval
Derived output parameters	
L	duration of the scan interval (listening)
n_b	number of advertisements per epoch

tion provides remarkably more configuration options. Second, the impact of collisions, which directly result in missed discoveries, is routinely neglected by models underlying existing protocols; their latency and lifetime estimates are therefore deceptive, as they do not match what is possible in actual implementations. In contrast, the BLEnd model in Section 2.4 explicitly accounts for collisions, enabling the optimizer to more realistically determine the best BLEnd configuration.

Application requirements constitute a service-level agreement of sorts that must be honored by the BLEnd configuration generated by the optimizer. The requirements input to the optimizer are shown in Table 2.1, along with the expected outputs. A developer must specify the maximum discovery latency Λ and minimum discovery probability P allowed in the application, along with

the maximum number N of nodes in a collision domain. P is defined for B-BLEnd and F-BLEnd as the probability that each node discovers all of its neighbors within Λ; for U-BLEnd, instead, P is the probability that at least one node in a pair discovers the other.

The inputs also include system-level parameters. The duration b of a BLE advertisement event may change depending on the hardware. Hereafter we consider $b = 3.2$ms as measured on our platform when using all three channels, which yields lower discovery latency and better resilience to collisions. The random slack introduced by the BLE stack is set to $s = 10$ms as per the specification [19].

The optimizer returns a configuration $\langle E, A \rangle$, i.e., the advertisement interval and epoch length, that satisfies the application requirements and minimizes energy consumption. For instance, assume the developer requires bi-directional discovery with $\Lambda = 2000$ms, $P = 0.95$, $N = 15$. The output configuration $E = 667$ms, $A = 71$ms guarantees that *i)* in the worst case where 15 nodes are all within range of one another, each of them has a 95% probability of discovering the others within 2s, and *ii)* this is achieved with the minimal energy consumption. The optimizer also outputs derived parameters: the scan interval $L = A + b + s$ and the number of advertisements, which is $n_b = \lfloor \frac{E}{2A} \rfloor - 1$ in the uni-directional case and $n_b = \lfloor \frac{E}{A} \rfloor - 1$ in the bi-directional (worst) case.

The optimizer has two fundamental components. The first is a model

of the drain of electrical charge during one epoch:

$$Q(E, A) = I_{scan}L + I_{adv}n_b b + I_{idle}(E - L - n_b b) \tag{2.1}$$

where I_{adv} is the (average) instantaneous current consumed by the radio when advertising, I_{scan} when scanning, and I_{idle} when not engaged in neighbor discovery. These values must be measured for a given hardware, as we did for the SensorTag in Section 2.1. I_{idle} is determined by application specifics, e.g., whether the device senses, computes, or engages in other communication. In Section 2.5, we report lifetime values derived with $I_{idle} = 0\mu$A, as we are focused on the application-independent neighbor discovery functionality, and $I_{idle} = 80.64\mu$A, as measured on the SensorTag.

The second component of the optimizer is a model that, for a given BLEnd configuration $\langle E, A\rangle$, estimates the discovery probability P_d as a function of all the optimizer inputs, *by considering collisions*. This model is one of the contributions of this paper, and is presented in detail in Section 2.4.

Based on these components, the optimizer performs an exhaustive search across all possible $\langle E, A\rangle$ configurations. We test each possible epoch value smaller than the target latency, $E \leq \Lambda$; for each value E, we test all advertisement intervals allowed by the BLE specification [19], i.e., $A \geq 20$ms. Further constraints trivially rule out degenerate combinations, e.g., $E \leq A$ or when an epoch cannot accommodate a complete BLEnd schedule. For each pair $\langle E, A\rangle$ the optimizer computes the expected discovery probability P_d according to the model in Section 2.4. If $P_d \leq P$, i.e., the expected discovery probabil-

ity does not satisfy the one targeted by application requirements, the $\langle E, A\rangle$ configuration is discarded as invalid. Otherwise, the current drain $Q(E, A)$ is computed; the optimizer outputs a valid $\langle E, A\rangle$ that minimizes this value.

The search space determined by the values of E and A is explored in (configurable) increments of 1ms, as this is close to the resolution of timers commonly found in BLE devices. Along with the other constraints mentioned above, this limits the computational overhead; our R implementation of the optimizer computes the optimal configurations for the scenarios considered in this paper in at most a few minutes on a common laptop.

2.4 Modeling Discovery Probability

We next derive a model of the discovery probability P_d using the parameters in Table 2.1. Discovery probability is intimately intertwined with the probability of colliding beacons. Other factors, most notably interference in the crowded 2.4GHz band that BLE uses, can also cause a beacon to be lost. However, the three channels (37–39) used for advertising are defined by the BLE specification [19] *i)* to be different from those used for actual communication, and *ii)* to avoid interference with the most commonly used WiFi channels. Further, to the best of our knowledge, no existing work analyzes the impact of collisions of the beacons within continuous neighbor discovery; these collisions are the primary source of beacon loss, and we analyze these impacts in Section 2.4.1. The discovery probability is also affected by specifics of the BLE communication stack, most notably the random slack, as we discuss in

Section 2.4.2.

2.4.1 Analyzing the BLEnd Schedule

We first derive discovery probability by considering the chance that beacons collide due uniquely to the BLEnd schedule.

In the following, the same model handles uni-directional and bi-directional discovery; the only difference is the number C of nodes in the collision domain. In the uni-directional case (U-BLEnd), $C = \frac{N}{2}$ on average because a node's schedule does not have beacons in the second half of the epoch. In the bi-directional case we assume $C = N$. F-BLEnd always schedules beacons throughout the second epoch, while B-BLEnd schedules beacons on-demand only if and when needed. However, B-BLEnd intends that all $N - 1$ nodes that are not the listener attempt to "hit" the listen interval with a beacon. Therefore, our model focuses on F-BLEnd because *i)* it provides the worst case for both collisions and energy consumption, and *ii)* it yields a more tractable model.

Within an epoch, collisions among beacons are harmful only when they occur within another node's listen interval; beacon collisions not overlapping with the listen interval of some node are irrelevant. Therefore, we focus on the listen interval and observe that, due to the structure of a BLEnd schedule, there are at most $C - 1$ other nodes that have a beacon scheduled within another given node's listen interval.

Consider one of these $C-1$ senders, with a beacon start time t falling in

the listen interval. Because successful discovery requires the listen interval to overlap entirely with the beacon, this sender is not discovered by the listener if another beacon (from another sender) overlaps even partially with the first one. This collision happens if the other sender chooses a beacon start time in the interval $(t - b, t + b)$. Therefore, $2b$ of the possible options for beacon start times cause collisions. Because beacon start times must be chosen in the interval $[0, L - b]$ to ensure that the entire beacon is received before the listening interval ends, the probability that some other sender collides with our selected sender is $\frac{2b}{L-b}$.

The discovery probability can therefore be computed as

$$P_{nc} = \left(1 - \frac{2b}{L - b}\right)^{\gamma} \tag{2.2}$$

where $\gamma = C - 2$ is the number of beacons potentially colliding with the chosen sender within the given listen interval. Since each sender is expected to send exactly one beacon during the listen interval, γ is simply the number C of nodes in the collision domain, minus the listener and the sender for which we are computing the discovery probability.

2.4.2 Accounting for BLE Specifics

The model above assumes that two consecutive beacons are spaced exactly by the advertisement interval A. However, if this were the case in BLE, two colliding advertisements would collide forever. To avoid this, BLE adds a "random slack" $r \in [0, s]$ to the start time of all advertisements in

a sequence, except the first one [19]. This slack has a subtle yet significant impact on discovery probability.

Discovery Is Possible Across Epochs. Arguably, the most important effect of the slack is that it unlocks the possibility that a beacon experiencing a collision in the first epoch is discovered in a later epoch, as the random slack moves the colliding beacons slightly relative to each other. Therefore, we model the discovery probability across epochs and its interplay with the maximum latency Λ.

In principle, based on (2.2), the probability of discovering a given node across k epochs is simply:

$$P_{d,k} = 1 - (1 - P_{nc})^k \tag{2.3}$$

where $(1 - P_{nc})^k$ is the probability that a node is *not* discovered across k epochs.

On the other hand, the only constraint relating the maximum latency and the epoch is that $E \leq \Lambda$, to guarantee that a complete BLEnd schedule can unfold. As a consequence, Λ is not necessarily a multiple of E. This implies that a node not discovered in the first k epochs may be discovered in the "spillover" $\Lambda - kE$; Figure 2.6 shows the relationship between Λ, E, and this spillover.

We observe that, within a given epoch, the rate at which nodes discover each other is constant, as the phases between nodes' epochs are independent. Therefore, the discoveries that occur in the spillover can be quantified simply

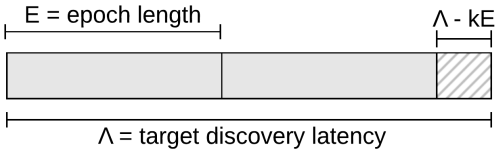


Figure 2.6: Relationship between Λ, E, and the "spillover".

by multiplying the discovery probability P_{nc} by the fraction of epoch $\frac{\Lambda - kE}{E}$ corresponding to the spillover. Moreover, the spillover increases the likelihood of discovery only if the node was *not* discovered in the first k epochs. Therefore, the probability that a node is discovered *for the first time* in the spillover is

$$P_{d,sp} = (1 - P_{nc})^k \left(\frac{\Lambda - kE}{E} \right) P_{nc} \tag{2.4}$$

and the overall probability of discovery within Λ is

$$P_d = P_{d,k} + P_{d,sp} \tag{2.5}$$

We next extend this simple formulation to account for other subtleties induced by the random slack.

Extra Beacons. In the presence of random slack, a listen interval of $L = A+b$ could miss discoveries even in the absence of collisions. Figure 2.7 shows an example, where a sender's beacon scheduled at the very end of a listen interval is "pushed out" of it by a slack $r > 0$. This is why BLEnd sets the listen interval to $L = A + b + s$, to guarantee that a beacon always overlaps a listen interval. On the other hand, this choice may lead to situations where *two* beacons from the same sender fall in the listen interval of the same listener, as also shown

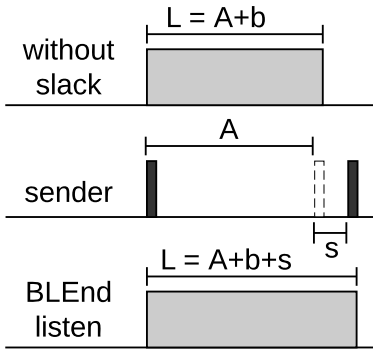


Figure 2.7: Compensating for slack.

in Figure 2.7. As a result, the number of beacons potentially colliding in (2.2) becomes $\gamma > C - 2$.

To quantify the impact of these extra beacons, we observe that they *may* occur only when a beacon is within s of the beginning (or end) of the listen interval. The sum $b + A$ of the beacon duration and the advertisement interval leaves enough room for the next (or previous) beacon to fall within L. Each node has a $\frac{s}{L-b}$ chance of choosing such a beacon starting time. However, the extra beacon is generated only in half of the cases, on average. To see why, consider Figure 2.7. The first beacon occurs at a start time $t = 0$ w.r.t. the beginning of the listen interval, and only the choice of the maximum slack $r = s$ *prevents* an extra beacon from occurring; in the dual case where the first beacon is sent at $t = s$ into the listen interval, only a slack of $r = 0$ *causes* an extra beacon.

This fraction of extra beacons can therefore be modeled as

$$\sigma = \frac{1}{2}\left(\frac{s}{L-b}\right) \tag{2.6}$$

and must be added to those potentially colliding "naturally", i.e., because of the base BLEnd schedule, as discussed in Section 2.4.1. This is accounted for by modifying the exponent of (2.2) into

$$\gamma = (C-2)(1+\sigma) \tag{2.7}$$

Beacon Start Time Dependence. In Section 2.4.1, two beacons colliding in one epoch would collide in all subsequent ones. In essence, it is as if we modeled the discovery probability in the first epoch, which then remains the same because each node behaves according to its periodic schedule. The random slack introduced by BLE mitigates this situation by randomly "nudging" each beacon; beacons that collide in an epoch are no longer *always* colliding in the subsequent ones, and can be discovered across multiple epochs. However, the beacons that collided in the first epoch are still *more likely* to collide also in subsequent epochs, simply because they are kept close to each other by the random slack. The model in (2.3)–(2.5) does not account for this dependency across epochs, as it assumes that colliding beacons may always appear anywhere across the entire interval $L - b$.

The probability P_{nc} of *not* colliding in the first epoch is the same as in (2.2). However, for epochs $k > 1$, this probability becomes $P_{nc,W} \leq P_{nc}$, due to the fact that the starting time of beacons is constrained by the schedule

in the first epoch. The discovery probability across k epochs in (2.3) becomes

$$P_{d,k} = 1 - (1 - P_{nc})(1 - P_{nc,W})^{k-1} \tag{2.8}$$

To derive the expression of $P_{nc,W}$, we observe that a beacon occurring at time t_i in the first epoch occurs within an interval W centered on t_i in subsequent epochs, due to the presence of the random slack. W represents the new *window of contention* for colliding beacons. If $W \geq (L - b)$, then the window is the same size or larger than the window used when assuming the beacons were randomly tossed in the interval $[0, L - b]$, and therefore P_{nc} in (2.2) still holds. However, when $W < L$, the probability of collisions *increases*, i.e., $P_{nc,W} \leq P_{nc}$, because beacons that collided once are more likely to collide again as they are somewhat loosely synchronized.

Estimating W is complicated by the fact that the slack introduced by BLE is added *relative to the start time of the previous beacon*, and therefore the offset among the same beacons in different epochs compounds across all beacons within an epoch. Consider the situation in Figure 2.8 and recall that, as per the BLE specification, the first beacon has no slack. Assume the extreme case where the value of the slack for all four remaining beacons is $r = 0$ for

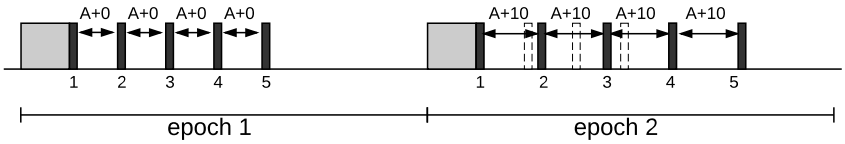


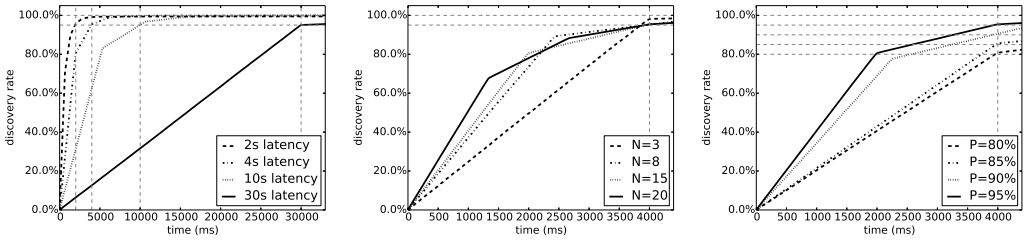
Figure 2.8: BLEnd compound slack. In epoch 2, positions of beacons from epoch 1 are shown with dashed lines.

epoch 1 and $r = 10$ for epoch 2. Looking at the start time of a given beacon in each epoch, it is clear that the offset across epochs increases with the position of the beacon; beacon 2 occurs 10ms later in epoch 2 w.r.t. epoch 1, while beacon 5 occurs 40ms later. This of course could go either direction; swapping the choice of r for the two epochs would result in beacon 5 occurring 40ms *earlier* in epoch 2 w.r.t. epoch 1.

More generally, assume that a sender's i^{th} beacon is sent at time t_i in the listener's first epoch. Then the sending time for the same sender's i^{th} beacon in the listener's second epoch (or any epoch $k > 1$) falls in the interval:

$$[t_i - (i-1) \times s, t_i + (i-1) \times s]$$

This expression captures the *maximum* window of contention for the same beacon across two consecutive epochs. The *average* interval is only *half* of the above, since it is 0 for the first beacon, s for the second one, $2s$ for the third, and $s(i-1)$ for the i^{th} beacon.



(a) $N = 15$, $P = 95\%$ **(b)** $\Lambda = 4s$, $P = 95\%$ **(c)** $N = 15$, $\Lambda = 4s$

Figure 2.9: Model-simulator validation for F-BLEnd.

Using these insights, and the fact that n_b is the total number of beacons sent in an epoch, we compute the *average* size W of the window of contention

across all beacons in the epoch:

$$W = \frac{s\sum_{i=1}^{n_b-1} i}{n_b - 1} = s\frac{n_b}{2} \tag{2.9}$$

We can then compute the probability of *not* having a collision in an epoch $k > 1$ as

$$P_{nc,W} = \begin{cases} \left(1 - \frac{2b}{W}\right)^{\omega} & \text{if } W < (L-b) \\ P_{nc} & \text{otherwise} \end{cases} \tag{2.10}$$

The first expression is similar to (2.2), with the denominator $L - b$ replaced by W. The number of beacons potentially colliding within the window W is

$$\omega = 1 + \frac{W}{L-b}(\gamma - 1) \tag{2.11}$$

Recall that $P_{nc,W}$ accounts for the discovery probability after a beacon collision in the first epoch; the first term represents such a collider, which is bound to fall within W. The second term represents the fraction of the γ beacons from (2.7) that may fall in the window W surrounding the sender's beacon, minus the collider already considered in the first term. This formulation may underestimate the case where multiple beacons collide in the first epoch; in practice, this model already returns good estimates, as shown in Sections 2.5 and 2.8.

The complete expression of the discovery probability P_d remains the one in (2.5). However, its component due to the spillover $P_{d,sp}$ in (2.4) must also be modified along the same reasoning that led to the modified expression of $P_{d,k}$ in (2.8):

$$P_{d,sp} = (1 - P_{nc})(1 - P_{nc,W})^{k-1}\left(\frac{\Lambda - kE}{E}\right)P_{nc,W} \tag{2.12}$$

Table 2.2: Model validation via simulation. Λ is in seconds; E and A are in milliseconds; probabilities P, P_{sim} and duty cycle DC are in percentage; lifetime is in days, assuming a battery capacity of 320mAh. We show the lifetime for both $I_{idle} = 0\mu A$ (LT_0) and $I_{idle} = 80.64\mu A$ (LT_{80}).

Configuration (optimizer)					**Simulation results**									
Input			**Output**		**F-BLEnd**					**B-BLEnd**				
N	Λ	P	E	A	P_{sim}	P_{sim}–P	DC	LT_0	LT_{80}	P_{sim}	P_{sim}–P	DC	LT_0	LT_{80}
3	4	95	3995	111	98.06	+3.06	5.93	37.45	30.61	97.16	+2.16	4.58	47.69	37.12
8	4	95	2430	106	95.25	+0.25	7.89	27.82	23.85	93.56	−1.44	6.74	32.21	27.01
15	4	95	1995	121	95.37	+0.37	9.38	23.19	20.37	93.91	−1.09	8.68	24.91	21.68
20	4	95	1335	91	95.36	+0.36	11.27	19.34	17.34	94.36	−0.64	10.54	20.59	18.34
15	2	95	667	71	95.74	+0.74	17.09	12.70	11.80	95.51	+0.51	16.32	13.25	12.28
15	4	95	1995	121	95.37	+0.37	9.38	23.19	20.37	93.91	−1.09	8.68	24.91	21.68
15	10	95	5363	149	95.11	+0.11	5.14	42.94	34.17	93.02	−1.98	4.36	50.12	38.57
15	30	95	29969	555	95.04	+0.04	2.49	86.93	57.22	93.37	−1.63	2.25	95.35	60.75
15	4	95	1995	121	95.37	+0.37	9.38	23.19	20.37	93.91	−1.09	8.68	24.91	21.68
15	4	90	2309	110	90.58	+0.58	8.22	26.70	23.03	88.28	−1.72	7.30	29.81	25.31
15	4	85	4000	174	85.51	+0.51	6.50	33.43	27.87	82.69	−2.31	5.95	36.30	29.83
15	4	80	3939	128	80.74	+0.74	5.96	37.00	30.30	76.97	−3.03	5.08	42.96	34.19

The first two factors account for the probability that the beacon collides in all the first k epochs, and the last captures the probability that the beacon does not collide in the spillover, accounting for the loose synchronization due to the slack.

The final expression of discovery probability across k epochs from (2.5) can therefore be rewritten as:

$$P_d = 1 - (1 - P_{nc})(1 - P_{nc,W})^{k-1} \left(1 - \left(\frac{\Lambda - kE}{E}\right) P_{nc,W}\right) \tag{2.13}$$

2.5 Simulating BLEnd

To confirm the correctness of our model, we created a discrete event simulator in Java that steps through the BLEnd schedules of multiple nodes, simulating discoveries. The simulator accounts for collisions (or optionally ignores them), considers the three BLE advertisement channels, and implements all three versions of BLEnd. The simulator allows us to cross-validate the model underlying the optimizer with the protocol behavior observed in simulation.

Simulation Setup. We ran our simulations using the same assumptions made in the model. Further, we guarantee that any two nodes' epochs do not start within b time of each other, as this aspect is currently not captured by our model. Our evaluation tests the three dimensions of application requirements (i.e., discovery latency, node density, and discovery probability) by fixing two of them and varying the other. For each combination, we plot the cumulative distribution function (CDF) of discovery latencies and show target probabilities (as horizontal, dashed lines) and target latencies (as vertical, dashed lines); each curve represents 10,000 independent simulation runs. For lifetime, we assume a battery capacity of 320mAh—the same of the SensorTag we use in Section 2.8.

Model-simulator Validation: Bi-directional Discovery. Figure 2.9 shows the simulation results for F-BLEnd, which, among the BLEnd variants, is most accurately represented by the model in Section 2.4. We note a number of in-

flection points in each curve, caused by the fact that the target latency may be composed of multiple epochs; inside each epoch, BLEnd shows a constant discovery rate that decreases in each subsequent epoch as fewer nodes remain to be discovered. In all tested scenarios, our simulator produces discovery probabilities and latencies in line with the established targets. This is seen in Figure 2.9 by noting the difference between each CDF and the target discovery probability (horizontal line) at the target latency (vertical line). Table 2.2 shows the same data numerically, showing that the simulated F-BLEnd discovery probability is always higher than the target for the associated latency; the difference is always below 1%, except for the first combination.

Table 2.2 shows also the results for B-BLEnd, whose CDFs are here omitted due to space limitations, confirming that the model and the associated optimizer are effective at generating realistic BLEnd configurations enabling bi-directional discovery while accounting for BLE constraints and beacon collisions. Further, Table 2.2 also allows us to quantify the benefits brought by the on-demand beacon scheduling of B-BLEnd w.r.t. the naïve solution provided by F-BLEnd. B-BLEnd offers 4 to 27% improvement in lifetime w.r.t F-BLEnd in the configurations studied, therefore confirming quantitatively that it provides a more efficient solution.

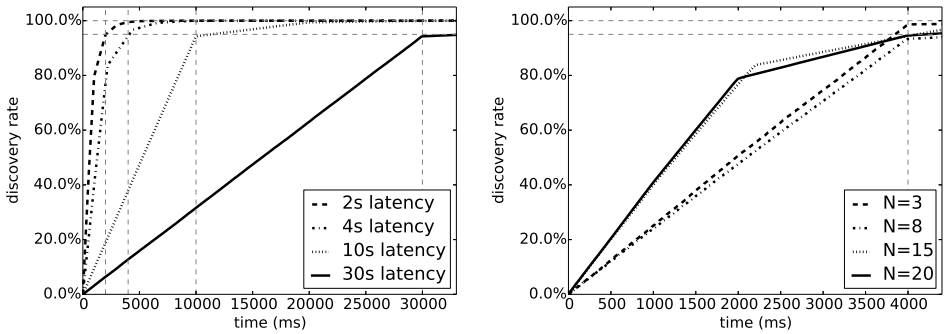
In the case of B-BLEnd, however, Table 2.2 shows that the simulator yields a discovery probability slightly lower than the target, except for two of the combinations. This is due to the fact that our model and the associated optimizer are based on F-BLEnd, for the reasons mentioned in Section 2.4,

and therefore do not completely capture all intricacies of B-BLEnd.

In particular, a subtle dependence among beacons occurs in B-BLEnd that does not exist in F-BLEnd. Remember that F-BLEnd schedules all beacons in all cases and that, in general, when $P < 100\%$ the optimization does not require discovery of *all* nodes. Therefore, a valid configuration output by the optimizer may allow A to not detect B due to a collision. While this results in a discovery loss that is properly accounted in F-BLEnd, in B-BLEnd it may lead to a secondary loss that the model does not consider. Specifically, bi-directional discovery is achieved in B-BLEnd by the explicit addition of a beacon, triggered by detection. Therefore, if A does not detect B due to a collision, the additional beacon is not scheduled and B does not discover A, thus reducing the discovery rate. This motivates future work to improve the accuracy of our model and optimizer by explicitly considering the subtleties of B-BLEnd.

Model-simulator Validation: Uni-directional Discovery. We now investigate the relative performance of U-BLEnd. Although the use cases of uni-directional and bi-directional discovery differ, the expectation is that uni-directional discovery will save a significant amount of energy. These measurements also validate the application of the model to uni-directional U-BLEnd.

Figure 2.10 and Table 2.2 show that our simulation results come very close to the target discovery probability P in most cases. Recall that P is computed in U-BLEnd as the probability that at least one of each pair of nodes discovers the other within Λ time. U-BLEnd does miss the target discovery



(a) $N = 15$, $P = 95\%$ **(b)** $\Lambda = 4s$, $P = 95\%$

Figure 2.10: Model-simulator validation for U-BLEnd.

latency by a small amount in all but two parameter combinations. The reason is similar to the one discussed for B-BLEnd; essentially, the discovery rate in U-BLEnd is penalized twice for each missed detection. Nevertheless, these results do indicate that, when bi-directional discovery can be ascertained offline, U-BLEnd provides significant benefits over B-BLEnd. Meeting the same application requirements in terms of N, Λ, P with U-BLEnd and B-BLEnd yields comparable discovery rates, but U-BLEnd lifetime that is up to 1.8x longer than the one of B-BLEnd.

2.6 Comparing to the State of the Art

In our simulator, we also implemented both Searchlight [13] and Nihao [101] exactly as described in the literature. This allows us to compare against these recent protocols that are considered the best performing state of the art protocols but do not directly support BLE and whose implementations are not publicly available.

Table 2.3: Model validation via simulation, for U-BLEnd. Units are the same as in Table 2.2.

Configuration (optimizer)					**Simulation results**				
Input			**Output**		**U-BLEnd**				
N	Λ	P	E	A	P_{sim}	P_{sim}–P	DC	LT_0	LT_{80}
3	4	95	3999	80	98.51	+3.51	4.24	52.27	39.84
8	4	95	3983	83	93.26	−1.74	4.25	51.98	39.66
15	4	95	2197	69	94.36	−0.64	5.98	36.68	30.09
20	4	95	1998	72	94.58	−0.42	6.42	34.07	28.31
15	2	95	1001	51	94.74	−0.26	9.39	23.24	20.40
15	4	95	2197	69	94.36	−0.64	5.98	36.68	30.09
15	10	95	10000	232	94.24	−0.76	3.15	68.71	48.71
15	30	95	29951	234	94.26	−0.74	1.51	146.59	78.15
15	4	95	2197	69	94.36	−0.64	5.98	36.68	30.09
15	4	90	3995	111	88.97	−1.03	4.52	48.30	37.48
15	4	85	3999	80	84.89	−0.11	4.24	52.27	39.84
15	4	80	3999	80	84.89	+4.89	4.24	52.27	39.84

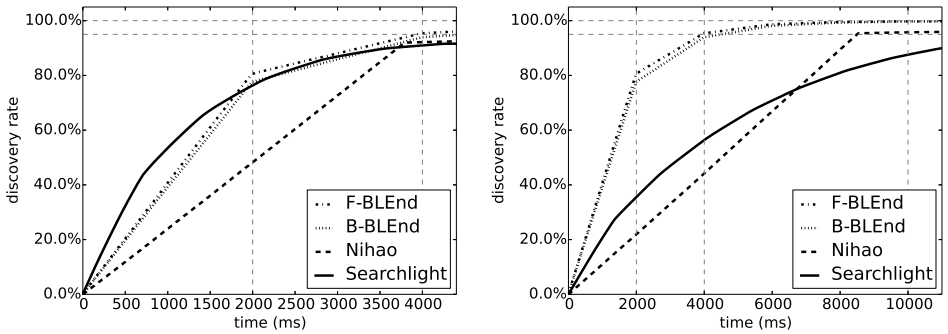
In both protocols, a key parameter is the slot duration, which is also the "unit of measure" of latency and duty cycle. Recall that duty cycle is a commonly used proxy for energy consumption; for clarity we report both duty cycle and battery lifetime for all protocols. Unfortunately, neither paper provides guidance on selecting an appropriate slot size. However, in Nihao α captures the ratio between the advertisement event duration and the slot duration; the value $\alpha = 0.054$ is used in [101]. In our hardware, a BLE advertisement event lasts 3.2ms, yielding a slot duration of 59.26ms. We use this value for both protocols.

Searchlight builds a schedule around a fixed period of t slots, which contains one "anchor" active slot at the beginning and a second "probe" slot

somewhere in the first half of the period. The value of t relates directly to the schedule's duty cycle ($\frac{2}{t}$) and target discovery latency ($\frac{t^2}{2}$). A (Balanced) Nihao schedule is instead built around a parameter n; one period of the schedule consists of n^2 active slots. The node beacons at the beginning of every n^{th} active slot and then listens for the first n slots. The value of n is similarly tied to duty cycle ($\frac{1+\alpha}{n}$) and latency (n^2). In the following, we leverage these relationships to set t and n based on a target discovery latency (or duty cycle) equivalent to that achieved by BLEnd in a given configuration.

Protocol comparison. Since Searchlight and Nihao both provide bi-directional discovery, we compare only our F-BLEnd and B-BLEnd variants against them. For all protocols, we use a target latency $\Lambda = 4s$. For BLEnd, we set a target discovery probability of $P = 95\%$ with $N = 15$ nodes; the other protocols attempt to reach 100% because, unlike BLEnd, they do not offer any alternative.

Figure 2.11a shows the results; notably, neither Searchlight nor Nihao reach 100% discovery, despite that this is their goal. The reason is two-fold: if two nodes choose nearly aligned slot start times, the nodes are always sending simultaneously and never discover each other; and the probability of collisions among beacons is not accounted for in these protocols. Instead, the BLEnd variants do take these aspects into account, via the model in Section 2.4; BLEnd succeeds in meeting a discovery rate around around the 95% targe. Figure 2.11a also shows that Searchlight most quickly discovers nodes at the beginning, though all protocols reach better than 92% discovery probability at or before the target latency. Nihao reaches its inflection point before the



(a) [Fixed latency: $N = 15$, $P = 95\%$, $\Lambda = 4s$

(b) Fixed duty cycle: $N = 15$, $P = 95\%$, $DC = 8.77\%$

Figure 2.11: Comparing BLEnd against the state of the art.

target discovery because its period is computed as n^2 slots; to hit a target latency, one must divide the latency into slots then take the square root. We conservatively choose n to be less than the square root as doing otherwise would cause Nihao to miss the target. In Figure 2.11a, Nihao uses $n = 8$; with a value of $n = 9$, Nihao misses the target latency by 850ms. This highlights the significant benefit that BLEnd is not artificially constrained by a rigid slotted structure.

Although all protocols, despite their different goals and configurations, have roughly the same discovery probability at the target latency, it is worth analyzing the expended energy. Table 2.4 shows the percentage of neighbors discovered, duty cycle, and resulting lifetime (based on different values of I_{idle} as in Table 2.2) when all protocols are configured with a target latency $\Lambda = 4$s, as considered thus far. Both BLEnd variants clearly outperform the competitors, with an expected lifetime for B-BLEnd that is 1.5x and 2x higher

than Nihao and Searchlight, respectively.

To offer a dual perspective, we configured Searchlight and Nihao with a target duty cycle instead of a target latency, to observe their performance when given roughly the same energy budget of B-BLEnd. Therefore, we configured both to achieve a target duty cycle as close as possible to $DC = 8.68\%$. The results are shown in the last two rows of Table 2.4 and in Figure 2.11b, and show that BLEnd clearly outperforms the competition; both Searchlight and Nihao discover almost half of the nodes when given roughly the same energy budget as BLEnd. The duty cycles of Searchlight and Nihao are actually slightly higher than B-BLEnd; this is because their slotted operation significantly reduces the configuration options for these protocols. We chose the closest matching configuration, which while having a slightly higher energy usage still perform significantly worse in terms of discovery probability. This confirms that BLEnd, thanks to its flexible unslotted operation and the ability to take into account collisions, efficiently uses the available energy budget to

Table 2.4: Discovery probability vs. lifetime for $\Lambda = 4s$. BLEnd variants are configured with $N = 15$, $P = 95\%$. Notation and units are the same as in Table 2.2.

Protocol	P_{sim}	DC	LT_0	LT_{80}
F-BLEnd	95.37	9.38	23.19	20.37
B-BLEnd	93.91	8.68	24.91	21.68
Searchlight (target: latency)	90.96	18.52	11.53	10.79
Nihao (target: latency)	92.16	13.20	16.12	14.70
Searchlight (target: duty cycle)	56.36	9.66	22.11	19.53
Nihao (target: duty cycle)	44.19	8.81	24.16	21.11

Table 2.5: Implementation validation. Λ is in seconds; E and A in milliseconds; P and P_{eval} are percentages.

Configuration (optimizer)					**Results**			
Input			**Output**		**F-BLEnd**		**B-BLEnd**	
N	Λ	P	E	A	P_{eval}	$P_{eval} - P$	P_{eval}	$P_{eval} - P$
3	2	95	2000	77	97.70	+2.70	95.98	+0.98
8	2	95	1055	66	95.18	+0.18	95.72	+0.72
3	4	95	3995	108	96.58	+1.58	94.16	−0.83
8	4	95	2430	106	95.48	+0.48	94.39	−0.61
3	10	95	9975	172	97.22	+2.22	93.16	−1.84
8	10	95	9986	253	96.26	+1.26	93.00	−2.00

Configuration (optimizer)					**Results**	
Input			**Output**		**U-BLEnd**	
N	Λ	P	E	A	P_{sim}	$P_{eval} - P$
3	2	95	2000	53	95.90	+0.90
8	2	95	1991	83	92.86	−2.14
3	4	95	4000	77	96.43	+1.43
8	4	95	3983	83	95.28	+0.28
3	10	95	9999	122	98.81	+3.81
8	10	95	9999	122	94.89	−0.11

meet the target discovery and latency application requirements.

A Stress-Test for BLEnd. To push the limits of B-BLEnd, we considered a configuration inspired by a small conference scenario ($N = 100$, $\Lambda = 10s$ and $P = 90\%$). First, when the configuration from the optimizer is fed to the simulator, B-BLEnd reaches 91.94% discovery by the target latency, exceeding the target discovery probability. Next, we observe that optimizer output yields an epoch $E = 3334$ms and an advertising interval $A = 444$ms. This implies that each epoch contains only 6 advertisements, a surprising result given the high node density. This schedule goes against the *talk more, listen less* principle

driving Nihao, showing that "talking more" is not required at high density, as the same beacon enables discovery by multiple nodes. Interestingly, BLEnd is flexible enough to accommodate the advertising strategy best suited to the node density at hand, as configured via the optimizer.

2.7 A Practical Implementation

In this section, we relate the abstract description of the BLEnd protocols in Section 2.2 to our specific implementation on the TI CC2650STK SensorTag. The latter is based on the CC2650 wireless MCU, which contains a 32-bit ARM Cortex-M3 processor, a 128kB programmable memory, 20kB of SRAM, and a complete system-on-chip BLE solution. The radio module uses a 2.4GHz RF transceiver fully compatible with BLE 4.2, with a receiver sensitivity of -97dBm and a range up to 50m/160ft.

Uni-directional Discovery. Our implementation of U-BLEnd delegates the protocol timing largely to the timers available in the BLE stack. An epoch always begins with a listening interval, implemented by initiating a BLE scan for the `scanDuration` established by the optimizer, $L = A + b + s$. As soon as the listening interval ends (indicated by the `GAP_DEVICE_DISCOVERY_EVENT` generated by the BLE stack), the protocol initiates advertising, with the `advertisingInterval` from the optimizer, A. A single BLE advertisement event duplicates the advertisement on BLE's three advertising channels (37, 38, and 39).

Bi-directional Discovery. To implement F-BLEnd atop U-BLEnd, we simply let the BLE advertising continue until the end of the epoch. However, the more efficient B-BLEnd requires each advertisement beacon to include the time until the next listen. The receiver uses this to activate the correct additional beacon for bidirectional discovery. Unfortunately, updating advertisement data inside a BLE beacon is non-trivial. Advertising must be stopped, the data updated, then advertisement can be restarted. As a result, in B-BLEnd, all timing between beacons (including BLE's random slack) is handled in application space.

Storing Data into Advertisements. A major constraint of implementing continuous neighbor discovery on BLE without pairing is that all exchanged information must fit inside an advertisement. In BLE, an entire beacon is always sent, regardless of whether it contains usable application data. Practically, SensorTag BLE advertisements have 31B of application-writeable data [19, 120]. In BLEnd, we use 5B to identify packets as belonging to the BLEnd protocol (though less could conceivably be used) and 2B to carry a unique node identifier, enabling other nodes to determine which node has been discovered. B-BLEnd also needs to include the time to the next listen interval (2B) to enable receivers of the beacon to opportunistically schedule their beacons in the second half epoch.

2.8 Real-world Evaluation

In this section, we report results from experiments with the SensorTag using the implementation described above. Our goal is to verify that our implementation matches the results from the simulator and the predictions of the model in terms of meeting the application requirements concerning discovery probability P within a given latency Λ and for a given node density N. We setup an indoor testing environment with 8 SensorTags in a 4×2 grid with inter-node spacings of 30cm/11.8in. We used an additional node that continually scanned the three BLE advertisement channels. This node was connected to a desktop computer for data collection.

In a run, each node first performs an initialization (e.g., setting BLEnd parameters, initializing the BLE stack, etc.) then emulates a *random arrival* in which it remains inactive for a random time $t_r \in (0, E)$ before starting to run the BLEnd protocol. This process simulates a real scenario in which participants move into range of one another at different times. To achieve randomness, we use the True Random Number Generator (TRNG) from the CC2650 MCU.

In our evaluation, we use the remaining space available in beacons to carry information useful to our experiments. To monitor energy consumption, we include the SensorTag battery level (2B) in each beacon. In addition, when a node discovers a new neighbor for the first time, it adds the corresponding timestamp (2B) to the beacon payload. Given the available space in the beacon, a node can convey information for up to 10 discovered neighbors.

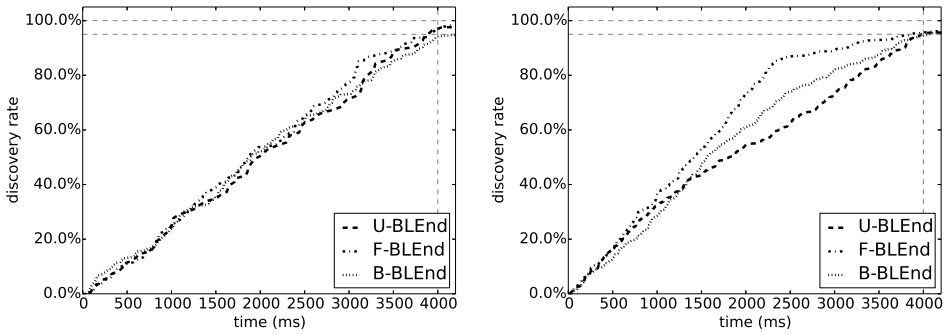
However, note that this 10-node limitation is only an artifact of data collection for our experimental setup and is not a restriction on BLEnd itself, which can support an arbitrarily large number of nodes. In our experiments, the sink node scans for these beacons and transfers the results to the desktop using the SimpleLink Debugger DevPack [1].

Table 2.5 reports tests for 3- and 8-node experiments with a target discovery probability $P = 95\%$ and different values for the target latency Λ. We used the optimizer to generate the $\langle E, A \rangle$ configuration for each BLEnd variant. The results are the average of 30 experiments for each combination of parameters. By considering the difference $P_{eval} - P$ between the measured and target discovery probability we see that our experiments track the target discovery rate within a few percentage points and, for all three protocols are in line with the expectations from the simulation. F-BLEnd always exceeds the target discovery probability as in simulation, while B-BLEnd and U-BLEnd narrowly miss the target in some cases, again similar to the results from simulation. As already discussed, this stems from the fact that the model in Section 2.4 does not capture all the subtleties of these two BLEnd variants.

Figure 2.12 provides a finer-grained perspective by showing the empirical CDFs of all three BLEnd variants for 3- and 8-node experiments with a fixed target latency Λ=4s and discovery probability of $P = 95\%$. We again show the target latency as a vertical dashed line and target discovery proba-

[1] `http://www.ti.com/tool/cc-devpack-debug`

bility as a horizontal dashed line. Figure 2.12a shows that, with $N = 3$, the discovery rate of all protocols steadily increases up to the target. This is a consequence of the fact that the epoch identified by the optimizer is very close ($E = 3995$ms for F-BLEnd and B-BLEnd) or equal (U-BLEnd) to the target latency Λ, as collisions are rare. Increasing the density to $N = 8$ bears little effect on U-BLEnd, whose epoch $E = 3983$ms remains close to Λ, and whose discovery rate is only marginally affected, as shown in Figure 2.12b. On the other hand, the optimal epoch becomes significantly smaller ($E = 2430$ms) for the bi-directional cases, to account for the increase in number of beacons and therefore collisions. F-BLEnd shows an inflection point precisely at that this point; discoveries occurring in the second epoch are due to collisions in the first one. In contrast, the discovery rate in B-BLEnd is smoother as this protocol sends significantly fewer beacons than F-BLEnd in the second half of the epoch. However, discoveries occur at a slower rate, precisely due to the fewer beacons in the first epoch and the need to explicitly schedule beacons in the second epoch to enable discovery. This is visualized effectively by Figure 2.12b, showing that the rate at which nodes are discovered in B-BLEnd sits between F-BLEnd and U-BLEnd—a consequence of its design that trades off the speed at which nodes are discovered for better energy consumption. In any case, for all BLEnd variants, the shape of the discovery rate curves matches very closely the results from simulation, as shown, e.g., in Figure 2.12a and 2.12b.



(a) $N = 3$, $P = 95\%, \Lambda = 4s$ **(b)** $N = 8$, $P = 95\%, \Lambda = 4s$

Figure 2.12: Experimental evaluation of BLEnd.

2.9 Conclusion

We presented BLEnd, a protocol designed with practical concerns in mind. On one hand, we choose BLE as a communication platform and design a protocol that, at its core, takes BLE's peculiar constraints into account. On the other hand, we put application users at the center and devise a protocol that is easily configurable to meet application requirements. These requirements include a target discovery latency, and consider, for the first time in the literature, the practical impact of collisions on continuous neighbor discovery.

The cornerstone of BLEnd is the realization that unidirectional discovery can be accomplished very efficiently without precluding its extension to bidirectional discovery, if and when needed. Further, the resulting protocol is adaptive, in that it automatically adjusts the amount of beaconing as density increases. Our evaluation in simulation shows quantitatively that BLEnd is significantly more performant than state of the art protocols once the latter are placed in the context of real practical constraints. Our design is reified in

an implementation on TI SensorTags, but the key element enabling practical use is the companion optimizer, which enables users to identify the best configuration for a given set of application requirements. The optimizer is based on an analytical model of BLEnd, and we showed that model, simulator, and implementation are in good agreement, thereby enabling immediate use in applications.

Chapter 3

SCENTS: Collaborative Sensing in Proximity IoT Networks

The past decade has seen a rise of IoT devices with multiple low-powered commodity sensors and the ability to communicate wirelessly with nearby devices. Such devices are found in myriad domains, from wearables to home automation and smart city infrastructure. From a sensing perspective, an IoT device periodically sends readings from on-board sensors to a locally hosted application, to a paired device, or to the "cloud" via a gateway. This design works well for enterprise applications and smart homes. However, the popularity of applications that rely on *personal* devices remains low. Many of these devices are battery-operated, which limits the number of on-device sensors and the sampling frequencies. On the communication front, nearly all mobile devices are capable of short range wireless communication (e.g., via Bluetooth, ZigBee, etc.), and for some devices short-range connections are the *only* option. However, the use of these links to make sensed data available in the vicinity is underdeveloped.

Many envisioned pervasive computing applications rely on *continuously* sensed information about the surroundings, but the cost of continuous sensing

on battery-operated devices can be prohibitively high [14]. It is therefore reasonable to enable devices to collaborate to sense a collective context state. For instance, children participating in a walking school bus may share movement or activity patterns, indicating a shared route to school [97]. The use of a device-to-device network to allow co-located IoT devices to collaborate could: (1) enable user-facing applications to leverage sensing capabilities of nearby devices when the local device lacks some capability; (2) allow nearby devices to save energy spent on sensing "similar" values; and (3) enable sensing-heavy applications to be less reliant on infrastructure connections. In addition to bringing computation closer to the user [1], this last point eases concerns associated with offloading potentially private data to a third party [121].

Section 3.1 describes existing work in collaborative context sensing, which often involve cloud-based control [43, 112]. Rather than addressing an off-line data collection goal, we aim to satisfy *local needs* for sensed context using *local sensing resources*. We propose SCENTS (Sensing Collaboratively in Everyday NeTworkS), which allows devices to leverage local, device-to-device communication to *actively* request and supply locally sensed context. The key contributions of SCENTS are:

- We construct a framework that utilizes commonly available connectionless communication to share sensing capabilities directly among co-located heterogeneous IoT devices.
- We devise a heuristic to identify the best mechanisms for sensing, ac-

counting for sensing and communication costs, and predicting and adapting to mobility-induced failures.

- We evaluate SCENTS on an extensive set of IoT scenarios using an expressive and realistic smart city simulator.

3.1 Related Work and SCENTS Vision

To motivate SCENTS, we introduce two application scenarios that we revisit throughout the paper. We then discuss efforts related to SCENTS, described in the next section.

Application Scenarios. Consider a walking tour group [109] whose participants carry smartphones that inform them about the surroundings: weather, nearby crowds, wait times for points of interest, etc. Participants' devices could individually collect all of the needed information. However, these devices need to communicate simply to maintain the group's digital connectedness. By leveraging these group messages, devices can also share the burden of sensing needed ambient context.

As a second scenario, imagine a jogger in a smart city. The jogger does not desire to carry a bulky smartphone but instead wears only a simple watch with wireless connectivity but no sensing capabilities. This device can opportunistically collect information from devices embedded in the municipal infrastructure [3]. In this way, the jogger can log run details, from pace to the weather or crowd conditions. This information is similar to the details that

smartphone running apps currently collect, but with less cost and burden to the jogger.

Related Work. The IoT raises new challenges in both sensing and collaborating among highly-capable yet battery-operated devices [1, 26]. Many efforts simplify applications by abstracting or virtualizing their use of sensors [23, 74] or by enabling access to sensors on nearby devices [6, 136]. Even given these efforts, sensing remains a major cost of IoT applications [3]. Other approaches allow devices to rely on edge services to perform sensing on their behalf [102] or intelligently task sensors based on predicted high-level context values, thereby reducing sensing cost [62, 92].

Cloud-assisted collaborative sensing aims to maximize sensing coverage or improve data quality [43, 64, 112]. Most approaches collect information for use offline; a few push data back to distributed devices [25, 87]. In sensor networks, local neighborhoods [91, 129] aggregate collected data before sending it to a gateway. The challenge of uncoordinated collaboration in a frequently changing network using only a local view remains open. Work in device-to-device collaboration [53, 99] sets the stage for our efforts, and defining long-lived groups of co-located users for the purpose of sharing local information has a growing interest in smart cities [35, 71, 109]. Yet opportunistic context sharing over highly dynamic links demands transient light-weight abstractions.

Our own prior work shares context among co-located devices [28, 80]. These prior efforts are passive and opportunistic. In SCENTS, in contrast, the surroundings become an extension of the device's own capabilities that

applications on the device can abstractly access on-demand.

Background. In SCENTS, devices must opportunistically discover what sensing capabilities are available nearby. An important technical building block is *continuous neighbor discovery*, made possible by the myriad communication capabilities in the IoT (e.g., BLE, IEEE 802.15.4). These widely used protocols create schedules of sending and receiving *beacons* to enable discovery of nearby devices [13, 37, 67]. SCENTS places contents related to shared sensing in these protocols' periodic beacons sent by the BLEnd protocol(section 2.2).

3.2 Collaborative Sensing in SCENTS

In SCENTS, we consider the *application* and the device *hardware* as two layers of an IoT node, where an application generates queries for sensor information, and the device uses sensing and communication capabilities to satisfy queries. SCENTS sits between the two and governs collaboration among nodes. We first formulate the dynamic collaborative sensing problem and then present the details of SCENTS.

3.2.1 Problem Formulation

We start by establishing some terminology.

- *IoT node* $\underline{d}$: a device capable of sensing and wireless communication. We indicate a neighborhood as $d_1, \ldots, d_n$.

- *Context type* $\underline{s}$: a type of context a sensor provides (e.g., location). Each node d_i provides $S_i \subseteq S$; the complete set of context types, $S = \{s_1, \ldots, s_m\}$, is known *a priori*.

- *Sensing energy cost* $\underline{e}$: the value e_k is an averaged energy cost for sensing type s_k.

- *Link stability* $\underline{l_{i,j}(t)}$: how stable the link between d_i and d_j is at time t; $l_{i,j} \in [0, 1]$.

- *Device sensing cost* $\underline{E_i}$: node d_i's energy cost of sensing.

- *Energy capability* $\underline{cap_i}$: denotes to the energy capability of d_i. It is either d_i's battery capacity or ∞, if d_i is hardwired.

We now define the problem of *dynamic collaborative sensing*, using two metrics. The Fulfillment Ratio is defined as:

$$\mathsf{FR} = \frac{\text{Number of queries that receive a valid response}}{\text{Number of queries generated}} \tag{3.1}$$

The Unfairness of Energy Consumption metric captures the difference between the normalized energy consumption of different devices.

$$\mathsf{UEC} = \max_{(i,j)\in\{n\times n\}} \left(\left| \frac{E_i}{\text{cap}_i} - \frac{E_j}{\text{cap}_j} \right| \right) \tag{3.2}$$

Problem (Dynamic Collaborative Sensing): *A set of nodes* $D = \{d_1, d_2, \ldots d_n\}$ *with heterogeneous sensors* ($S_i \subseteq \{s_1, \ldots, s_m\}$) *move in a shared physical space. Each* $d_i \in D$ *hosts one or more applications that periodically*

generate sensing queries. For each query $q_i(t)$, the node d_i must decide the best node (either itself or a neighbor) to fulfill the query such that: for a given time period T, (1) as many queries as possible are answered, i.e., FR *is maximized; and (2) the unfairness of the normalized sensing cost of the participants is minimized, i.e.,* UEC *is minimized.*

Most sensed context that SCENTS targets, e.g., attributes of the ambient environment, can be made public without privacy concerns. Other data may be sensitive or could lead to privacy breaches through context inference. Previous study shows sharing these data only with people in close proximity raises fewer concerns [130]. Also, efforts exist to protect sensed data through encryption [131] and obfuscation [79]. SCENTS remains useful even considering only ambient context that is not privacy sensitive, including sensing that a device can delegate to a nearby device for cost reasons.

3.2.2 System Overview

As shown in Fig. 3.1, SCENTS sits between applications and the device hardware and has two primary components: the *neighborhood agent* and the *collaboration agent*. The former continuously detects arriving and departing neighbor devices using beacons. The latter intercepts applications' queries, selects and invokes the best approach to satisfy a query, and delivers data back to applications. It sends sensing requests by updating the beacon content of the neighborhood agent and receives sensing responses when the neighborhood agent receives fulfillers' beacons. The best approach to satisfy a query could

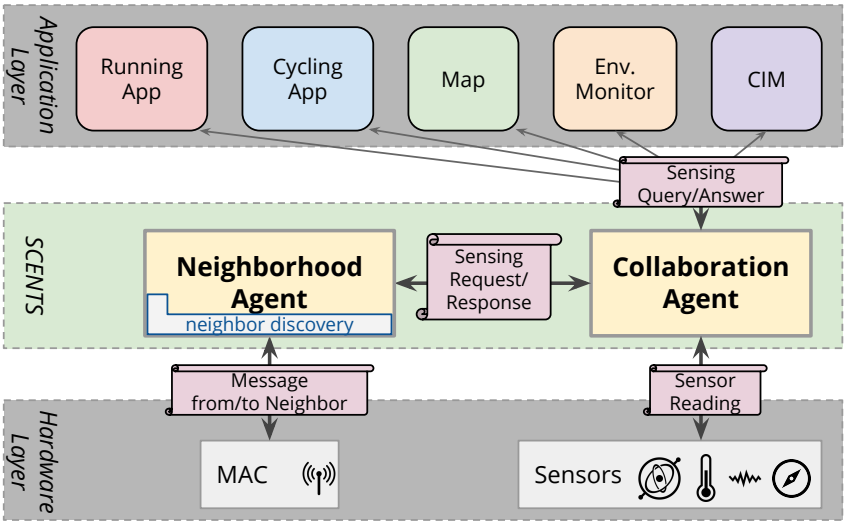


Figure 3.1: SCENTS system overview.

be to (1) sense the desired context using local hardware; (2) return a recent observation; or (3) communicate with neighboring devices to request and receive context.

3.2.3 Neighborhood Agent

When current IoT systems use a remote device for sensing (e.g., a smartphone using a wearable fitness sensor), the communication requires a pairing process. Once paired, the devices enter a "central/peripherals" model, where access to a peripheral's sensing capability is restricted to a *single* application (process) at a time. In SCENTS, all devices act as equals and direct their own behaviors. Any sensing participant can opportunistically communicate with any peer in range.

To support mobile applications, the neighborhood agent must quickly

adapt to changes in the surrounding network and its sensing resources. We presume the use of the BLEnd protocol for continuous neighbor discovery [58]. This protocol is characterized by a repeating schedule of short *beacons* and one longer *scanning* period that captures beacons from other nodes. The size of the beacon fixed by the underlying BLE technology, but only the id of the sender is necessary for the BLEnd protocol, leaving the remaining beacon payload *unused*. The neighborhood agent leverages these unused bits for sensing exchanges. The most relevant parameter of the underlying protocol is Λ, the expected maximum latency to discover a neighbor *with high probability*; for our purposes, Λ equates to the expected maximum time for a node to receive a neighbor's sensing request, r, or sensing response, p.

The neighborhood agent must also detect node departures. When d_i receives a beacon from d_j, the neighborhood agent infers the distance between d_i and d_j from the received signal strength (RSS) value $P_{Rx,i}$. We use the log distance path loss model [16, 41] to estimate distance from RSS:

$$PL(dBm) = P_{Tx,j} - P_{Rx,i} = PL_0 + 10 \cdot \gamma \cdot \log(a_{i,j}/a_0) + X \qquad (3.3)$$

where PL is the path loss signal strength, $a_{i,j}$ is the distance between d_i and d_j, γ is the path loss exponent, X denotes a zero-mean Gaussian variable caused by flat fading, and PL_0 is the path loss signal strength at reference distance a_0. Node d_i creates a distance queue, A_j, to keep time-stamped distances

from d_i to d_j based on recently received radio frames. The neighborhood agent computes the (relative) velocity of d_j:

$$V_j(t) = \frac{dA_j}{dt} \tag{3.4}$$

Using this relative velocity, the neighborhood agent estimates the stability of d_j relative to a sensing task using the *safe distance* [53] to determine how likely d_j is to move out of range before the sensing request and response complete. We denote the estimated communication range of d_j as R_j, calculated using equation 3.3. If d_i and d_j have different transmission ranges, d_i can still calculate the communication range of d_j at no additional cost[1]. Because SCENTS requires bidirectional communication, the neighborhood agents at d_i and d_j use the minimum of the two ranges. The neighborhood agent computes a smaller range, $r^{th}(t)$, that accounts for the times to send, receive, and respond to a sensing request, relying on the worst case delay in each direction (i.e., Λ):

$$r_j^{th}(t) = R - \Lambda \cdot |V_j(t)| \tag{3.5}$$

We use a logistic function to model the fact that the computed stability of a sensing partner decreases as the two nodes move away from each other:

[1]For instance, TxPower field in the BLE extended header.

$$l_j(t) = \frac{1}{1 + e^{\frac{1}{10} \cdot (a_{i,j} - \frac{r_j^{th}(t)}{2})}} \tag{3.6}$$

where $l_j(t)$ is node d_i's stability value for d_j at time t.

The neighborhood agent later uses stability to compute whether a neighbor makes a good collaborating partner. To account for the potential change in $l_j(t)$ over time, we compute $L_j(t')$, the aged stability of d_j at time t' . The aging factor uses an exponential decay; the exponential decay constant Λ is the maximum latency used above:

$$L_j(t') = l_j(t) \cdot e^{-\Lambda(t'-t)} \tag{3.7}$$

The neighborhood agent receives beacons, computes stability values, and passes beacon contents on to the collaboration agent, which is SCENTS's decision-making process.

3.2.4 Collaboration Agent

The collaboration agent has three main components: a *query interface*, a *neighbor cache*, and a *decision process*.

Query Interface. Interactions with the collaboration layer are driven by queries created by application; applications remain agnostic to how or where context is sensed. The interface contains a `query` method that takes a context type s_k and a handle to a callback to be invoked when the context is ready.

The collaboration agent maintains a local view of nodes in proximity, building and maintaining a neighbor cache over time using information from the neighborhood agent.

Neighbor Cache. In the neighborhood agent's schedule of scanning and beaconing, the default beacon content is a bitmap indicating the on-board sensors. That is, d_j's beacon contains a compressed vector of its capabilities, S_j, in which each bit uniquely identifies a type of context. The collaboration agent logically maintains a matrix $C \in B^{n \times m}$ in which bit C_{jk} denotes whether d_j is capable of sensing s_k. This matrix is updated whenever the neighborhood agent receives a beacon.

The neighbor cache also learns from the content of received beacons. If the beacon contains a response directed from d_j to this node, then the node delivers the data to the application. The neighbor cache also stores received context values in case they are useful for another application in the near future. It creates an entry $\langle s_k, \mathsf{value}, t \rangle \in \mathsf{Store}$, where s_k is the type, value is the measured value, and t is the time of the measurement. Expired entries are eliminated periodically given a mapping of type s_k to the expected duration of its validity.

The beacon could also contain a sensing *request* for d_i. The node first checks its local Store for a valid sample; if it finds one, it sends the cached value as a *reused* value. Otherwise, d_i samples the requested sensor, sends a response, and updates the Store with the new reading.

Finally, if the beacon contains a response for a node *other than* d_i, the content is still used to update d_i's Store if the value is newly sensed, i.e., not marked as reused. This allows d_i to cache sensor readings for future application requests. In addition, the collaboration agent monitors the estimated energy expenditure for sensing on nearby nodes. If a received beacon indicates that d_j recently sensed s_k, then d_i updates its local estimate of d_j's energy expenditure by updating a vector $E_j \in \mathbb{R}^n$ to be $E_j + e_k$. Over time, d_i's estimate of E_j accumulates the energy cost of d_j's sensing actions that d_i observes.

Decision Process. The final obligation of the collaboration agent is to resolve local application queries. When the collaboration agent receives a query $q(t)$ for type s_k, it checks the Store to see if a valid reading exists. If so, the query is fulfilled immediately. Otherwise, it determines the best way to use sensing resources in the vicinity to satisfy the query using the following guidelines:

- If the best node to answer the query is the local node, the collaboration agent requests the value from the local sensor. It also creates an artificial response hat it passes to the neighborhood agent to insert in the outgoing beacon for the next Λ time. This proactively shares sensed context.

- If the best node to answer the query is neighbor d_j, the collaboration agent creates places a sensing request, r in the outgoing beacon for Λ time. When d_j receives r, it responds as described above. All other neighbors that receive r use the contained information to update C.

Algorithm 1: Sensing candidate selection algorithm

1 **Function** SELECTCANDIDATE: *query q(t), α*
2 $s_k \leftarrow q(t)$
3 $L_j(t) \leftarrow l_j(t') \cdot e^{-\Lambda(t-t')}$ for $j = 1 \ldots n$
4 construct $w \in \mathbb{R}^n$ s.t. $w_k \leftarrow \frac{\sum_{j=0}^{n} \neg C_{jk} \cdot L_j(t)}{n}$, $k = 1 \ldots m$
5 construct $D \in \mathbb{R}^{m \times m}$ s.t. $D_{kk} = w_k$, for $k = 1 \ldots m$
6 $H \leftarrow CD \in \mathbb{R}^{n \times m}$
7 $\tilde{H} \leftarrow$ row normalize H //each $\tilde{H}_{j,:}$ is a probability vector
8 $y = f_\alpha(\tilde{H}_{:,k}, E)$ //affine combination
9 return $\exists d_j . y_j > 0 \wedge y_j = min(y)$
10 **end**

Our algorithm relies on the neighbor cache (i.e., $C^{n \times m}$, E^n, and $L^{n \times 2}$), the query, and a parameter $\alpha \in [0, 1]$ that represents local dynamics (i.e., the rate of change in links to neighbors). Algorithm 1 attempts to maximize the Fulfillment Ratio (FR) while evenly distributing energy costs (i.e., minimizing the UEC metric). Line 2 pulls the desired context type from the query. Line 3 computes the time-decayed stability for each neighbor using Equation 3.7. Next, the algorithm computes the *rarity* of the each type. Line 4 computes a vector, w, in which w_k combines the existence of a sensing capability on a neighbor with the link stability, then divides this by the total number of neighbors. Lines 5 through 7 transform the sensor availability matrix $C^{n \times m}$ into a local view of sensing preferences $\tilde{H}$, in which $\tilde{H}_{jk}$ indicates how suitable d_j is for sensing s_k, from d_i's perspective. The intuition is to avoid assigning a widely available sensing task s_k to a node that is capable of sensing some other rarer context type. The set of $\langle \tilde{H}_{:,k}, E \rangle$ gives a partially ordered list of preferences for the capable sensing nodes. Our algorithm then applies an

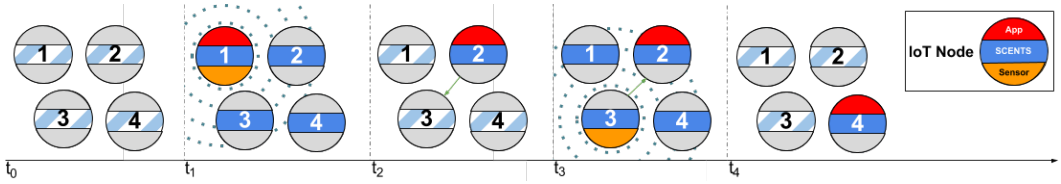


Figure 3.2: Collaborative sensing in SCENTS.

affine mapping function (Line 8) to generate a total ordering of these pairs. The optimal candidate is given by the infimum of the set of $\langle \tilde{H}_{:,k}, E \rangle$ pairs (Line 9), which gives the positive minimum in the mapped outcome, if it exists.

3.2.5 An Example of SCENTS Interaction Flow

Figure 3.2 provides an example of sensing task coordination in SCENTS by showing a hypothetical series of interactions among four nodes [2]. Note that nodes are not synchronized in time, so the notations of $t_0 \ldots t_4$ are simply for convenience of explanation. Each node is illustrated with three layers, which represent, from top to bottom, the application layer, the SCENTS layer, and the hardware sensing layer. The status of each layer in each node is denoted by color: inactive is gray; active layers are shown in color; when a node is only participating in neighbor discovery, the SCENTS layer is depicted in blue

[2]Colored areas indicate activation in the given time step. At t_1, node d_1 generates a sensing query that is resolved locally and its result is shared proactively with neighboring nodes. At t_2, the collaboration agent at node d_2 determines that the best way to resolve the application's query is through a sensing request to node d_3. At t_3, node d_3 acquires the data and adds the result to the beacon, which is received by d_2 but also overheard by other nodes. Finally, at time t_4, the collaboration layer at node d_4 determines that an application query can be resolved by cached values.

stripes. The example walks through an application request satisfied by local sensing (at t_1), an application request satisfied by remote sensing (at t_2-t_3) and an application query satisfied by a cached context value (at t_4).

3.3 SCENTS Experimental Evaluation

3.3.1 IoT Device Power Profiling

We first profile the capabilities of an off-the-shelf IoT device: the Nordic Thingy 52 sensor kit [94]. This multi-sensor lightweight device is equipped with BLE and powered by a rechargeable Li-Po battery with a capacity of 1440 mAh. The Thingy allows fine-grained control over BLE beaconing and scanning, which is required by continuous neighbor discovery. We extended the on-board sensors with an external GPS module [2]. We used this setup to create a power consumption model for the Thingy using a profiling program that executes a series of operations (i.e., transmitting beacons, scanning advertising channels, and sampling sensors).

We sampled the current at 10KHz using an Infiniium DSO9404A oscilloscope, then reconstructed the current measurements into pairs of power and device behavior. Table 3.1 shows the measurement results. The first row (*Idle*) is the power consumption when the Thingy is battery-powered but inactive (i.e., radio and all sensors are disabled). We then sample different operations' consumption in isolation.

[2]Volatile organic compounds (VOC) `https://www.epa.gov/indoor-air-quality-iaq/volatile-organic-compounds-impact-indoor-air-quality`

Table 3.1: Power consumption

Operation	Power (mW)
Idle	0.05387
Scanning	16.86997
Beaconing	3.49649
GPS locating	95.34249
Humidity	1.87975
Air pressure/altitude	2.39284
Air quality (VOCs)	0.14427
Motion	6.74112
Color and light intensity	7.55269
Temperature	4.44521

3.3.2 System Implementation

To evaluate SCENTS, we built a street-level simulated world, including heterogeneous and dynamic sensing resources. The core of our simulation is the OMNeT++ v.5.4.1 discrete event simulator [122]. The wireless physical and MAC layers are based on the INET Framework v.4.0 [55]. Geographic coordinates and 3D physical obstacles rely on Open Scene Graph [95] and Open Street Map [96]. Algorithm 1 is implemented using the Eigen3 template library [38].

Simulated IoT devices use the device profile above, including multiple sensors, low-radio duty cycle, and short-range device-to-device communication. The available sensors include all the sensors in the Thingy52, plus a position sensor.

For the link layer, we modified INET's IEEE802.15.4 narrow band bea-

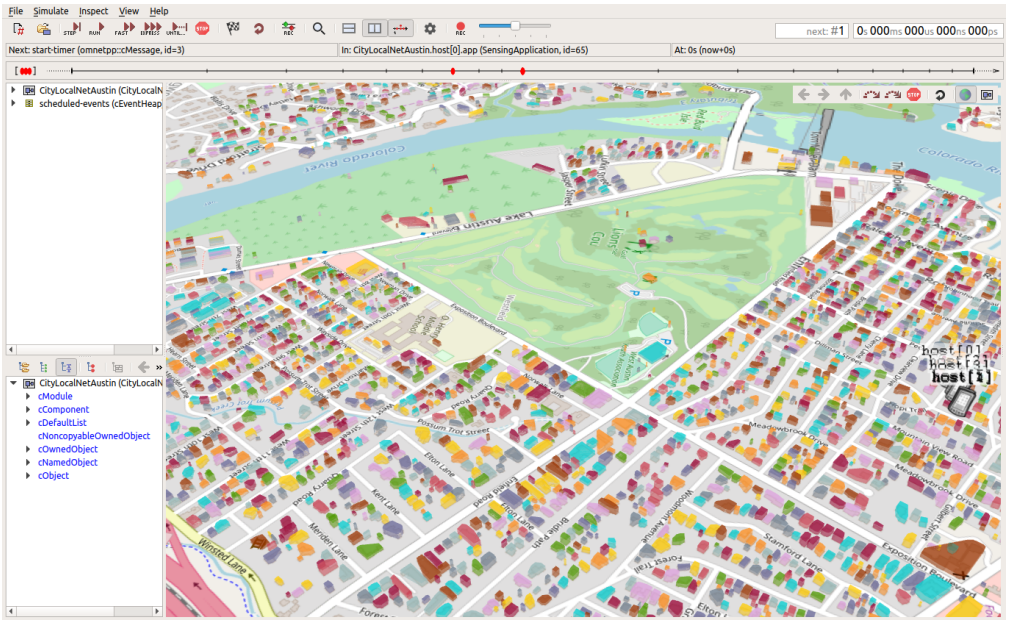


Figure 3.3: A suburban scene in the simulator for SCENTS.

con operation. In particular, the node mimics BLE beaconing by adding a random slack to the beacon intervals. Within this networking framework, we implemented the BLEnd protocol [58], with which our neighborhood agent interfaces to send and receive beacons. Fig. 3.4 shows the structure of a SCENTS beacon, which fits in the 31 byte payload of a BLE advertisement. The first three bytes are used for required identity information for neighbor discovery. Each beacon also contains the node's sensing capabilities (4 bytes), followed by *exchange* segments, which encapsulate sensing requests and responses. Each exchange segment contains the source and destination ids (each 1 byte), the context type (5 bits) and a context value (4 bytes). Some context types have values larger than 4 bytes (e.g., GPS readings); SCENTS supports an extended

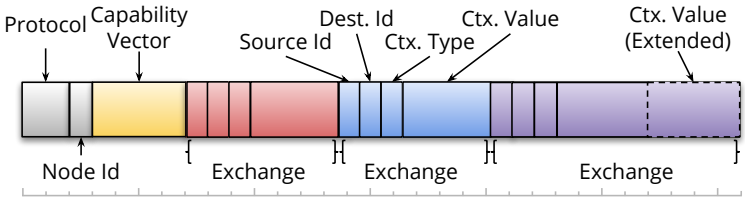


Figure 3.4: Beacon payload structure.

exchange segment of up to 8 bytes.

3.3.3 Scenarios

We use three realistic and parameterizable real-world scenarios for our evaluation[3]:

Fleet: *a set of IoT devices are carried by a group of people (whose size may vary). The group members have nearly identical trajectories.*

Commuters: *a set of devices follow partially overlapped trajectories. A subset move toward the same destination along partially different paths. The remaining share the trajectory at the start then diverge to a different destination.*

Individual: *a user carrying an IoT device moves through a smart space with embedded sensing resources.*

[3]Code repository: `https://github.com/UT-MPC/collab-sensing-simulation`

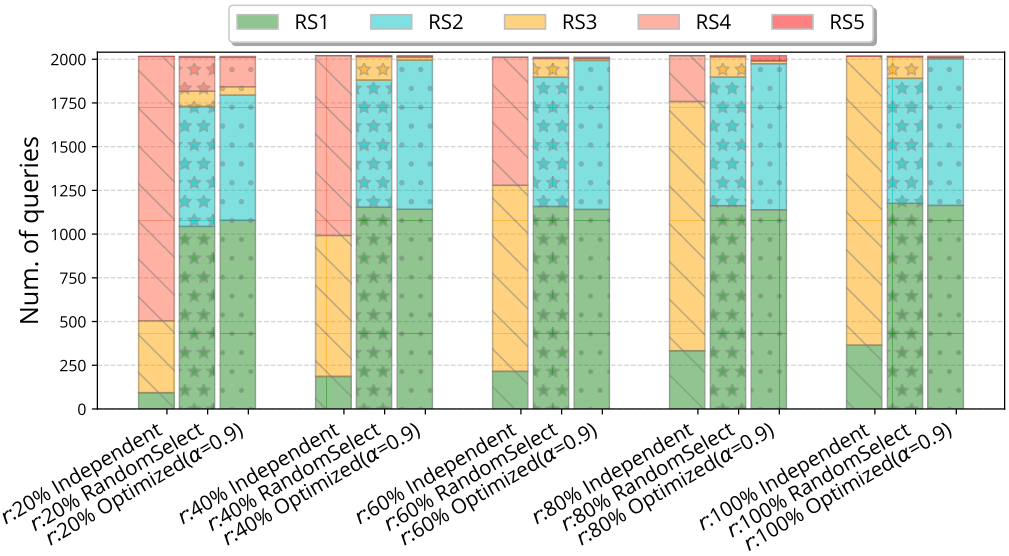


Figure 3.5: Query fulfillment comparison.

3.3.4 Collaboration Analysis

Our first experiments evaluate the effectiveness of collaboratively sharing the sensing task. Specifically: (1) Does collaboration mitigate missing sensing capabilities using proximal sensing resources? (2) How are sensing queries resolved (i.e., individually or collaboratively), and how do distributions of capabilities across devices impact performance?

We compare our algorithm with two naïve strategies. In the independent strategy, each device fends for itself, answering queries only with on-board sensing capabilities. In the random strategy, nodes randomly choose a *capable* neighbor to collaborate. The results we report use six nodes in the Fleet scenario. We vary sensing capabilities by assigning each node a random subset of sensors; the size (r) of the subset varies from 20% of the available types up to

100%. Each node hosts an application that randomly generates a query every 10s. Each simulation lasts for 850s, and we repeat each experiment five times with different random seeds. Each query can result in one of the five states:

- ***RS1*** *(cached reading)*: query fulfilled by a cached value
- ***RS2*** *(answered request)*: query fulfilled by an answered sensing request
- ***RS3*** *(self-sensing)*: query fulfilled by local sensors.
- ***RS4*** *(failure: no response)*: unanswered sensing request
- ***RS5*** *(failure: no capability)*: lack of required sensor in the neighborhood

In Fig. 3.5, each bar is the summed for all runs of each setting. The settings are grouped first by the fraction of capabilities allocated to each node, then by strategy. The differences between the individual and both the random and optimized (i.e., SCENTS) strategies highlight the benefits of collaboration. By leveraging the heterogeneous capabilities in proximity, significantly many more application queries can be satisfied. As r increases, slightly more requests can be satisfied by cached values when using algorithm 1 than random selection, reducing the energy cost of sensing in the local network neighborhood.

To further evaluate SCENTS's performance when devices have heterogeneous capabilities, we use the same scenario but with 10 nodes and a wider

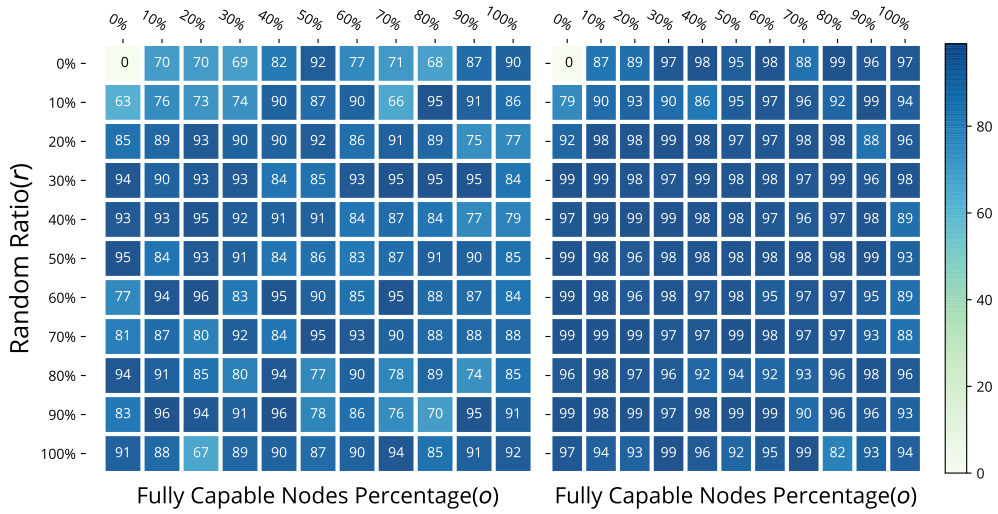


Figure 3.6: Fulfillment ratio with mixed capabilities. Query interval = 5s (left), 10s (right)

range of capability settings. This time we analyze the Fulfillment Ratio metric from Equation 3.1. We generated the configurations by iterating over two parameters. The first, o, captures the fraction of nodes that are fully capable, i.e., can directly sense all context types. The second parameter, r, is the same random ratio as above. We vary the interval of each node generating sensing queries between 5s (the same as Λ) and 10s. These parameters generate 242 different configurations; we repeat each run three times. Fig. 3.6 shows the results.

Except in the extreme case (when $o = 0$ and $r = 0$, i.e., there are no sensing capabilities in the neighborhood), the Fulfillment Ratio climbs quickly as either parameter increases. Over 63% and 79% of queries are satisfied with the help of collaboration (in $o = 10\%$, $r = 0$ and $o = 0$ $r = 10\%$ cases).

3.3.5 Quality-of-Service Analysis

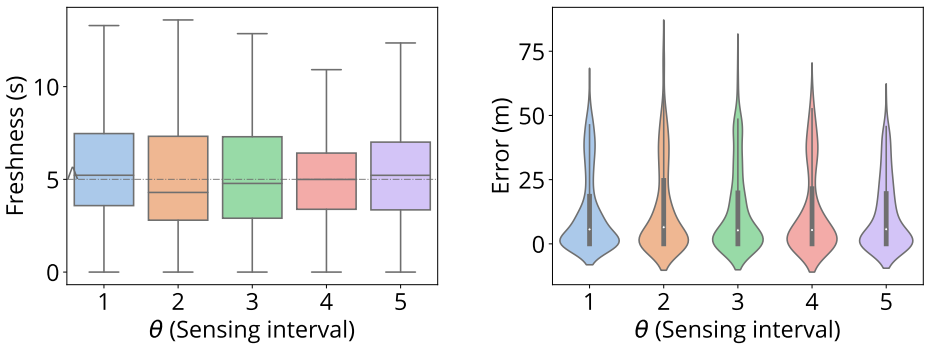
We next assess SCENTS's quality-of-service (QoS) under increased dynamics in the neighborhood's sensing resources: (1) How adaptive is SCENTS to realistic dynamics in a smart city IoT scenario? (2) To what extent is sensing quality traded for increased capability and reduced energy consumption?

We use the Commuters scenario[4] with 10 devices that follow different yet overlapping trajectories; six start and end at the same locations and deviate in the middle; the remaining devices start with the others but head to a different destination.

We assess the faithfulness of SCENTS in sensing a device's position when the simulated devices collaborate. We compute the *freshness* of sensed values and their *error* from the ground truth. We define Freshness (F) to be the *time* that elapses between sensing and delivering a query response and the Error (E) to be the *magnitude* of the difference between the sensed value and the ground truth. We varied the interval between queries using a parameter θ. In particular, node d_i receives one sample for each desired context type within the time window $\theta \times \Lambda$. Smaller values of Λ are associated with higher sampling frequencies and therefore higher total energy costs for sensing. We repeat each experiment five times, resulting in a total of 17729 queries for each setting.

In Fig. 3.7a each box shows quartiles of freshness in seconds, while the

[4] `https://youtu.be/KPqtK9t2efs`



(a) Context freshness **(b)** Context Error

Figure 3.7: Freshness and error analysis (commuter scenario).

whiskers extend to show the distribution. The dashed line marks the maximum discovery latency Λ (5s). Most queries are nearly as fresh as Λ, meaning the parameters of neighbor discovery dominate delays in receiving sensed values. The medians are above Λ but within 15%. While motion related context types (e.g., orientation) may be affected by this freshness, the majority of context types (e.g., humidity) do not vary at the second scale in the nearby physical world.

Next, we evaluate the error caused by sensing delay. Fig. 3.7b depicts the kernel density estimation of the errors (in meters) for varying sensing frequencies. The majority of errors are less than 10 meters. The median errors are all less than 5 meters. SCENTS tends to choose nearby resources to collaborate with, and these resources are likely to have context values close to the ground truth since the context types are spatially correlated. Errors do not increase as query frequency changes.

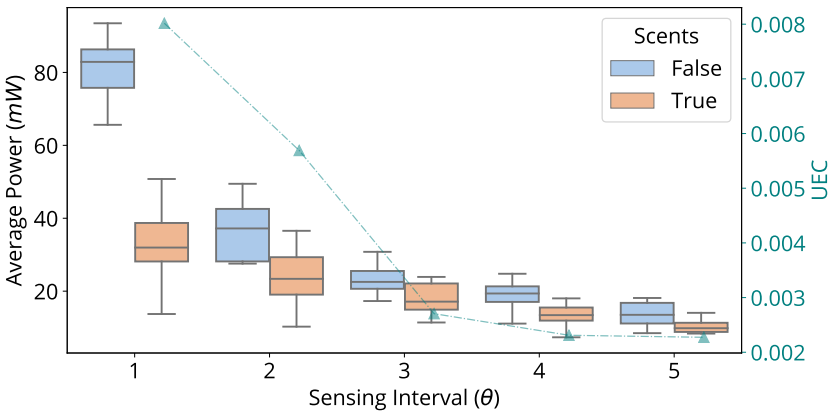


Figure 3.8: Energy cost distribution (E) and UEC.

As our goal is to maximize query fulfillment while evenly distributing energy costs of sensing, we measure the distribution of E values and the UEC from Equation 3.2. Fig. 3.8 shows these values for the Fleet scenario with 10 nodes for both the individual strategy (blue) and SCENTS (orange). We first compute the average power consumption per second for each node, then average over all runs. Not only does collaboration save substantial energy across all values of θ, but the burden of context sensing is more evenly distributed (as indicated by the tighter box and whisker plots for SCENTS). However, there is still some variance, especially at high query frequencies, indicating that refining the way Algorithm 1 considers the energy costs of collaborative sensing is an area for future work.

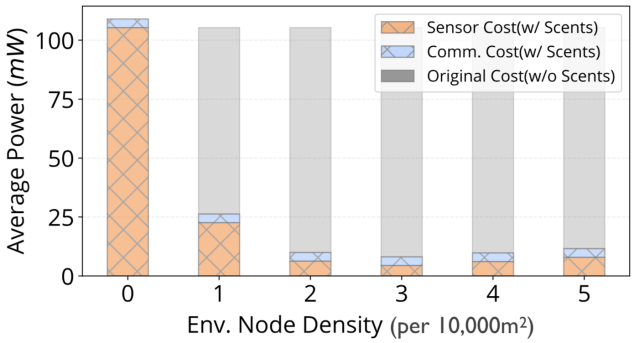


Figure 3.9: Energy cost (individual scenario).

3.3.6 Energy cost Analysis

Our final experiments analyze the energy savings of collaborative sensing. We use the Individual scenario, a smart city with situated environmental sensing beacons. Such devices commonly have a power source that frees them from worry about energy costs. In the experiments, the stationary nodes are placed randomly in the smart city space; we vary the density of stationary nodes as the number of devices per $100 * 100m^2$. The SCENTS layer used on these devices is otherwise the same as described previously. The mobile device is fully capable and hosts an application that queries a randomly selected context type every 10s. We measure the energy savings afforded the mobile device from collaboration with the stationary sensors.

Figure 3.9 shows the energy consumption, comparing a mobile device using only on-board sensors to fulfill queries (the backing gray bar) to the energy consumption when employing SCENTS. In each bar, the orange depicts the energy spent on sensing, while the blue depicts energy spent on commu-

nication. We compute the average power consumption per second for each node, then average over all nodes. Starting from a density of one device, the user node receives a significant benefit from collaborative sensing and fulfills its sensing demands with 70% power savings.

3.4 SCENTS Conclusion

We demonstrated the practical benefits of collaboratively sensing context in highly dynamic IoT environments. SCENTS leverages the context sensing capabilities of opportunistically encountered devices in a non-intrusive manner. Because SCENTS encapsulates access to both local and remote sensors, individual sensors are no longer tied to specific applications or application schedules, allowing much more flexibility in utilizing IoT sensing resources.

Chapter 4

PINCH: Self-Organized Context Neighborhoods for Smart Environments

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 [109] or to situated beacons in the city that give information about interesting sights [47, 84]. 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 [98]. Enabling these applications requires *continuous neighbor discovery*, and protocols for neighbor discovery have received significant attention, from wireless sensor networks to smart-* applications [13, 37, 58, 61, 67, 86, 101, 124, 126].

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* [109,

134]. 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 [62, 92]. Using co-located devices to extend a device's sensing capabilities has also been explored. ChitChat [28] 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 [6, 136]. As described in more detail in Section 4.1, these approaches are all driven ex-

plicitly 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 4.1, 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 4.2 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. The contributions of this chapter are the following:

- 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.

4.1 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.

4.1.1 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 [62, 63, 92, 104, 138]. 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 [31, 44, 56, 129]. However, these systems PINCH 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 [28] 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 [7, 136]. 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.

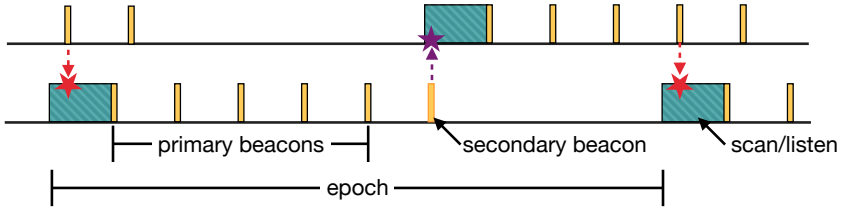


Figure 4.1: BLEnd schedule of two devices. Solid rectangles are scanning (listening) periods; skinny rectangles are beacons. Discovery occurs when one device receives another's beacon. Beacons are scheduled in the second half of an epoch as a result of receiving a *primary beacon*; these *secondary beacons* enable bi-directional discovery.

4.1.2 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. As introduced in section 2.2, BLEnd [58] 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 4.1 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

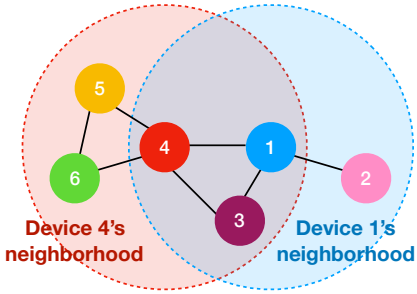


Figure 4.2: Asymmetric neighborhoods defined by BLEnd beacon exchange. Device 1 exchanges beacons with 2, 3, and 4; device 4 exchanges beacons with 1, 5, and 6.

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 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 4.2 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.

4.2 The PINCH Approach

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 4.3 shows the overall operation of PINCH.

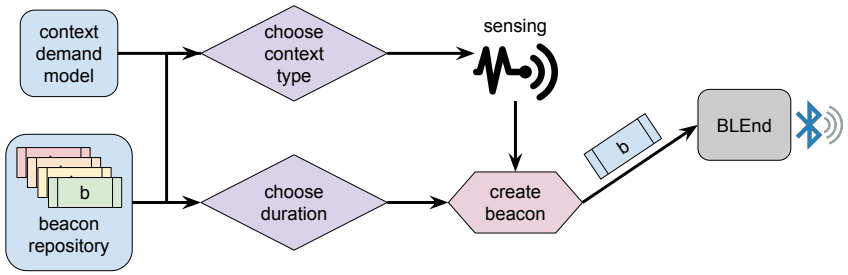


Figure 4.3: Overall PINCH operation

We assume that the set of 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.

A device stores the most recently received beacons in a *beacon repository*, removing them when devices move out of range. Information from re-

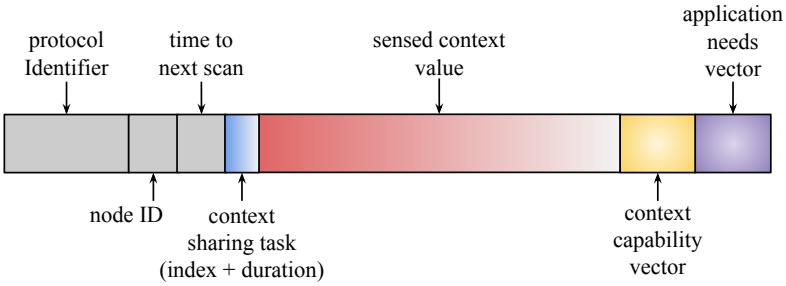


Figure 4.4: PINCH beacon payload

ceived 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.

4.2.1 Beacons for Self-Organized Context Sharing

In the BLEnd beacon of Figure 4.4, 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 4.3), 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 4.2.2. The remaining payload (the red portion in Figure 4.4) 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.

4.2.2 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. 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.: $\sum_{c=1}^{k} w_c = 1$.

4.2.2.1 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.

4.2.2.2 A Dynamic Context Demand Model

Including the application needs vector in the beacon enables PINCH's context demand model to respond to changing context demands arising from mobility or from changing application conditions. 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:

$$w_c = \frac{\eta_c}{\sum_{i=1}^{k} \eta_i} \tag{4.1}$$

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 4.4) as

$b.a$. Specifically, η_c is:

$$\eta_c = \sum_{j=1}^{n} |j.b.a \ \& \ 2^c| \tag{4.2}$$

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. 4.1 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. 4.1 to generate normalized context weights w_c.

4.2.2.3 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.

4.2.3 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.

4.2.3.1 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:

$$\mathcal{T} = \bigvee_{j=1}^{n} 2^{j.b.type} \tag{4.3}$$

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:

$$\mathcal{S} = \neg\mathcal{T} \,\&\, cap \tag{4.4}$$

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:

$$P(s) = \begin{cases} 1, & \text{if } \langle \max c : c \in \mathcal{S} :: w_c \rangle = w_s \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

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

[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., $c \in \mathcal{S} \leftrightarrow (\mathcal{S} \,\&\, 2^c = 2^c)$.

4.2.3.2 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 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:

$$P(s) = \begin{cases} \frac{w_s}{\sum_{c \in \mathcal{S}} w_c}, & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

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:

$$P(s) = \begin{cases} \frac{w_s}{\sum_{c \in cap} w_c}, & \text{if } s \in cap \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

4.2.3.3 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:

$$prev(c) = \frac{\sum_{j=1}^{n} |(j.b.cap \,\&\, 2^c)|}{n} \tag{4.8}$$

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:

$$P(s) = \begin{cases} \frac{1-prev(s)}{\sum_{c \in \mathcal{S}}(1-prev(c))}, & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

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

$$P(s) = \begin{cases} (1-\alpha)\left(\frac{w_s}{\sum_{c \in \mathcal{S}} w_c}\right) + \alpha\left(\frac{1-prev(s)}{\sum_{c \in \mathcal{S}}(1-prev(c))}\right), & \\ & \text{if } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

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 $P(s)$.

4.2.4 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 the 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:

$$\mathcal{C} = \bigvee_{j=1}^{n} j.b.cap \tag{4.11}$$

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 $\mathcal{C}(t) \oplus \mathcal{C}(t-\Lambda)$:

$$distance = |(\mathcal{C}(t) \oplus \mathcal{C}(t-\Lambda)| \tag{4.12}$$

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:

$$m = \begin{cases} (2^d - 1), & \text{if } distance = 0 \\ \left\lfloor (2^d - 1) \times \left(1 - \frac{distance}{k}\right) \right\rfloor + 1, & \text{otherwise} \end{cases} \tag{4.13}$$

4.2.5 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:

$$\mathcal{T}_{coll} = \sum_{j=1}^{n} 2^{j.b.type} \tag{4.14}$$

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 $\mathcal{T}_{coll}$:

$$p'_c = 1 - (1 - p_d)^{\mathcal{T}_{coll}[c]} \tag{4.15}$$

where p_d is the BLEnd neighbor discovery probability.

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

$$\mathcal{T}^*_{coll}[c] = \begin{cases} 1, & \text{if } p'_c < p_c \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

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):

$$\mathcal{S}_{coll} = \neg\mathcal{T}^*_{coll} \,\&\, cap \tag{4.17}$$

and used in the algorithms in Section 4.2.3 in place of $\mathcal{S}$.

4.3 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 [120]. BLEnd [58] 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.

4.4 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 described in Section 4.1 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 technology 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

[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/`

204ms (Figure 4.1).

4.4.1 Benchmarking the Algorithms

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

$$\text{coverage percentage(c)} = \frac{\text{\# covered demands for } c}{\text{total \# demands for } c} \tag{4.18}$$

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*:

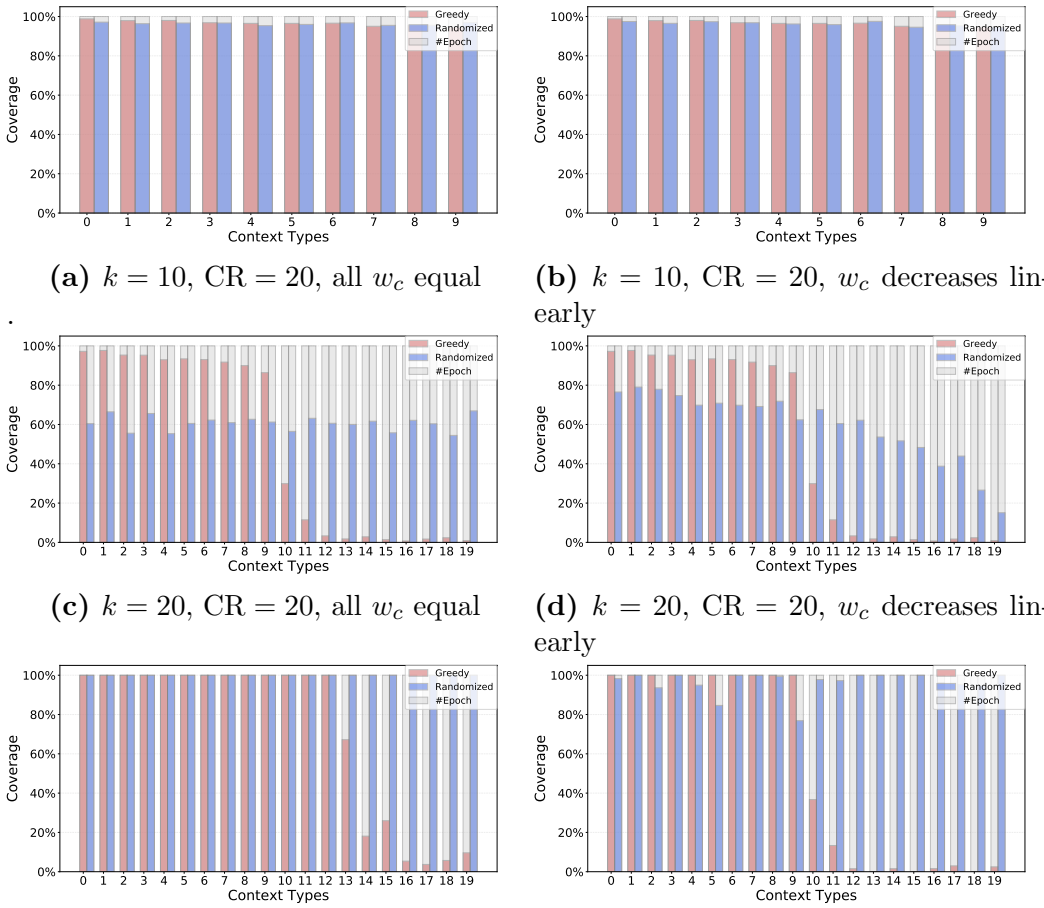
$$\text{coverage quality} = \sum_{c=1}^{k} \left[\left(\frac{\text{\# covered demands for } c}{\text{total \# demands for } c} \right) w_c \right] \tag{4.19}$$

We define a 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 [109], school children, family members, or friends moving together.

4.4.1.1 Context Sharing for Improved Context Coverage

Figure 4.5 shows the coverage percentage for each context type with various parameter settings. Figures 4.5(a) and (b) use relatively few possible

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



(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 4.5: Benchmarking coverage of greedy and randomized selection.

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 the greedy (Eq. 4.5) and randomized selection algorithms (Eq. 4.6). Figure 4.5(a) employs the static context demand model in which all context types are equally important (Eq. 4.1); Figure 4.5(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 4.5(c) and (d), we increase k to 20. In Figure 4.5(c), all context types are equally important. The greedy algorithm favors the lower indexed types (it chooses greedily), while the randomized algorithm spreads coverage more evenly. In Figure 4.5(d), where we employ the linearly decaying demand model, the model itself is apparent in the trend for the randomized algorithm, whose selection of types to share follows the weights in the model.

In Figures 4.5(e) and (f), we employ a dynamic context demand model. In Figure 4.5(e), every device selects a random subset of 20% of the context types it needs (i.e., DR = 20); in Figure 4.5(f), we increase this demand to 60% of the types. In both cases, CR = 20. In Figure 4.5(e), the simple greedy algorithm is unable to adjust when the capabilities are constrained; this is further highlighted in Figure 4.5(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.

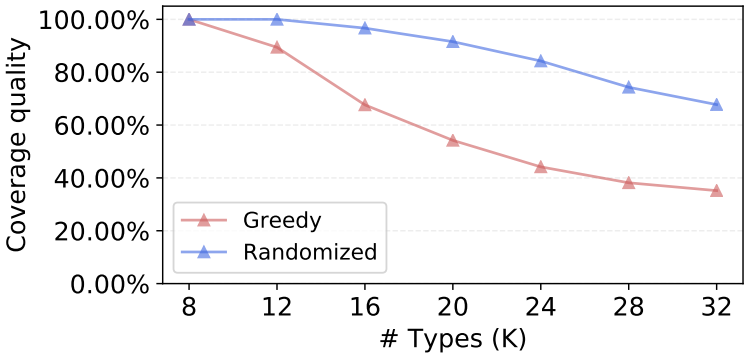


Figure 4.6: The pressure of increasing numbers of context types (CR = 20, DR = 60).

4.4.1.2 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 many high-level context abstractions. Using the same set of ten devices moving together and the same demand and capability models as in Figure 4.5(f), we varied the number of context types (k) from 8 to 32. Figure 4.6 shows the average *coverage quality* for the greedy and randomized algorithms. PINCH maintains relatively 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.

4.4.1.3 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

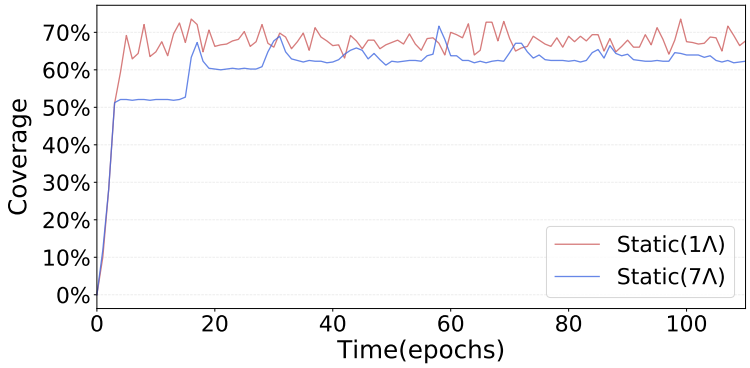


Figure 4.7: Adapting to changing context demands ($k = 16$, CR $= 20$, DR $= 20$).

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 4.7 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]

[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

4.4.1.4 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. To that end, 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 4.8 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 4.9(b) shows the cost to devices as their demands for context types increase from 10% to 100%. When a device is capable of sensing all of 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.

---

that BLEnd can be manipulated to achieve even higher levels of context coverage. This investigation is left for future work.

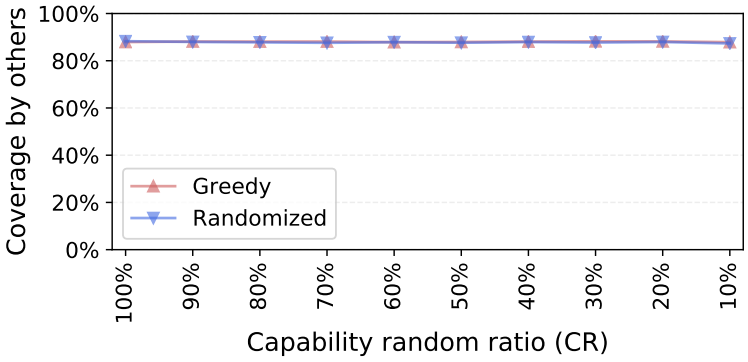


Figure 4.8: Benefits of PINCH ($k = 10$, DR $= 100$).

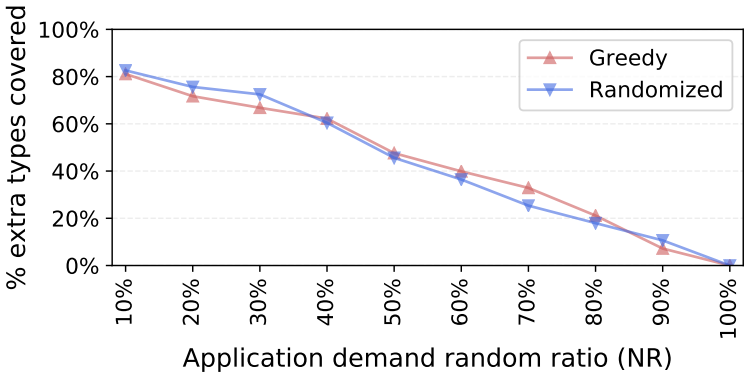


Figure 4.9: Costs of PINCH ($k = 10$, CR $= 100$),

4.4.1.5 Considering the Rarity of Context Types

The rarity-weighted algorithm in Eq. 4.10 allows a device to favor selecting a context type that it alone in the neighborhood 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 4.10 shows the coverage percentage of the rare

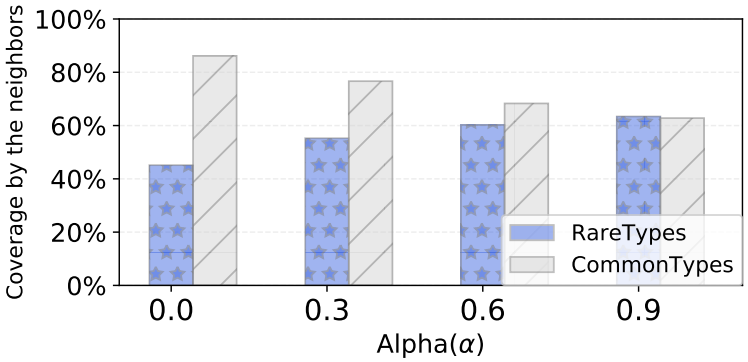


Figure 4.10: Rarity weighting in PINCH.

(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.

4.4.1.6 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 4.2.5 we use the same settings as those in Figure 4.5(d) but with increased capabilities (CR=60) and an increased num-

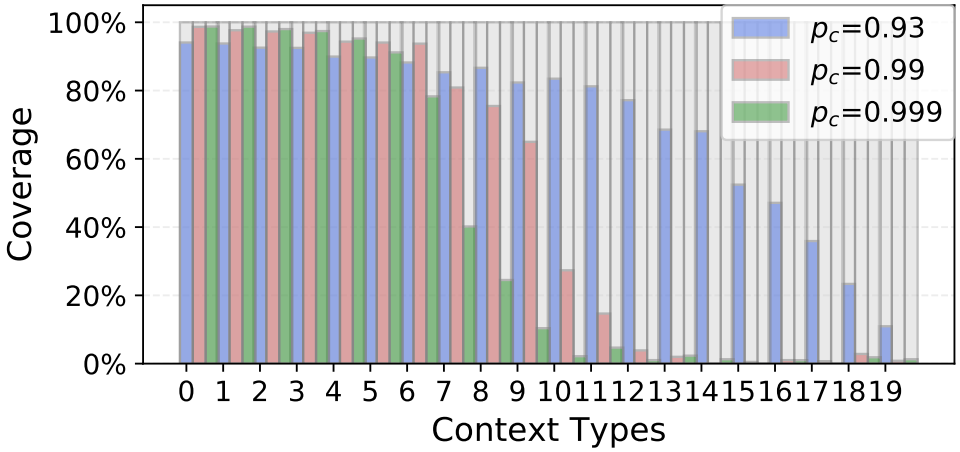


Figure 4.11: Collision awareness (CR = 60, $n = 20$).

ber 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 4.11 shows the results. As the figure shows, 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.

[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 4.12: Snapshot of the central meeting point scenario based in Trento, Italy.

4.4.2 Realistic Smart-City Scenarios

The remainder of our evaluation explores more dynamic scenarios, both in terms of scenario and device capabilities.

4.4.2.1 A Central Meeting Point

We defined a central point in the city as a meeting place; devices travel towards this meeting point, remain for a period of time, then move away before repeating the process. Figure 4.13 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.

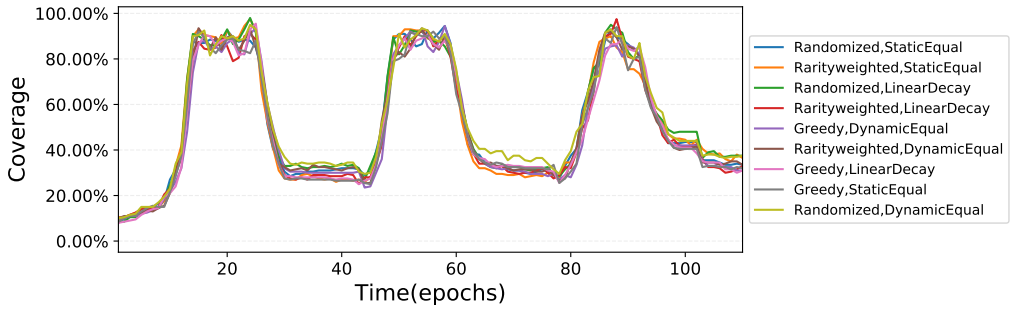


Figure 4.13: Trace of coverage percentage for congregating devices (CR = 20, DR = 100, $k = 10$).

4.4.2.2 Leveraging Situated Beacons

PINCH also has potential benefit in enabling 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 embedded 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 4.14 shows, the device is able to achieve some coverage of context as it moves. Figure 4.14 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.

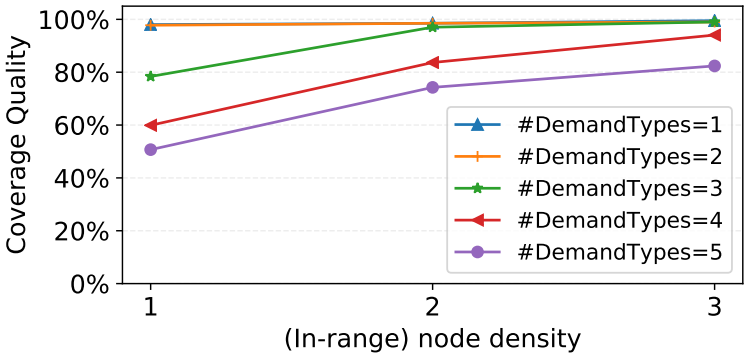


Figure 4.14: PINCH with situated beacons.

4.5 Conclusions

This chapter introduces PINCH, a system that embodies a suite of heuristics to enable devices to opportunistically share sensed context information 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 on PINCH 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 (e.g., automatic device selection[52]). 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.

Chapter 5

Facilitate IoT Sensing Collaboration in Deployments

In this chapter we focus on the real-world implications of facilitating sensing collaboration in proximity networks. We start with introducing a fully automated context-sharing system Stacon in section 5.1. In section 5.2, we assess the feasibility, cost-effectivenes, and privacy factors of real-world sensing collaboration system. To showcase the potential of the proposed systems throughout this thesis, we deploy a context-sensing system in an academic building and collect an extensive dataset with real traces and people's both implicit and explicit interactions with the smart environment to encourage further studies. In the last section of this chapter, we verify the performance of the SCENTS collaboration framework (section 3) using the collected dataset.

5.1 Stacon: Context Neighborhood Automation in the Internet-of-Things

5.1.1 Motivation: Self-stabilizing Neighborhood Dynamics

In the past decade, sensor-equipped systems have been pervasively deployed in our world. These low-powered devices *sense* physical attributes of their surrounding environments and provide digital services accordingly as a

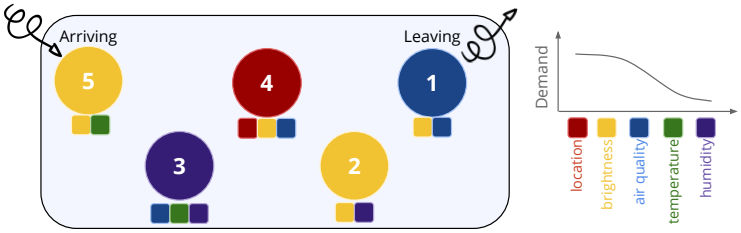


Figure 5.1: Dynamics in a context neighborhood.

form of *calm computing* [127]. As previously discussed in section 3.2 and 4.1, there is an emerging sub-field that leverages coordination among *smart agents* hosted on co-located devices and explores the potential of building more flexible and sustainable systems using multi-agent collaboration.

As the beginning section of this chapter, we evaluate the real-world implications of sensing collaboration in IoT proximity networks. We start from a context neighborhood scenario similar to the PINCH storyline, focus on the network *dynamics* in real-life environments and derive a generic framework to enable fully-automated sensing collaboration on off-the-shelf SoCs.

We introduce Stacon, a framework for low-powered mobile IoT devices to share sensing capabilities. Stacon enables devices to automatically detect other nearby sensing resources and to self-select sensing tasks based on the device's own needs, the capabilities of the environment, and the needs of other nearby devices. The goal is to maximize the context fulfillment of the neighborhood as a whole. Here we use the term context neighborhood to refer a set of participating devices that are mutually reachable in one-hop network.

Fig. 5.1 shows a schematic of the Stacon framework. The figure shows

five available sensing devices (numbered one through five), each with heterogeneous sensing capabilities (depicted as colored squares underneath each numbered device). The neighborhood also has a *context demand model* (section 4.2.2), which characterizes the needs of the neighborhood for the various sensable attributes (the location attribute is the most important to devices in the neighborhood, while humidity is least important). The goal of Stacon is to enable each device to make a local decision about which attribute to sense (these choices are depicted as the colors of devices in the figure) based on information other devices share about their needs, capabilities, and selected tasks. Achieving a self-stabilizing behavior in this environment is plagued by challenges that are largely tied to network dynamics. When devices enter and leave the neighborhood, the neighborhood's needed context types, sensing capabilities, and selected tasks all change.

We build the Stacon middleware with the complete software stack on Nordic Thingy sensor kits[94]. The underlying wireless communication substrate is built on top of BLEnd from chapter 2. In Stacon, the neighbor discovery beacons carry descriptions about capabilities, needs, and tasks to aid in the self-stabilizing algorithm (Section 5.1.2). The middleware supports performing sensing tasks on the SoC mentioned before with the extension of customized development boards (Section 5.1.3). Besides the Stacon embedded systems, we also include an Android tablet scanner that sniffs the BLE beacons nearby to monitor the context neighborhood, to visualize the heterogeneous sensing configurations of each IoT device, and to analyze the sharing

strategies in comparison to global optimal in real-time.

5.1.2 Self-stabilization in Distributed Stacon

We next formalize a self-stabilizing context neighborhood. We then describe Stacon in detail.

PROBLEM (SELF-STABILIZATION OF CONTEXT NEIGHBORHOOD): *A network of sensing devices $D = \{d_1, \ldots, d_n\}$ collaboratively construct a context neighborhood via device-to-device communication. $C = \{c_1, \ldots, c_m\}$ context attributes can potentially be sensed, and each device d_i can sense $C_i \subseteq C$ depending on its on-board sensors. C is known* a priori*, and every c_j has an associated weight w_j that indicates its demand in the neighborhood. We define a device-context pair (i, j) to mean that d_i is sensing and sharing c_j. At time t, an assignment S_t is a set of pairs (i, j) such that $c_j \in C_i$ for all $(i, j) \in S_t$, and, for each d_i, there is at most one pair $(i, j) \in S_t$. The goal of Stacon is to stabilize the network neighborhood such that: (1) the neighborhood's context demands are fulfilled (i.e., $\max_{S_t} \sum_{j:\exists i,(i,j)\in S_t} w_j$), and (2) the self-stabilization process finishes within the discovery latency (Λ) with probability p_d.*

System overview. BLE requires fixed-length beacons, but continuous discovery information does not use the entire beacon payload. We therefore encode Stacon-specific information into the unused beacon space, as described in Section 5.1.3. As shown in Fig. 5.2, Stacon receives packets from the scan reports of the underlying BLEnd protocol. Every beacon b received during the BLEnd scan period will be stored into a beacon repository $\mathcal{B}$. At the end

of each scan period, Stacon removes outdated beacons from $\mathcal{B}$ and computes the current sensing capabilities and demands of the neighborhood. Using this update, Stacon then *re-selects* a context type as its new task, queries the corresponding sensor if necessary, and encodes the new packet into the neighbor discovery beacon payload.

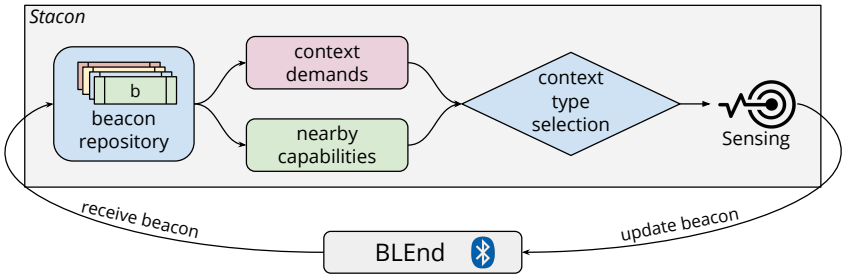


Figure 5.2: Stacon overview.

Self-stabilization. The above self-stabilization problem can be mapped to a classic bipartite matching where the device is one party and the context sensing task is the other. Like bipartite matching, each device can select only one context type to sense in a round, and each context should only be sensed and shared by one device since repeated sensing in a neighborhood is not optimal; instead a device should conserve energy. In common IoT settings, a practical number of devices for slot-less symmetric discovery is typically under 50 [58, 101, 135]. For a battery-powered IoT device with limited computation resources (e.g., a Cortex M4 based system-on-chip runs at 64MHz), the worst case time complexity of a $O(n^3)$ algorithm for bipartite matching is acceptable.

Stacon's self-stabilization algorithm is shown in Algorithm 2. Each device calls the AfterScan function every epoch. The device uses information

received in beacons to maintain a capability list for the other devices in the neighborhood; this list uses hashes to succinctly describe other devices' sensing capabilities. During self-stabilization, each device updates the list using the beacons it receives in this epoch. If there are changes in the received beacons relative to the previous epoch, i.e., a device has left, arrived, or changed its capabilities or task, the sensing capability of the neighborhood has changed and Stacon runs a matching algorithm similar to the *Hungarian* algorithm to solve the bipartite matching problem and get the optimal assignment $S_{optimal}$. Each device d_i chooses their context c_j as in the optimal assignment, i.e., $(i, j) \in S_{optimal}$. As we have mapped the problem to bipartite matching, the $\max_{S_t} \sum_{j:\exists i,(i,j)\in S_t} w_j$ of a fully-connected neighborhood is guaranteed by the *Hungarian* algorithm in a distributed fashion. Since BLEnd guarantees discovery within a target latency (Λ) and target probability (p_d), Stacon will stabilize with the same probability and the same latency, satisfying the second condition in the problem formulation.

Algorithm 2: Self-stabilization Algorithm

```
1 Function AfterScan: device d_i, B, snap_prev
2     Compute capability list C_j s.t. j ≠ i from B.
3     Take a snapshot of local < C_1, ··· , C_n, D > as snap_new.
4     if snap_new ≠ snap_prev then
5         Compute S_optimal using matching algorithm.
6         Select c_j as its type of context where (i, j) ∈ S_optimal.
7     snap_prev ← snap_new
8 end
```

5.1.3 System Implementation

Stacon Beacon Payload

We implemented Stacon on top of the BLEnd scheduler[1]. BLEnd uses BLE to exchange wireless packets on 2.4GHz channels. The user beacon payload is 31 bytes; BLEnd uses only 5 bytes for neighbor discovery purposes. Therefore, we put information to support Stacon's self-stabilization into the 26 unused bytes. We use one byte to identify the Stacon and one byte for the local id. Following that are two bit vectors of two bytes each, encoded from the sensors of the device and the context demand of its upper-layer applications. The shared sensor reading is placed in the next five to nine bytes. The first byte is used for a header and is followed by four bytes of the actual sensor reading. While four bytes are generally sufficient for most types of context, sometimes we need additional space to contain richer values (e.g., location); for those cases we allocate four more bytes as optional content. Fig. 5.3 shows the complete 15 bytes used in Stacon and its placement in the BLEnd beacon payload.

Network Dynamics Emulation

The prototype Stacon framework is integreated with the Nordic nRF52 series platform including Thingy52 [94] and nRF52840 development kits.The device has 8 on-board sensors and is able to provide even more types of fine-grained context information about the environment (e.g., air quality, ambient

[1]Code repository `https://github.com/UT-MPC/BLEnd_Nordic`.

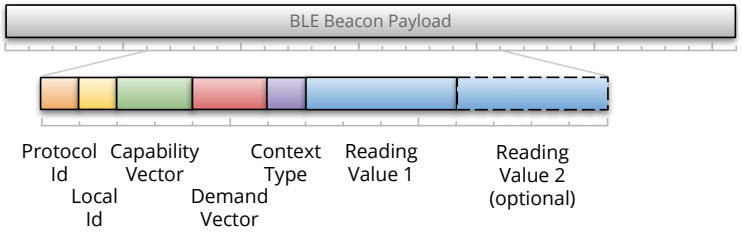


Figure 5.3: Stacon payload structure.

noise level). Thingy52 is equipped with BLE for wireless connectivity and a rechargeable lithium polymer battery of 1440 mAh (nominal capacity) The nRF52840 DK is capable of wide range of sensor extension. In our prototype system we use it to provide location information using an Adafruit GPS module [2]. In addition we provided an Android application to visualize the wireless beacons that are exchanged and the *context fulfillment* of the network neighborhood compared to the optimal task allocation *on-the-fly*.

To simulate heterogeneous networks, each IoT sensor kit has only a random subset of sensors enabled. Context types are assigned with distinguishable RGB color values and we use the on-board LED lightwells to visualize the sensing task a device is currently executing. Together with user-triggered device *arrival* and *departure* events (triggered by power cycling the devices), this lightwell representation essentially shows Stacon's stablizing process (shown in fig 5.4).

The instantaneous system status and estimated energy cost(based on

[2] `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf`

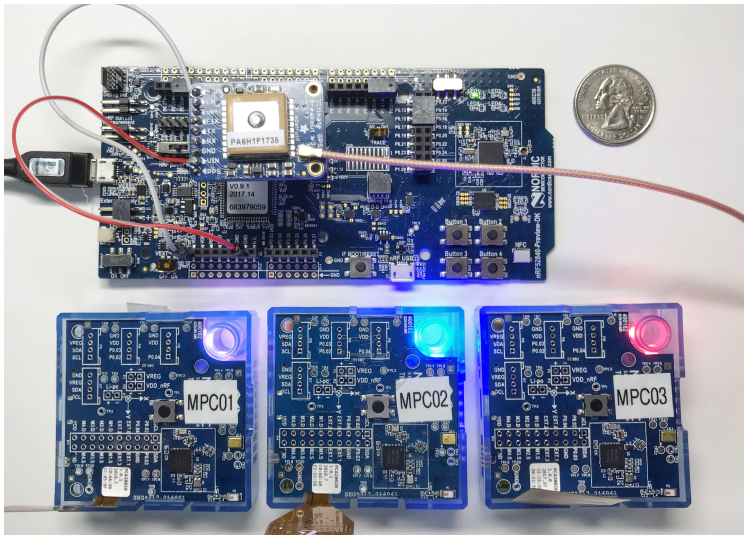


Figure 5.4: Prototype of Stacon.

section 3.3.1) is monitored on the Android tablet application [3] as shown in fig 5.5. Based on the Stacon beacons received from the nearby hosts, the middle panel of the monitor displays the sensing tasks that each host is currently performing with the result sensor reading, sensing capability, context demands and other basic device information. The bottom region of the monitor shows the neighborhood from a "centralized" perspective. From left to right, each column details the most recent capability snapshot of the neighborhood, context demand model, and the task assignments from centralized computation. One can easily compare and identify the discrepancy between the actual sensing tasks of decentralized IoT nodes and the ideal assignment. We tested the system in a radio-noisy environment[4] and verified Stacon's effectiveness in

[3]Code repository: `https://github.com/UT-MPC/BeaconObserverAndroid`
[4]A conference room with hundreds of mobile devices sharing the bandwidth.

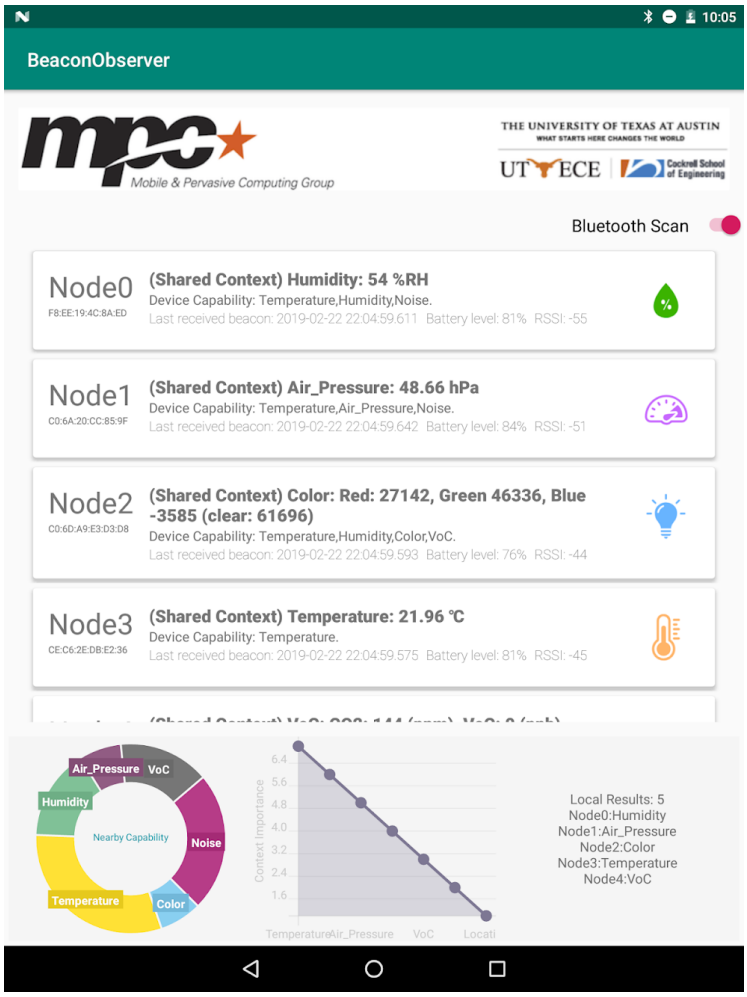


Figure 5.5: Monitoring a Stacon neighborhood.

adapting neighborhood dynamics towards ideal sensing task assignments with expected latency (i.e., Λ).

5.2 Smart-building Deployment and User-side Data Acquisition

We envision a world where people-centric sensing and personalized services can be achieved without centralized data collection and processing or the reliance on video-based surveillance. To help assess the feasibility of building context-aware applications while allowing users to fully control their potentially sensitive data, in this section we study the feasibility of people-centric sensing using only user-side data acquisition. Using off-the-shelf, energy restricted sensor kits in a smart environment together with an energy-efficient message exchange scheme implemented on top of BLE, the collected dataset provides insight on a continuous cyber-physical view from users' individual perspectives. The availability of the dataset encourages further studies of users' activities, for instance to perform distributed inference on users' social interactions and activity trajectories.

5.2.1 Support Opportunistic Collaboration in People-centric Sensing

In people-centric sensing [22], individuals utilize sensors to learn and share information about themselves and their environments to enable personalized digital services or contribute to a social good. With technological advances in embedded systems and wireless communication, today's low power IoT devices enable numerous, formerly impossible opportunities for sensing and sharing information about the ambient environment. In this work, we are interested to how user-side data acquisition can stimulate the fusion of

personal, public, and social sensing in a privacy-preserving way.

Wireless fingerprinting, in particular via Bluetooth technology, is known to be useful to provide distance estimations in indoor localization [42, 83]. Recent years have also seen an increasing interest in leveraging Bluetooth beacons as a tool for device interaction in many application-specific contexts, including tourism [84, 109], device-to-device collaboration [75, 76, 80], social interactions [17], security [89], smart buildings [106], and smart cities [3]. However, most publicly available datasets are collected from service-side devices (e.g., devices connected on unmetered networks). There is a lack of datasets directly acquired from the user side with context information that can correlate to device-to-device or human-to-device interactions.

In this section we focus on the idea of people-centric sensing and provide a dataset containing *continuous* views of users' digital surroundings with rich contextual information. We introduce the setup of our data acquisition in Section 5.2.2. Section 5.2.3 provides details of the presented dataset. Lastly we discuss the potential usage of this dataset in Section 5.2.4.

5.2.2 Hardware Setup and Continuous Collection

To keep our study focused on data collection and make it easy to repeat in other environments, our hardware setup uses off-the-shelf Nordic Thingy52 IoT sensor kits as previously described in section 3.3.1. Once again, BLEnd [58] is implemented on top of the BLE stack as the communication substrate. In the deployment, the devices continuously transmit beacons that

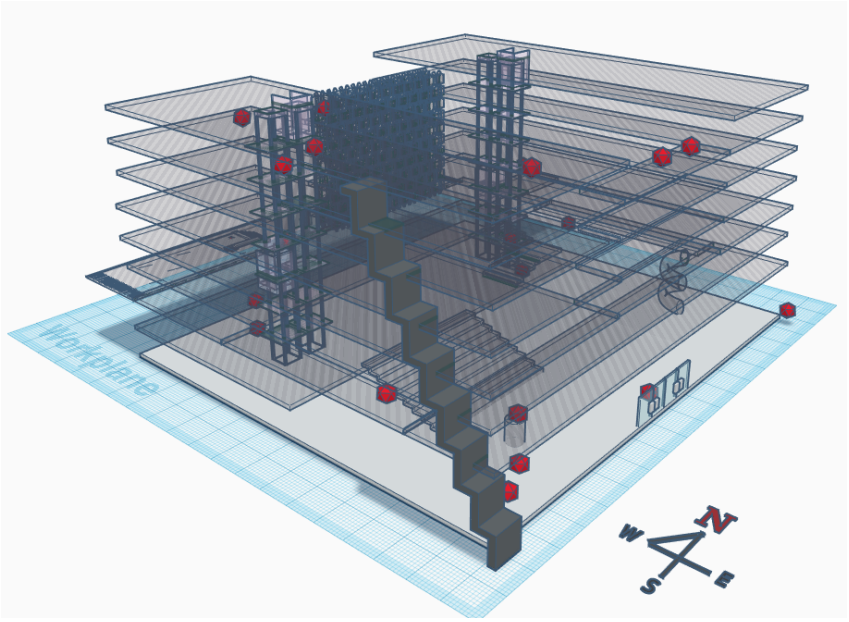


Figure 5.6: Static beacon deployment.

can be received by other devices in range. In particular, we parameterize the BLEnd protocol on each device to achieve a 95% guarantee that each neighboring device receives that device's beacon every four seconds.

In our deployment, we programmed and deployed 55 IoT sensor kits during the 6-week data collection period. Of these kits, 48 were placed as anchor nodes at 24 locations inside a university academic building[5]. These stationary beacons (illustrated in Figure 5.6) were programmed as pairs; at any time, one device in a pair was deployed at the anchor location, while the other was pulled for battery charging. The remaining 7 sensor kits were carried by the human participants in the data collection. All devices beaconed

[5]`http://www.ece.utexas.edu/about/facilities/eerc`

as described above. Each human participant also carried an Android device that also collected any overheard beacons and stored the data from all received beacons into a local SQLite database. Finally, the deployment also included three Android tablets installed at three separate building locations (the two doors to an interior space and an open cubicle work area). The deployment used these tablets to collect participants' explicit "check-ins," which served as ground truth participant location information.

5.2.3 Data Description

We collected the beacon data from the local databases on the participants' Android devices into a single merged database. Within this database, each "row" contains information about one received beacon, as shown in Table A1 (in appendix). These are indexed by the identifier of the participant device that received the beacon (i.e., *HostId*, a value in the range [0..6]). Each row contains information about the received beacon: a description of the received beacon including the timestamp, the received signal strength indicator (RSSI), the sender's Bluetooth address, and the receiver's id. The data also includes the summary of the sensor information read from the sender's on-board sensors, including sound level indicators (peak and average), temperature, air quality measurements (eCO^2 particles per million and volatile organic compound particles per billion), air pressure, and humidity.

During the data collection period, it was possible for a device to receive more than one beacon from a given neighboring device within a short time

period. Because the phenomena being sensed were unlikely to change at a very high frequency, we down sampled the beacons received to one second (i.e., each device stores at most one beacon from any other device each second). Our participants also carried the Android devices for 24 hours; we pruned any data collected off-campus for privacy reasons. The main table in our final dataset contains 20,612,286 entries. As described above, we used tablet check-in locations to collect additional ground truth labels for the data set; we stored these events in a second table as a time series of *ParticipantId*, check-in location, and timestamp.

5.2.4 Uses and Conclusion

The data set has a variety of potential use cases.

Composition: This dataset leverages user-side acquired beacons that carry a rich set of ambient context information that can be useful, for example, for mining the correlation between context snapshots and human activity recognition and prediction [34, 92]. Context snapshots can be derived from the beacon entries in the main *beacon table* at any aggregation level as needed. Inter-human encounters can be inferred from the senders' Bluetooth addresses, which are cross-referenced in the *device description table*. While the relative location of the hosts could be estimated using the *RSSI* of the beacons sent from the stationary nodes (category provided as part of device description), the *check-in table* offers explicit human-to-machine interactions.

Beacon Carried Context: The sensed attributes contained in each beacon are sampled with the same interval from the device (identified by its Bluetooth address). Note that time consumption for sampling can vary between different types of sensors. Most sensors can return the readings back to the controller with negligible delay. The air quality sensor and microphone, however, need more sampling time. The accuracy of the embedded sensors are referred to the hardware specification in [94]. Depending on the use case, certain pre-processing or noise reduction should be applied when extracting higher-level contexts from the raw data. For instance, characteristics of *RSSI* (e.g., multi-path fading) [132] need to be considered when deriving proximity information.

Example Usage: The presented dataset can be explored from both user and environment perspectives. The nature of user-side acquisition makes it straightforward to navigate the context changes for a given user through time. For instance, one of the things the first row in table A1 can tell us is that *host* #4 is relatively close to a device d_i(C1:DA:6A:2A:4E:D5) at time t. Then we can do a simple filter on the *HostId* and timestamp columns to figure out what other devices *host* #4 had encountered in an arbitrary look back time(e.g., last five minutes). Alternatively, we could make predictions on what *host* #4 would encounter in the next five minutes and the subsequently assess the accuracy of these predictions. From the environment perspective, we have leveraged the dataset to develop an approach for *continuous authorization* in smart buildings [60]. The dataset allows us to use the frequency of beacons

within a given interval of time to reflect the "presence" of a device from the perspective of a *host*. The density of beacons in the dataset is sufficient for us to generate useful access control rules at the smart building scale, based on attributes derived from this "presence" information. Existing datasets do not provide the necessary granularity of beacons necessary to support these types of applications.

In this sub-section, we presented dataset that provides a new perspective on studying people-centric sensing that is essential in continuous context-aware applications. The people-centric nature of the dataset lends itself to the development of privacy conscious applications, where the sensors take a passive role and the users are in control of the sensor aggregation. The data collection is publicly available on the Zenodo open science platform and can be found at [77].

5.3 Collaboration Performance in the Real World

5.3.1 Opportunistic Connection Extraction and Experimental Setup

In section 3.3 and 4.4, we evaluate the performance of sensing collaboration with realistic scenarios using our smart-city simulator. The SCENTS framework takes an active request-response approach and we evaluated its sensor selection performance in the fleet, commuter, and individual scenarios. The PINCH framework opportunistically distributes context information in a local area with *proactive* exchange schemes. Unlike the SCENTS approach, decisions are made locally based on the (potential) anticipations for context

data. The two approaches have different goals. SCENTS aims to balance sensing fulfillment and the fairness of energy consumption. PINCH proactively determine the most *useful* type of context to sense and share, based on the estimated neighborhood status and demand model. But they both utilize the opportunistic connections between the nearby devices and use BLEnd under the surface. This provides a chance for us to evaluate the two collaboration schemes with the data collection as described in section 5.2.1. The extensive, context-rich dataset collected from our testbed contains the *real* user traces and their opportunistic connections with each other and environmental IoT devices. Through the experiments and analysis in the rest of this chapter we will be able to evaluate the benefits of proximity-based sensing collaboration over a long time period of *hybrid* scenarios.

To simulate the collaboration operations, we first need to extract the opportunistic connections between IoT devices from the dataset [78]. Figure 5.7 highlights the key concept of this extraction process. Because of the

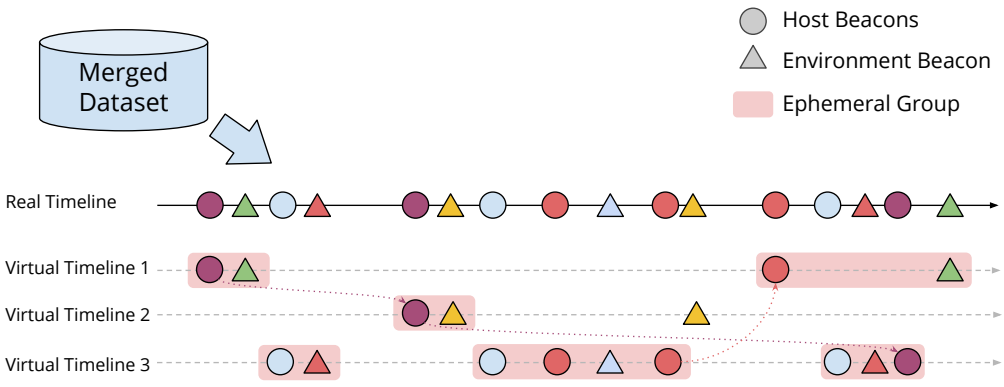


Figure 5.7: Extract opportunistic connections from the dataset.

data set is merged from the data collection from individual devices, distance between the beacons in the time-series does not reveal the spatial relation of two devices. In order to simulate the behaviors of the mobile hosts without running separate simulations for each individual, we split the beacons into multiple *virtual timelines* such that the spatiotemporal aspects of received beacons can be revealed for the spontaneous group extraction. This process is done with the help of environmental beacons. The simplest approach is illustrated in figure 5.7. For instance, the purple host[6] receiving a environmental beacon from the green stationary indicates the co-existence between the two. Later on the leaving of the same host can be determined when it visits the location that has yellow beacons(i.e., purple host transit to virtual timeline 2). Similarly, the host leaves this ephemeral group when it comes to the range of red stationary (i.e., enters virtual timeline 3). In this way, we are able to recover the opportunistic wireless connections, between the IoT devices in proximity, and grouping the beacons into ephemeral co-located groups (denoted as rectangles with pink background).

Then we implemented a Python version of SCENTS and a trace executor in the Google Colaboratory Python3 runtime to simulate the sensing collaboration based on the user-side collected beacons presented in section 5.2.1. Similar to the previous evaluation of SCENTS in the street scenarios (section 3.3.4), we build the trace executor with the flexibility of changing the on-board sensing capability of the IoT devices and the query frequency of

[6]The mobile host with purple beacons in the figure.

each host. This allows us to simulate the SCENTS behavior with real life user events.

5.3.2 Sensing Collaboration in the Smart-building

5.3.2.1 Weekday Evaluation

As discussed in section 5.2.3, the data collection contains more than 20 million beacon entries after we filtered out the off-campus beacons for privacy preservation. For the remaining on-campus entries, there are numerous scenarios beyond what we have considered with the smart-city simulator in section 3.3.3 and 4.4. Thus, simulating the SCENTS system with the more *organic* traces between users and the environment and between the user themselves, which provided from the dataset, can give us more insights on how SCENTS benefits IoT sensing applications in the real world.

In the first experiment, we focus on the workday scenarios of the sensing collaboration between the IoT devices. The IoT hosts are the mobile devices carried by the participants during the data collection period. Each host is capable of sensing 70% of the context types that the tenant application is interested in, using its on-device sensor equipment. The other stationary devices, that were attached to 22 places in the building, are fully capable for all sensors.

The applications running on the mobile hosts are set to *randomly* query the SCENTS framework *every minute*. In other words, each SCENTS agent expects to receive a sensing query of random context type from the upper-layer

at the intervals of 60 seconds. Note that we filtered out the some beacons for privacy reasons as described in section 5.2.3. An interaction is captured only when they are co-located with a stationary device or another participant in the data-set. So if a group of participants leave the building to have lunch, their simulated applications will continue to query SCENTS framework for context information as long as there is another participant in proximity.

In this experiment, the query result is classified into four categories: 1) *answered_original* means the query is fulfilled with on-device sensors, 2) *failed* denotes a query that is not fulfilled because of no capable candidate nearby or communication failure, 3) *fulfilled_original* indicates that the query is fulfilled by collaborating with a nearby device, which queried its own sensor and delivered the result back using BLEnd beacons, and 4) *fulfilled_reused* denotes the query is answered through collaboration and the context is reused from a valid sensor reading.

Next we use the trace executor to run SCENTS on the weekdays in the week of April 15, 2019. To intuitively visualize the collaboration performance we resampled the time series over *one hour* frequency and stacked the result of four categories respectively. The result is shown in figure 5.8. The x-axis is the hourly ticked timeline of the week and the y-axis is the number of times queries were generated in each hour. The colored segments of each bar denote the number of times the sensing queries resulted in the corresponding status (within that hour). The bars are separated into five clusters since those are the times when most participants at work or stay with other participants.

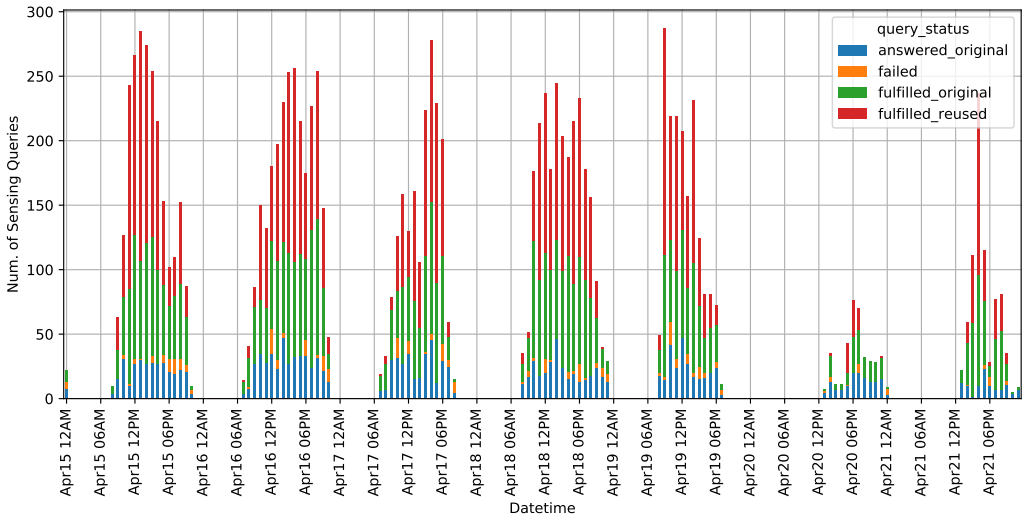


Figure 5.8: SCENTS query results over a week.

The most noticeable portions of the bars are colored in red (fulfilled_reused status), which indicates that a significant portion of the sensor readings are saved from the caching values through direct or indirect collaboration. The second greatest portion of the result status is in green (fulfilled_original), which means the query is sent to a nearby device where the peer executes the sensing task and broadcasts the result reading back to the requester. The peer device can be either a mobile host or a stationary device in proximity. The energy consumption in the latter case is ignored since the environmental sensor kits can easily have external power source. As we can see these two status, enabled by SCENTS collaboration, contributed most of the sensing fulfillment. There is a small portion of the queries are answered with on-device sensors (showing in blue). The yellow colored failures cases are rare.

From above analysis, we developed the impression of how sensing col-

laboration of SCENTS benefits the mobile IoT applications in a real world smart-building setting over a week's time. In the next section, we are going to parameterize the potential requirements of the upper-layer applications as well as the SCENTS middleware settings to deep dive into the performance of sensing collaboration over the whole six weeks' data.

5.3.2.2 SCENTS Performance Over Six Weeks

Up to this point, we have gained some insights on how SCENTS can help the collaboration among the opportunistically connected devices. In order to evaluate SCENTS thoroughly, we conduct three sets of experiments over the entire data collection. These experiments are parameterized with three control variables: 1) device capability, 2) sensing frequency, and 3) combination coefficient α of algorithm 1.

Varying device capability: In the first set of experiments, we simulate the diversity of the participating devices in our system. The common foundation of the different collaboration approaches presented in this thesis is the continuous discovery protocol from section 2.2. One of the advantages of BLEnd is that the underlying wireless communication technology BLE is pervasively available. From this perspective we simulate the heterogeneity in device sensing capability with random ratios varying from 10% to 90%. At the beginning of each run, each IoT host is assigned with the capability of every context type based on the given ratio. A static context demand model ,with equal importance of the context types described in section 5.2.3, is used throughout all

runs. Other parameters are set the same as in the previous section. The two million entries from the data set are iterated repetitively with each setting.

Table 5.1: Fulfillment with various capability ratios.

Capability Ratio	Num. of Queries	Num. of Failures	Num. of Sensor Readings (mobile)	Num. of Cache Readings
90%	84707	924 (1.09%)	22027	48653
70%	84707	1084 (1.28%)	22141	48214
50%	84707	1767 (2.09%)	19999	46513
30%	84707	3254 (3.84%)	12774	39487
10%	84707	8417 (9.94%)	1087	42612

Table 5.1 highlights the collaboration performance across the decreasing capability ratios in the first column. The numbers of generated sensing queries are the same since the total *valid* time passed in the user-collected traces are the same. The third column lists the numbers of sensing failures in each execution. The ratio of encountered failures increase as the sensing capabliity cr decreases as expected. The collaboration benefits are obvious given the fact that the failure ratios in all settings are significantly lower than $(1-cr)$, where the sensing queries of individual hosts could only read on-device sensors. Even with $cr = 10\%$ (i.e., last row of the table), the vast majority of the queries can still be fulfilled with the help from SCENTS. This matches the previous result analysis from section 3.3.4. The total time of sensor readings on the mobile hosts reduces as the device capability decreases. This may indicate that in the lack of on-device sensing capabilities, the sensor selection algorithm favors environmental sensor kits regardless of possible failures caused by movements [7].

[7]Staying closely with co-workers at in the building is one of the common scenarios in the

Table 5.2: Fulfillment with various query frequencies.

Query Interval	Num. of Queries	Num. of Failures	Num. of Sensor Readings (mobile)	Num. of Cache Readings
20 sec.	243598	3580 (1.47%)	27170	174442
1 min.	84707	1084 (1.28%)	22141	48214
3 min.	28712	433 (1.51%)	10005	7715
10 min.	8821	316 (3.58%)	5287	1169
30 min.	3058	174 (5.69%)	1836	169

Varying query frequency: In the second set of experiments, we simulate the various sensing frequencies from the tenant applications, which are hosted on top of the SCENTS middleware. The system architecture is introduced in section 3.2.2. Without loss of generality, we assume a uniform distribution of in the interested context types. Because the entries are merged from user-side acquired collections, we do not have a universal timeline in the dataset. Instead, we work on the (parallel) proximity-based traces which are extracted from mobile hosts' perspectives of the nearby events. This process is conceptually depicted in figure 5.7. As a result, we orchestrate the query scheduler on each hosts to issue a context request within the ephemeral proximity-based group at the intervals from $20s$, $1min$, ..., to $30min$.

The key statistics of the result are shown in table 5.2. The numbers of issued queries (the second column) reduce proportionally as the sensing frequency reduces throughout out the six week's period. Both the absolute numbers of failures (third column) and on-device sensor readings (fourth col-

data collection.

umn) show the same trend, while failure rates increase as the sensing interval grows longer. One possible explanation is rooted from the design of the query scheduler. Devices are deemed to be idle when none of the participants is expressing context interests in proximity. This case could be mitigated with the environmental devices proactively execute sensing queries based on presumed context demand model and share the results, similar to how we build context neighborhoods with the PINCH approach (section 4.2).

Varying Combination Coefficient: SCENTS offers the flexibility of balancing the sensing preferences $\tilde{H}$ and the cost estimation E. We introduced the coefficient α to generate total ordering in the set of $\langle \tilde{H}_{:,k}, E \rangle$ pairs in section 3.2.4. This minor caveat is used in algorithm 1. We conduct the last set of experiments with $\alpha \in \{0.3, 0.6, 0.9\}$. The results are shown in table 5.3. With the query interval set to one minute per query and a 70% capability ratio, the failure rates across three simulations are below 2%. As the coefficient α increases(i.e., prioritizes cost distribution), the number of mobile sensor readings decreases in the third column. This might indicate that the mobile hosts are affected to depend more on the stationary sensor kits with a higher α value and less concerned about mobility-induced failures (failure rate increases in the sceond column).

Table 5.3: Fulfillment with various SCENTS α parameters.

SCENTS Alpha(α)	Num. of Queries	Num. of Failures	Num. of Sensor Readings (mobile)	Num. of Cache Readings
0.3	84707	1066 (1.26%)	16024	38653
0.6	84707	1196 (1.41%)	15442	45624
0.9	84707	1435 (1.69%)	11676	54464

Chapter 6

Conclusion

With the continuous growth of sensor-integrated IoT devices, the requirement for context information has increased dramatically in the ubiquitous computing paradigm. Smart devices are emerging at every corner, interacting with human carried devices, and providing digital assistance. In order to help these context-aware applications to seamlessly acquire sensed information anytime and anywhere, we explored several approaches in this dissertation to facilitate collaboration among the co-located IoT devices.

We started by creating a key building block of collaboration in IoT proximity networks: a continuous neighbor discovery protocol. We acknowledged the fact that low-power wireless communication technologies are commonly equipped in today's IoT devices, and introduced a lower-level software BLEnd to enable the automatic and energy-efficient peer discovery using wireless beacons. By incorporating the BLEnd with BLE characteristics, this low duty-cycle protocol enables the mobile devices to be constantly aware of the digital resources in the nearby environment (i.e. in range of wireless communication) with significantly improved sustainability.

From the second part, BLEnd began to serve as the energy-efficient

communication substrate for building collaborative middlewares that allow us to take advantage of the pervasive yet opportunistic wireless connections. To mitigate the sensing incapabilities of individual devices and distributing the sensing cost in proximity, we presented SCENTS as a generic collaborative sensing framework. SCENTS encapsulates access to both on-device and remote sensors in a non-intrusive manner, and breaks the tight binding between the hardware equipment and IoT application. Each context query from the upper-layer tenants is handled by the SCENTS middleware with the proper fulfilling method at the moment. Besides optimizing towards energy distribution and fulfillment rate, it also uses the notion of *safe distance* to tackle mobility-induced failures.

Besides the active context requesting and sharing approach, we took a step further to allow the ad hoc neighbors to act in fidelity and concordance with their self-organized context neighborhoods. The PINCH framework employs several heuristics and make the host device proactivly sense and disseminate the context of choice, based on their limited view of the neighborhood. We then built a OMNeT-based simulator to showcase the potential of implicit context sharing in multiple realistic smart-city scenarios.

We further explored the sensing task assignment in a distributed setting and studied the software integration with off-the-shelf, energy restricted IoT sensor kits using the Stacon system. We showcased its resilience against network dynamics in a radio-noisy environment and its ability to adjust the sensing assignments as from a centralized scheduler, with expected latency.

Lastly, we deployed an IoT sensing testbed with 48 environmental beacons and 7 mobile nodes in a university building. The acquired context-rich, real-life dataset helped us to assess the feasibility, cost-effectiveness, and privacy factors of sensing systems. The extensive dataset is publicly available to encourage further context-awareness studies in a smart-building setting. Based on the extracted the opportunistic connections, we examined the performance of SCENTS in weekdays. The benefits of SCENTS collaborative sensing approach are evaluated in various settings over a six-week duration.

Appendix

Table A1: Sample data entries in the beacon table.

HostId	*Timestamp(ms)*	*Bluetooth Address*	*RSSI*	*Sound avg.*	Sound max	...	*Temp.*	*Humidity*	*Air Pressure*	*eCO^2(ppm)*	*tVOC(ppb)*
4	1554330245495	C1:DA:6A:2A:4E:D5	-67	269	669	...	22.96	48%	994.88	427	4
1	1556731374232	D1:DE:0B:60:CA:95	-99	1215	2193	...	27.70	73%	996.43	400	0
0	1557160392903	F0:55:C1:1B:5D:6B	-98	306	669	...	23.29	86%	993.97	408	1

Bibliography

[1] Gregory D Abowd. Beyond weiser: From ubiquitous to collective computing. *Computer*, 49(1):17–23, 2016.

[2] Adafruit. Adafruit Ultimate GPS. `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf`.

[3] Joshua Adkins, Branden Ghena, Neal Jackson, Pat Pannuto, Samuel Rohrer, Bradford Campbell, and Prabal Dutta. The signpost platform for city-scale sensing. *arXiv preprint arXiv:1802.07805*, 2018.

[4] Charu C Aggarwal, Amotz Bar-Noy, and Simon Shamoun. On sensor selection in linked information networks. *Computer Networks*, 126:100–113, 2017.

[5] Gianluca Aloi, Giuseppe Caliciuri, Giancarlo Fortino, Raffaele Gravina, Pasquale Pace, Wilma Russo, and Claudio Savaglio. A mobile multi-technology gateway to enable iot interoperability. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pages 259–264. IEEE, 2016.

[6] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. Rio: A system solution for sharing i/o between mobile systems. In *Proceed-*

ings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, pages 259–272, 2014.

[7] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. Rio: a system solution for sharing i/o between mobile systems. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 259–272. ACM, 2014.

[8] Emmanuelle Anceaume, Bernadette Charron-Bost, Pascale Minet, and Sam Toueg. On the formal specification of group membership services. Technical report, Cornell University, 1995.

[9] Jorgen Bach Andersen, Theodore S Rappaport, and Susumu Yoshida. Propagation measurements and models for wireless communications channels. *IEEE Communications Magazine*, 33(1):42–49, 1995.

[10] Michael P Andersen, John Kolb, Kaifei Chen, Gabriel Fierro, David E Culler, and Raluca Ada Popa. Wave: A decentralized authorization system for iot via blockchain smart contracts. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-234*, 2017.

[11] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International workshop on ambient assisted living*, pages 216–223. Springer, 2012.

[12] Akin Avci, Stephan Bosch, Mihai Marin-Perianu, Raluca Marin-Perianu, and Paul Havinga. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In *Architecture of computing systems (ARCS), 2010 23rd international conference on*, pages 1–10. VDE, 2010.

[13] Mehedi Bakht, Matt Trower, and Robin Hilary Kravets. Searchlight: Won't you be my neighbor? In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 185–196. ACM, 2012.

[14] Rajesh Krishna Balan, Youngki Lee, Tan Kiat Wee, and Archan Misra. The challenge of continuous mobile context sensing. In *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8. IEEE, 2014.

[15] Fehmi Ben Abdesslem, Andrew Phillips, and Tristan Henderson. Less is more: energy-efficient mobile sensing with senseless. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 61–62. ACM, 2009.

[16] Karl Benkic, Marko Malajner, P Planinsic, and Z Cucej. Using rssi value for distance estimation in wireless sensor networks based on zigbee. In *Systems, signals and image processing, 2008. IWSSIP 2008. 15th international conference on*, pages 303–306. IEEE, 2008.

[17] Andreas Biri, Pat Pannuto, and Prabal Dutta. Demo abstract: Totternary-a wearable platform for social interaction tracking. In *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 346–347. IEEE, 2019.

[18] D Blaauw, D Sylvester, P Dutta, Y Lee, I Lee, S Bang, Y Kim, G Kim, P Pannuto, Y-S Kuo, et al. Iot design space challenges: Circuits and systems. In *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, pages 1–2. IEEE, 2014.

[19] Bluetooth SIG. Bluetooth core specification 4.2. `https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439`, December 2014.

[20] Mate Boban, Joao Barros, and Ozan K Tonguz. Geometry-based vehicle-to-vehicle channel modeling for large-scale simulation. *IEEE Transactions on Vehicular Technology*, 63(9):4146–4164, 2014.

[21] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):33, 2014.

[22] Andrew T Campbell, Shane B Eisenman, Nicholas D Lane, Emiliano Miluzzo, Ronald A Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12(4), 2008.

[23] Rohit Chaudhri, Waylon Brunette, Mayank Goel, Rita Sodt, Jaylen VanOrden, Michael Falcone, and Gaetano Borriello. Open data kit sensors: Mobile data collection with wired and wireless sensors. In *Proceedings of the 2nd ACM Symposium on Computing for Development*, pages 9:1–9:10, 2012.

[24] Yufei Chen and Chao Shen. Performance analysis of smartphone-sensor behavior for human activity recognition. *Ieee Access*, 5:3095–3110, 2017.

[25] Yongyang Cheng, Shuai Zhao, Bo Cheng, Shoulu Hou, Xiulei Zhang, and Junliang Chen. A distributed event-centric collaborative workflows development system for iot application. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2547–2550. IEEE, 2017.

[26] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[27] Hosik Cho, Jianxun Ji, Zili Chen, Hyuncheol Park, and Wonsuk Lee. Measuring a distance between things with improved accuracy. *Procedia Computer Science*, 52:1083–1088, 2015.

[28] Sungmin Cho and Christine Julien. Chitchat: Navigating tradeoffs in device-to-device context sharing. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE, 2016.

[29] Tanzeem Khalid Choudhury. *Sensing and modeling human networks*. PhD thesis, Massachusetts Institute of Technology, 2004.

[30] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 211–224. ACM, 2008.

[31] Paolo Costa, Luca Mottola, Amy L Murphy, and Gian Pietro Picco. Programming wireless sensor networks with the teeny lime middleware. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 429–449. Springer, 2007.

[32] Tathagata Das, Prashanth Mohan, Venkata N Padmanabhan, Ramachandran Ramjee, and Asankhaya Sharma. Prism: platform for remote sensing using smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 63–76. ACM, 2010.

[33] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. An iot gateway centric architecture to provide novel m2m services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 514–519. IEEE, 2014.

[34] Debraj De, Pratool Bharti, Sajal K Das, and Sriram Chellappan. Mul-

timodal wearable sensing for fine-grained activity recognition in healthcare. *IEEE Internet Computing*, 19(5):26–35, 2015.

[35] Adrian A de Freitas and Anind K Dey. The group context framework: An extensible toolkit for opportunistic grouping and collaboration. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1602–1611. ACM, 2015.

[36] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, pages 588–599, 2004.

[37] Prabal Dutta and David Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 71–84. ACM, 2008.

[38] Eigen3 template library. `http://eigen.tuxfamily.org/`.

[39] Shane B Eisenman, Nicholas D Lane, Emiliano Miluzzo, Ronald A Peterson, Gahng-Seop Ahn, and Andrew Campbell. Metrosense project: People-centric sensing at scale. In *Workshop on World-Sensor-Web (WSW 2006), Boulder*. Citeseer, 2006.

[40] Shane B Eisenman, Emiliano Miluzzo, Nicholas D Lane, Ronald A Peterson, Gahng-Seop Ahn, and Andrew T Campbell. Bikenet: A mobile

sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):6, 2009.

[41] Ahmed Faheem, Reino Virrankoski, and Mohammed Elmusrati. Improving rssi based distance estimation for 802.15. 4 wireless sensor networks. In *Wireless Information Technology and Systems (ICWITS), 2010 IEEE International Conference on*, pages 1–4. IEEE, 2010.

[42] Ramsey Faragher and Robert Harle. Location fingerprinting with bluetooth low energy beacons. *IEEE journal on Selected Areas in Communications*, 33(11):2418–2428, 2015.

[43] Zhenni Feng, Yanmin Zhu, Qian Zhang, Lionel M Ni, and Athanasios V Vasilakos. Trac: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1231–1239. IEEE, 2014.

[44] Chien-Liang Fok, G-C Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 653–662. IEEE, 2005.

[45] Davide Giovanelli, Bojan Milosevic, Csaba Kiraly, Amy Lynn Murphy, and Elisabetta Farella. Dynamic group management with bluetooth low energy. In *Smart Cities Conference (ISC2), 2016 IEEE International*, pages 1–6. IEEE, 2016.

[46] Daniel Golovin, Matthew Faulkner, and Andreas Krause. Online distributed sensor selection. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 220–231. ACM, 2010.

[47] Ulrike Gretzel, Marianna Sigala, Zheng Xiang, and Chulmo Koo. Smart tourism: foundations and developments. *Electronic Markets*, 25(3):179–188, 2015.

[48] Yu Guan and Thomas Plötz. Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(2):11, 2017.

[49] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[50] Zongjian He, Jiannong Cao, and Xuefeng Liu. High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2542–2550. IEEE, 2015.

[51] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[52] Jie Hua, Chenguang Liu, Tomasz Kalbarczyk, Catherine Wright, Gruia-Catalin Roman, and Christine Julien. rIoT: Enabling seamless Context-Aware Automation in the Internet of Things. In *The 16th IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, Monterey, CA, USA, 2019. IEEE.

[53] Qingfeng Huang, Christine Julien, and G-C Roman. Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(2):192–205, 2004.

[54] Syed Rafiul Hussain, Shagufta Mehnaz, Shahriar Nirjon, and Elisa Bertino. Seamless and secure bluetooth le connection migration. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 147–149. ACM, 2017.

[55] INET. `https://inet.omnetpp.org/`.

[56] Martin Jonsson. Supporting context awareness with the context shadow infrastructure. *Wkshp. on Affordable Wireless Services and Infrastructure*, 2003.

[57] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2009.

[58] Christine Julien, Chenguang Liu, Amy L Murphy, and Gian Pietro Picco. Blend: practical continuous neighbor discovery for bluetooth low energy.

In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 105–116. ACM, 2017.

[59] Sanem Kabadayi, Adam Pridgen, and Christine Julien. Virtual sensors: Abstracting data from physical sensors. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 587–592. IEEE Computer Society, 2006.

[60] Tomasz Kalbarczyk, Chenguang Liu, Jie Hua, and Christine Julien. LAD: Learning access control policies and detecting access anomalies in smart environments. In *The 16th IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, Monterey, CA, USA, 2019. IEEE.

[61] Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks*, pages 350–361. ACM, 2010.

[62] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 267–280. ACM, 2008.

[63] Aman Kansal, Scott Saponas, AJ Brush, Kathryn S McKinley, Todd Mytkowicz, and Ryder Ziola. The latency, accuracy, and battery (lab) abstraction: programmer productivity and energy efficiency for continuous mobile context sensing. *ACM SIGPLAN Notices*, 48(10):661–676, 2013.

[64] Merkouris Karaliopoulos, Orestis Telelis, and Iordanis Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2254–2262. IEEE, 2015.

[65] Matthew Keally, Gang Zhou, Guoliang Xing, and Jianxin Wu. Remora: Sensing resource sharing among smartphone-based body sensor networks. In *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2013.

[66] Junaid Ahmed Khan, Romain Pujol, Razvan Stanica, and Fabrice Valois. On the energy efficiency and performance of neighbor discovery schemes for low duty cycle iot devices. In *Proceedings of the 14th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, pages 63–70, 2017.

[67] Philipp H Kindt, Marco Saur, Michael Balszun, and Samarjit Chakraborty. Neighbor discovery latency in ble-like protocols. *IEEE Transactions on Mobile Computing*, 2017.

[68] Charalampos Konstantopoulos, Grammati Pantziou, Damianos Gavalas, Aristides Mpitziopoulos, and Basilis Mamalis. A rendezvous-based approach enabling energy-efficient sensory data collection with mobile sinks. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):809–817, 2012.

[69] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522. ACM, 2010.

[70] Mihai T Lazarescu. Design of a wsn platform for long-term environmental monitoring for iot applications. *IEEE Journal on emerging and selected topics in circuits and systems*, 3(1):45–54, 2013.

[71] Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. Comon: Cooperative ambience monitoring platform with continuity and benefit awareness. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2012.

[72] Youngki Lee, Seungwoo Kang, Chulhong Min, Younghyun Ju, Inseok Hwang, and Junehwa Song. Comon+: A cooperative context monitoring system for multi-device personal sensing environments. *IEEE Transactions on Mobile Computing*, 15(8):1908–1924, 2015.

[73] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. Multiprogramming a 64kb

computer safely and efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 234–251. ACM, 2017.

[74] Amit A. Levy, James Hong, Laurynas Riliskis, Philip Levis, and Keith Winstein. Beetle: Flexible communication for bluetooth low energy. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 111–122, 2016.

[75] Chenguang Liu, Jie Hua, Changyong Hu, and Christine Julien. Stacon: Self-stabilizing context neighborhood for mobile IoT devices. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 361–363. IEEE, 2019.

[76] Chenguang Liu, Jie Hua, and Christine Julien. Scents: Collaborative sensing in proximity IoT networks. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 189–195. IEEE, 2019.

[77] Chenguang Liu, Jie Hua, Tomasz Kalbarczyk, Sangsu Lee, and Christine Julien. Dataset: User side acquisition of People-Centric Sensing in the Internet-of-Things. `https://doi.org/10.5281/zenodo.3450691`.

[78] Chenguang Liu, Jie Hua, Tomasz Kalbarczyk, Sangsu Lee, and Christine Julien. Dataset: User side acquisition of people-centric sensing in the internet-of-things. In *Second workshop on Data Acquisition To Analysis,(New York, NY), ACM*, 2019.

[79] Chenguang Liu and Christine Julien. Pervasive context sharing in magpie: adaptive trust-based privacy protection. In *International Conference on Mobile Computing, Applications, and Services*, pages 122–139. Springer, 2015.

[80] Chenguang Liu, Christine Julien, and Amy Lynn Murphy. Pinch: Self-organized context neighborhoods for smart environments. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 120–129. IEEE, 2018.

[81] Jia Liu, Canfeng Chen, Yan Ma, and Ying Xu. Energy analysis of device discovery for bluetooth low energy. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–5. IEEE, 2013.

[82] Jie Liu, Bodhi Priyantha, Ted Hart, Yuzhe Jin, Woosuk Lee, Vijay Raghunathan, Heitor S Ramos, and Qiang Wang. Co-gps: energy efficient gps sensing with cloud offloading. *IEEE Transactions on Mobile Computing*, 15(6):1348–1361, 2016.

[83] Dimitrios Lymberopoulos, Jie Liu, Xue Yang, Romit Roy Choudhury, Vlado Handziski, and Souvik Sen. A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *Proceedings of the 14th international conference on information processing in sensor networks*, pages 178–189. ACM, 2015.

[84] Claudio Martella, Armando Miraglia, Marco Cattani, and Maarten Van Steen. Leveraging proximity sensing to mine the behavior of museum visitors.

In *Pervasive Computing and Communications (PerCom), 2016 IEEE International Conference on*, pages 1–9. IEEE, 2016.

[85] Santiago Mazuelas, Alfonso Bahillo, Ruben M Lorenzo, Patricia Fernandez, Francisco A Lago, Eduardo Garcia, Juan Blas, and Evaristo J Abril. Robust indoor positioning provided by real-time rssi values in unmodified wlan networks. *IEEE Journal of selected topics in signal processing*, 3(5):821–831, 2009.

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

[87] Esunly Medina, David López, Dolors Royo, Roc Meseguer, and Sergio F Ochoa. Cosp: A collaborative sensing platform for mobile applications. In *2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 377–382. IEEE, 2015.

[88] Yan Michalevsky, Suman Nath, and Jie Liu. Mashable: Mobile applications of secret handshakes over bluetooth le. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 387–400. ACM, 2016.

[89] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, and N. Asokan. Conxsense: automated context classification for context-aware access control. In *Proc. of ASIACCS*, pages 293–304. ACM, 2014.

[90] Brandon J Moore and Kevin M Passino. Distributed task assignment for mobile agents. *IEEE Transactions on automatic control*, 52(4):749–753, 2007.

[91] Luca Mottola and Gian Pietro Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 150–168, 2006.

[92] Suman Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 29–42. ACM, 2012.

[93] Jim Mee Ng and Yan Zhang. A mobility model with group partitioning for wireless ad hoc networks. In *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*, volume 2, pages 289–294. IEEE, 2005.

[94] Nordic. Nordic Thingy52 Sensor Tag. `https://www.nordicsemi.com/eng/Products/Nordic-Thingy-52`.

[95] Open Scene Graph. `http://www.openscenegraph.org/`.

[96] Open Street Map. `http://www.openstreetmap.org/`.

[97] Pedestrian and Bicycle Information Center. Walking School Buses and Bicycle Trains. `http://guide.saferoutesinfo.org/encouragement/walking_school_bus_or_bicycle_train.cfm`.

[98] Gian Pietro Picco, Davide Molteni, Amy L Murphy, Federico Ossi, Francesca Cagnacci, Michele Corrà, and Sandro Nicoloso. Geo-referenced proximity detection of wildlife with wildscope: design and characterization. In *Proceedings of the 14th international conference on information processing in sensor networks*, pages 238–249. ACM, 2015.

[99] Aveek Purohit, Bodhi Priyantha, and Jie Liu. Wiflock: Collaborative group discovery and maintenance in mobile sensor networks. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 37–48. IEEE, 2011.

[100] Yihong Qi. *Wireless geolocation in a non-line-of-sight environment*. PhD thesis, Princeton University Princeton, 2003.

[101] Ying Qiu, Shining Li, Xiangsen Xu, and Zhigang Li. Talk more listen less: Energy-efficient neighbor discovery in wireless sensor networks. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[102] Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Peter J Rentfrow. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 85–93. IEEE, 2013.

[103] Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo,

and Peter J Rentfrow. Smartphone sensing offloading for efficiently supporting social sensing applications. *Pervasive and Mobile Computing*, 10:3–21, 2014.

[104] Kiran K Rachuri, Cecilia Mascolo, and Mirco Musolesi. Energy-accuracy trade-offs of sensor sampling in smart phone based sensing systems. In *Mobile Context Awareness*, pages 65–76. Springer, 2012.

[105] Valentin Radu, Catherine Tong, Sourav Bhattacharya, Nicholas D Lane, Cecilia Mascolo, Mahesh K Marina, and Fahim Kawsar. Multimodal deep learning for activity and context recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4):157, 2018.

[106] M.S. Rahaman, H. Pare, J. Liono, F. D. Salim, Y. Ren, J. Chan, S. Kudo, T. Rawling, and A. Sinickas. Occuspace: Towards a robust occupancy prediction system for activity based workplace. In *Proc. of PerCom Workshops*. IEEE, 2019.

[107] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. What does model-driven data acquisition really achieve in wireless sensor networks? In *2012 IEEE International Conference on Pervasive Computing and Communications*, pages 85–94, 2012.

[108] Gruia-Catalin Roman, Qingfeng Huang, and Ali Hazemi. Consistent group membership in ad hoc networks. In *Proceedings of the 23rd in-*

ternational conference on Software engineering, pages 381–388. IEEE Computer Society, 2001.

[109] Matteo Saloni, Christine Julien, Amy L Murphy, and Gian Pietro Picco. Lasso: A device-to-device group monitoring service for smart cities. In *Smart Cities Conference (ISC2), 2017 International*, pages 1–6. IEEE, 2017.

[110] Nicu Sebe. Human-centered computing. In *Handbook of ambient intelligence and smart environments*, pages 349–370. Springer, 2010.

[111] Ben Shaw, Martha Bicket, Bridget Elliott, Ben Fagan-Watson, Elisabetta Mocca, and Mayer Hillman. Children's independent mobility: an international comparison and recommendations for action. 2015.

[112] Xiang Sheng, Jian Tang, and Weiyi Zhang. Energy-efficient collaborative sensing with mobile phones. In *INFOCOM, 2012 Proceedings IEEE*, pages 1916–1924. IEEE, 2012.

[113] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[114] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *Wireless Communications and*

Networking Conference Workshops (WCNCW), 2012 IEEE, pages 232–237. IEEE, 2012.

[115] Mitali Singh and Viktor K Prasanna. A hierarchical model for distributed collaborative computation in wireless sensor networks. *International Journal of Foundations of Computer Science*, 15(03):485–506, 2004.

[116] Alberto Speranzon, Carlo Fischione, and Karl Henrik Johansson. Distributed and collaborative estimation over wireless sensor networks. In *Decision and Control, 2006 45th IEEE Conference on*, pages 1025–1030. IEEE, 2006.

[117] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. A berkeley view of systems challenges for AI. *arXiv preprint arXiv:1712.05855*, 2017.

[118] William Su, Sung-Ju Lee, and Mario Gerla. Mobility prediction and routing in ad hoc wireless networks. *International journal of network management*, 11(1):3–30, 2001.

[119] Hongsuda Tangmunarunkit, Cheng-Kang Hsieh, Brent Longstaff, S Nolen, John Jenkins, Cameron Ketcham, Joshua Selsky, Faisal Alquaddoomi, Dony George, Jinha Kang, et al. Ohmage: A general and extensible end-to-end participatory sensing platform. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):38, 2015.

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

[121] Güliz Seray Tuncay, Giacomo Benincasa, and Ahmed Helmy. Autonomous and distributed recruitment and data collection framework for opportunistic sensing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(4):50–53, 2013.

[122] Andras Varga. OMNeT++. https://www.omnetpp.org/.

[123] Karen H Wang and Baochun Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 2, pages 1017–1021. IEEE, 2002.

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

[125] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 179–192. ACM, 2009.

[126] Liangxiong Wei, Beisi Zhou, Xichu Ma, Dexin Chen, Jingyu Zhang, Jian Peng, Qian Luo, Limin Sun, Dingcheng Li, and Liangyin Chen.

Lightning: a high-efficient neighbor discovery protocol for low duty cycle wsns. *IEEE Communications Letters*, 20(5):966–969, 2016.

[127] Mark Weiser and John Seely Brown. The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer, 1997.

[128] Matt Welsh and Geoffrey Mainland. Programming sensor networks using abstract regions. In *NSDI*, volume 4, 2004.

[129] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, 2004.

[130] Jason Wiese, Patrick Gage Kelley, Lorrie Faith Cranor, Laura Dabbish, Jason I Hong, and John Zimmerman. Are you close with me? are you nearby? investigating social groups, closeness, and willingness to share. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 197–206, 2011.

[131] Hanno Wirtz, Torsten Zimmermann, Matteo Ceriotti, and Klaus Wehrle. Encrypting data to pervasive contexts. In *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*, pages 309–315. IEEE, 2017.

[132] Rong-Hou Wu, Yang-Han Lee, Hsien-Wei Tseng, Yih-Guang Jan, and Ming-Hsueh Chuang. Study of characteristics of rssi signal. In *2008*

IEEE International Conference on Industrial Technology, pages 1–3. IEEE, 2008.

[133] Daqing Zhang, Haoyi Xiong, Leye Wang, and Guanling Chen. Crowdrecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 703–714. ACM, 2014.

[134] Desheng Zhang, Tian He, Yunhuai Liu, Yu Gu, Fan Ye, Raghu K Ganti, and Hui Lei. Acc: generic on-demand accelerations for neighbor discovery in mobile applications. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 169–182. ACM, 2012.

[135] Rong Zheng, Jennifer C Hou, and Lui Sha. Asynchronous wakeup for ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 35–45, 2003.

[136] Xi Zheng, Dewayne E. Perry, and Christine Julien. Braceforce: A middleware to enable sensing integration in mobile applications for novice programmers. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, pages 8–17, 2014.

[137] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. Iot gateway: Bridgingwireless sensor networks into internet of things. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 347–352. Ieee, 2010.

[138] Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 315–330. ACM, 2010.

Vita

Chenguang Liu received the Bachelor of Engineering degree in Software Engineering from Beijing Jiaotong University in 2011, the Master of Engineering degree in Software Engineering from Peking University in 2014, and the Master of Science degree in Electrical and Computer Engineering from the University of Texas at Austin in 2017.

Permanent address: liuchg@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.