SCENTS: Collaborative Sensing in Proximity IoT Networks

Chenguang Liu, Jie Hua, and Christine Julien
The University of Texas at Austin
{liuchg,mich94hj,c.julien}@utexas.edu

***Abstract*—Mobile applications commonly use on-device sensors to continuously provide context: temperature, position, sound, etc. By collaborating to sense context, devices can save energy and share rare capabilities with minimal tradeoffs in sensing quality. Further, by leveraging already active communication behaviors, ambient context information can be collected at very little cost. We present a generic collaborative sensing framework, SCENTS, to support collective sensing for mobile IoT applications. SCENTS leverages two truths about IoT networks: (1) devices participate continuously in low-level *device discovery* mechanisms and (2) nearby devices tend to have similar values for many ambient context properties. We show that SCENTS balances sensing fulfillment and the fairness of energy consumption across devices. We measure the performance of SCENTS using real IoT devices and real world smart-city scenarios.**

***Index Terms*—Context awareness, Sensor systems and applications, Internet of Things**

I. Introduction

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 [6]. 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 [33]. 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 [38].

Section II describes existing work in collaborative context sensing, which often involve cloud-based control [16], [37]. 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. Our key contributions are:

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

II. 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 [36] 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], [10]. Many efforts simplify applications by abstracting or virtualizing their use of sensors [8], [24] or by enabling access to sensors on nearby devices [4], [43]. 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 [35] or intelligently task sensors based on predicted high-level context values, thereby reducing sensing cost [20], [29].

Cloud-assisted collaborative sensing aims to maximize sensing coverage or improve data quality [16], [21], [37]. Most approaches collect information for use offline; a few push data back to distributed devices [9], [27]. In sensor networks, local neighborhoods [28], [40] 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 [17], [34] 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 [12], [23], [36]. Yet opportunistic context sharing over highly dynamic links demands transient light-weight abstractions.

Our own prior work shares context among co-located devices [11], [26]. 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 [5], [13], [22]. SCENTS places contents related to shared sensing in these protocols' periodic beacons sent by the BLEnd protocol [19].

III. 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.

A. 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:

$$\mathsf{FR} = \frac{\text{Number of queries that receive a valid response}}{\text{Number of queries generated}} \tag{1}$$

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

$$\mathsf{UEC} = \max_{(i,j)\in\{n\times n\}} \left(\left| \frac{E_i}{\mathsf{cap}_i} - \frac{E_j}{\mathsf{cap}_j} \right| \right) \tag{2}$$

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 [41]. Also, efforts exist to protect sensed data through encryption [42] and obfuscation [25]. 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.

B. System Overview

As shown in Fig. 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 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.

C. 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

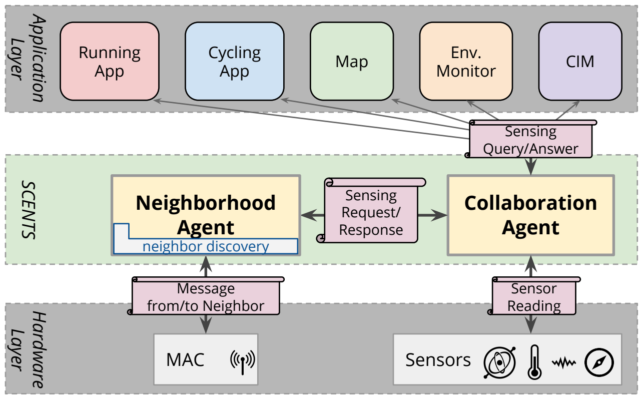

Fig. 1: SCENTS System Overview.

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 [19]. 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 [7], [15] to estimate distance from RSS:

$$PL(dBm) = P_{Tx,j} - P_{Rx,i} = PL_0 + 10 \cdot \gamma \cdot \log(a_{i,j}/a_0) + X \quad (3)$$

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:

$$V_j(t) = \frac{dA_j}{dt} \quad (4)$$

Using this relative velocity, the neighborhood agent estimates the stability of d_j relative to a sensing task using the *safe distance* [17] 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. 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):

$$r_j^{th}(t) = R - \Lambda \cdot |V_j(t)| \quad (5)$$

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:

$$l_j(t) = \frac{1}{1 + e^{\frac{1}{10} \cdot (a_{i,j} - \frac{r_j^{th}(t)}{2})}} \quad (6)$$

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:

$$L_j(t') = l_j(t) \cdot e^{-\Lambda(t'-t)} \quad (7)$$

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

D. 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.

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

ALGORITHM 1: Sensing candidate selection algorithm

```
1  Function SELECTCANDIDATE: query q(t), $\alpha$
2      $s_k \leftarrow q(t)$
3      $L_j(t) \leftarrow l_j(t') \cdot e^{-\Lambda(t-t')}$ for $j = 1 \dots n$
4      construct $w \in \mathbb{R}^n$ s.t. $w_k \leftarrow \frac{\sum_{j=0}^{n} \neg C_{jk} \cdot L_j(t)}{n}$, $k = 1 \dots m$
5      construct $D \in \mathbb{R}^{m \times m}$ s.t. $D_{kk} = w_k$, for $k = 1 \dots m$
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
```

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.

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 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 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.

TABLE I: 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

IV. Experimental Evaluation

A. IoT Device Power Profiling

We first profile the capabilities of an off-the-shelf IoT device: the Nordic Thingy 52 sensor kit [30]. 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 I 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.

B. 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 [39]. The wireless physical and MAC layers are based on the INET Framework v.4.0 [18]. Geographic coordinates and 3D physical obstacles rely on Open Scene Graph [31] and Open Street Map [32]. Algorithm 1 is implemented using the Eigen3 template library [14].

Simulated IoT devices use the device profile above, including multiple sensors, low-radio duty cycle, and short-

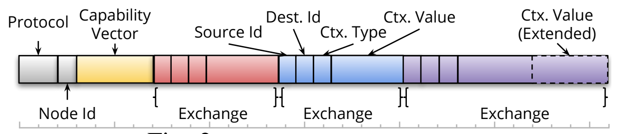

Fig. 2: Beacon Payload Structure

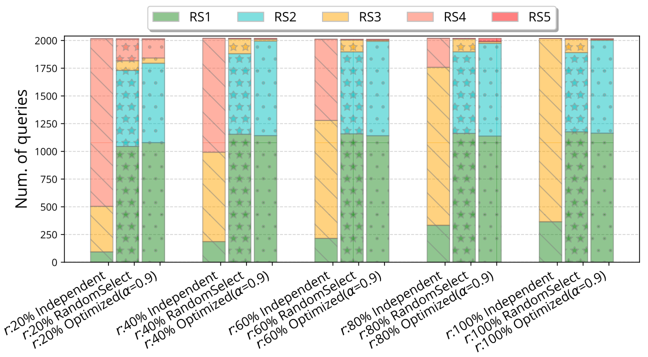

Fig. 3: Query fulfillment comparison.

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 beacon 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 [19], with which our neighborhood agent interfaces to send and receive beacons. Fig. 2 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 exchange segment of up to 8 bytes.

C. Scenarios

We use three realistic and parameterizable real-world scenarios for our evaluation[2]:

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.*

D. 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, 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 range of capability settings. This time we analyze the Fulfillment Ratio metric from Equation 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. 4 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).

E. 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

[2]https://github.com/UT-MPC/collab-sensing-simulation

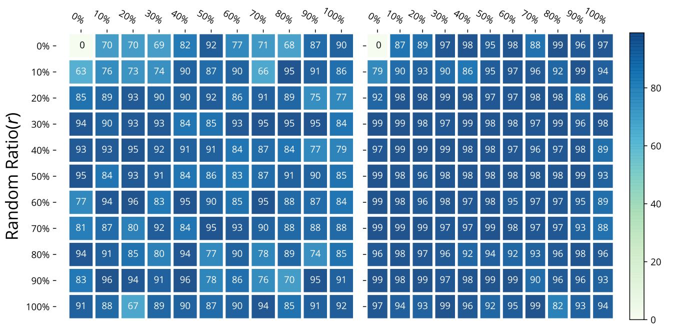

Fig. 4: Fulfillment Ratio with mixed capabilities. Query interval = 5s (left), 10s (right)

city IoT scenario? (2) To what extent is sensing quality traded for increased capability and reduced energy consumption?

We use the Commuters scenario[3] 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. 5a each box shows quartiles of freshness in seconds, while the 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. 5b 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.

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 2. Fig. 6 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

[3]https://youtu.be/KPqtK9t2efs

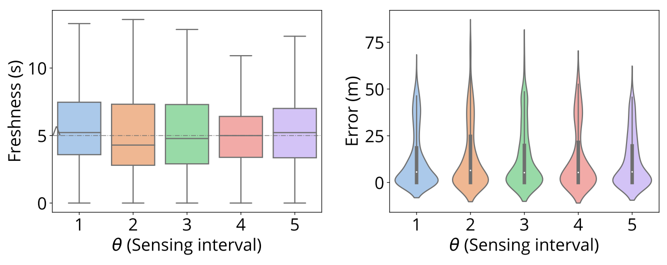

Fig. 5: Freshness & Error (Commuter scenario)

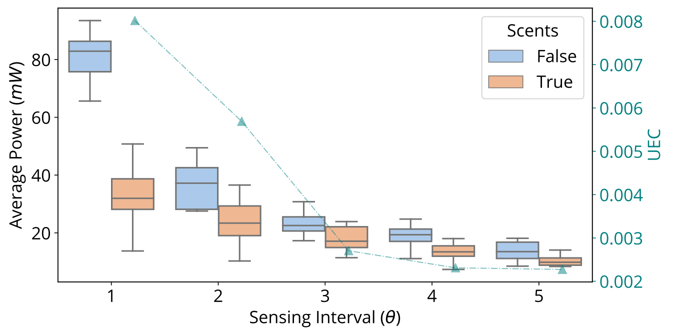

Fig. 6: Energy cost distribution (E) and UEC

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.

F. 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 $100m^2$. 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 7 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 communication. We compute the average power consumption per second for each node, then average over all nodes. Starting from a density of one device per $100m^2$, the user node receives a significant benefit from collaborative sensing and fulfills its sensing demands with 70% power savings.

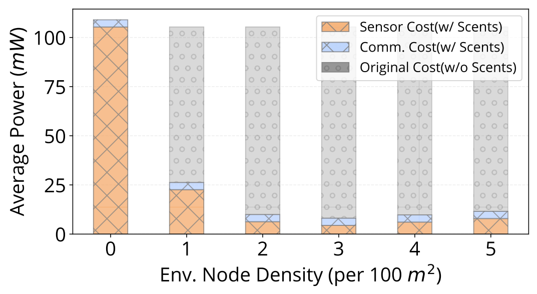

Fig. 7: Energy Cost (Individual scenario)

V. Conclusion and Future Work

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.

Future work could improve the use of collaborative sensing even more. For instance, the faithfulness of a context response could be validated using multi-party computation. A feedback mechanism can be integrated to guide the sensor selection strategy thus improve the QoS of SCENTS in proximity networks. The context types we share are mostly ambient environmental sensor readings. Secure multi-party aggregation could support more types of context sharing. For instance, many people might consider accelerometer data to be particularly private, however, if securely aggregated, one could acquire fused views of the context from a multi-device view and expose less sensitive information.

Acknowledgments

This work was funded in part by the National Science Foundation, CNS-1703497.

References

[1] G. 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] J. Adkins et al. The signpost platform for city-scale sensing. In *Proc. of ACM/IEEE IPSN*, 2018.

[4] A. Amiri Sani et al. Rio: A system solution for sharing i/o between mobile systems. In *Proc. of ACM Mobisys*, pages 259–272, 2014.

[5] M. Bakht, M. Trower, and R. Kravets. Searchlight: Won't you be my neighbor? In *Proc. of ACM Mobicom*, pages 185–196, 2012.

[6] R. K. Balan et al. The challenge of continuous mobile context sensing. In *Proc. COMSNETS*, 2014.

[7] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej. Using RSSI value for distance estimation in wireless sensor networks based on zigbee. In *Proc. of IEEE IWSSIP*, pages 303–306, 2008.

[8] R. Chaudhri et al. Open Data Kit Sensors: Mobile data collection with wired and wireless sensors. In *Proc. of ACM DEV*, 2012.

[9] Y. Cheng et al. A distributed event-centric collaborative workflows development system for IoT application. In *Proc. of IEEE ICDCS*, pages 2547–2550, 2017.

[10] M. Chiang and T. Zhang. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

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

[12] A. de Freitas and A. Dey. The group context framework: An extensible toolkit for opportunistic grouping and collaboration. In *Proc. of ACM CSCW*, 2015.

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

[14] Eigen3 Template Library. http://eigen.tuxfamily.org/.

[15] A. Faheem, R. Virrankoski, and M. Elmusrati. Improving RSSI based distance estimation for 802.15.4 wireless sensor networks. In *Proc. of IEEE ICWITS*, pages 1–4, 2010.

[16] Z. Feng et al. TRAC: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In *Proc. of IEEE INFOCOM*, 2014.

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

[18] INET. https://inet.omnetpp.org/.

[19] C. Julien et al. BLEnd: practical continuous neighbor discovery for bluetooth low energy. In *Proc. of ACM/IEEE IPSN*, 2017.

[20] S. Kang et al. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proc. of ACM Mobisys*, pages 267–280, 2008.

[21] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *Proc. of IEEE INFOCOM*, pages 2254–2262, 2015.

[22] P. Kindt et al. Neighbor discovery latency in ble-like protocols. *IEEE Trans. on Mobile Computing*, 2017.

[23] Y. Lee et al. CoMon: Cooperative ambience monitoring platform with continuity and benefit awareness. In *Proc. of ACM MobiSys*, 2012.

[24] A. Levy et al. Beetle: Flexible communication for bluetooth low energy. In *Proc. of ACM MobiSys*, pages 111–122, 2016.

[25] C. Liu and C. Julien. Pervasive context sharing in magpie: adaptive trust-based privacy protection. In *Proc. of MobiCASE*, pages 122–139. Springer, 2015.

[26] C. Liu, C. Julien, and A. Murphy. PINCH: Self-organized context neighborhoods for smart environments. In *Proc. of IEEE SASO*, 2018.

[27] E. Medina et al. CoSP: A collaborative sensing platform for mobile applications. In *Proc. of IEEE CSCWD*, pages 377–382, May 2015.

[28] L. Mottola and G. Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *Proc. of ICDCS*, 2006.

[29] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *Proc. of ACM Mobisys*, pages 29–42, 2012.

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

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

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

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

[34] A. Purohit, B. Priyantha, and J. Liu. WiFlock: Collaborative group discovery and maintenance in mobile sensor networks. In *Proc. of ACM/IEEE IPSN*, pages 37–48, 2011.

[35] K. Rachuri et al. METIS: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *Proc. of IEEE PerCom*, pages 85–93, 2013.

[36] M. Saloni et al. Lasso: A device-to-device group monitoring service for smart cities. In *Proc. of IEEE ISC2*, pages 1–6, 2017.

[37] X. Sheng, J. Tang, and W. Zhang. Energy-efficient collaborative sensing with mobile phones. In *Proc. of IEEE INFOCOM*, 2012.

[38] G. Tuncay, G. Benincasa, and A. Helmy. Autonomous and distributed recruitment and data collection framework for opportunistic sensing. *ACM SIGMOBILE Mobile Comp. and Comm. Rev.*, 16(4):50–53, 2013.

[39] A. Varga. OMNeT++. https://www.omnetpp.org/.

[40] K. Whitehouse et al. Hood: a neighborhood abstraction for sensor networks. In *Proc. of ACM Mobisys*, pages 99–110, 2004.

[41] J. Wiese et al. Are you close with me? are you nearby? In *Proc. of Ubicomp*, pages 197–206, 2011.

[42] H. Wirtz et al. Encrypting data to pervasive contexts. In *Proc. of IEEE PerCom*, pages 309–315, 2017.

[43] X. Zheng, D. Perry, and C. Julien. BraceForce: A middleware to enable sensing integration in mobile applications for novice programmers. In *Proc. of IEEE/ACM MobileSoft*, pages 8–17, 2014.