The Case for Crowd-Sourcing Communication *to* BLE-Only Devices

Alex Bellon*
UC San Diego

Tess Despres*
UC Berkeley

Prabal Dutta
UC Berkeley

Pat Pannuto
UC San Diego

Sylvia Ratnasamy
UC Berkeley

Abstract

With the widespread deployment of Bluetooth Low Energy (BLE) enabled devices, we are moving towards a world full of distributed, resource-constrained devices. With this growth comes the challenge of maintaining and communicating with millions of intermittently connected low-power BLE devices. While sending data from these BLE devices back to central servers has successfully been realized at scale (e.g. Apple AirTags), downlink communication is more difficult. Given the common usage of BLE-only devices in public infrastructure and other safety-critical environments, this inability to communicate with devices in a timely manner poses a barrier to realizing safe and secure ubiquitous computing. We explore the design space of providing downlink connectivity to BLE-only devices, give an overview of the challenges, and propose a system to enable downlink BLE communication.

CCS Concepts

• **Networks** → *Sensor networks*; **Mobile ad hoc networks**; **Wireless personal area networks**.

Keywords

BLE, downlink, opportunistic networks, firmware updates

ACM Reference Format:
Alex Bellon, Tess Despres, Prabal Dutta, Pat Pannuto, and Sylvia Ratnasamy. 2025. The Case for Crowd-Sourcing Communication *to* BLE-Only Devices. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3772356.3772394

*Both authors contributed equally to this research.

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772394

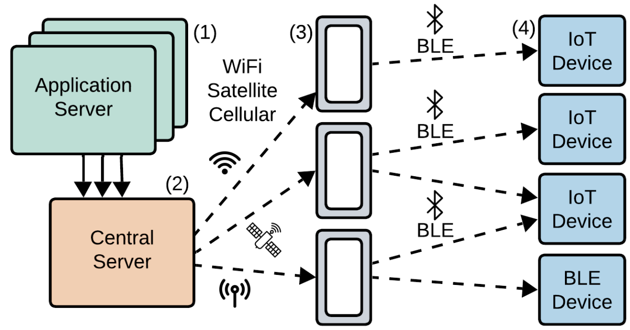


Figure 1: Our proposed approach uses phones to deliver traffic to BLE-only devices. Application servers (1) upload downlink "tasks" to a central server (2). When phones (3) detect a BLE-only device (4), they query the central server for a "task" and, if applicable, deliver the payload over BLE. Data backhaul travels in the opposite direction from IoT devices.

1 Introduction

Bluetooth Low Energy (BLE) devices are being deployed for increasingly diverse applications. In fact, projections predict that by 2027, billions of devices with BLE chipsets will be shipped annually [5]. Bluetooth Classic was initially developed for simple 1-to-1 connectivity between devices in close range, such as a headset or keyboard connected to a desktop computer [5]. Later, BLE, a distinct but related wireless protocol, was created for short data exchanges and low-power operation. While the 1-to-1 connectivity case is still common in the BLE ecosystem, recent applications like asset tracking have outgrown this domain in both scale and functionality.

Products like Tile [9] and Apple AirTags [1] show that it is possible to crowd-source the localization of BLE devices using a network of third-party phones, creating a 1-to-many system. However, a major limitation to deploying 1-to-many BLE systems is the current inability to seamlessly communicate back *to* BLE-only devices. Communicating with devices currently requires a trusted central device to physically travel to within radio range of the BLE-only device. For example, AirTags can only be updated and configured by another co-owned Apple device [3]. Adding to this challenge is the fact that BLE-only devices are not necessarily stationary (e.g. air quality sensors on buses), meaning they must first be located before communication is possible. Further, the status of BLE-only devices is unknown unless one is within BLE range, meaning a maintainer could spend (already limited) resources traveling to configure a BLE-only device, only to discover it is malfunctioning or does not need to be updated. To avoid these problems, many BLE devices are often further provisioned with power-hungry WiFi or cellular chipsets, and deployments are limited to areas with consistent coverage for these technologies. Ideally, BLE devices would not be bound to these expensive additions to provide connectivity and could function with BLE connectivity alone.

Providing downlink connectivity is crucial as deployment trends move beyond simple, personally-owned devices towards public infrastructure. For example, in a city-scale deployment of sensors, providing an update would require a visit to each device in order to send downlink traffic, a monumental task requiring considerable human and technical resources. *Smartphones*, provisioned with BLE, WiFi, and cellular chipsets, are well suited to fill the gap to provide downlink coverage for BLE-only devices, as illustrated in Figure 1. With the additional roll-out of satellite-connected phones that provide even wider connectivity coverage, this is the perfect opportunity to reconsider the role of phones in providing generalized read/write access to BLE-only devices [8].

Recent work has focused on extending 1-to-many BLE networks in the data backhaul use case, from edge device to cloud [10, 17, 18, 23]. However, there has been less exploration into the more complex case of providing connectivity in the opposite direction. The downlink case is distinct from backhaul, and generally challenging, because almost every piece of information relevant to routing is unknown a priori: the identity of the device to communicate with, its location and status, the data to send to the device, how long the device will be in BLE range, and more. In addition to determining all this information, this entire process, plus data transfer, must be performed without jeopardizing participant privacy. While the concept of using third-party phones to provide a crowd-sourced downlink network is intuitive, in practice there are a considerable number of challenges to overcome.

In this paper, we outline the design space for sending traffic to BLE-only devices, and make the case for a crowd-sourced approach. We then identify challenges that are distinct to crowd-sourced downlink, and show this approach is viable through empirical measurements and the construction of a prototype system. Finally, we highlight open research directions that must still be explored before downlink communication to BLE-only devices can be safely realized.

2 The Downlink Design Space

There are a few existing options for providing downlink data to BLE-only devices, including manually communicating, leveraging existing infrastructure, and deploying static gateways. We explore these options, and make the case for crowd-sourcing communication to BLE-only devices.

Manual Communication. The status quo for maintaining BLE-only devices is traveling to them to manually communicate with the devices. While this approach may hold up for dozens or even hundreds of devices, the increased scale of the now common "infrastructure deployments" threatens to make it infeasible. Consider a city that has 100,000 streetlight sensors with, on average, 500 sensors that need to be updated, fixed, or otherwise serviced every week[1]. Assuming this process takes 30 minutes of combined travel and update time, and operators cost \$100/hr on a fully-loaded basis, the city could expect to spend \$25,000 and 250 employee hours per week to update devices. Any increase in scale, distribution, update time, or update frequency can quickly skyrocket costs. From a financial, technical, and human resource perspective, the current maintenance process will buckle under such increased deployment scales.

Existing Infrastructure. A second approach could be to add BLE coverage to existing static infrastructure, such as cell towers. Assuming a city has a cellular tower every square mile, and there are no obstructions, it may be possible to form a BLE connection with highly optimized hardware, but with the tradeoff of increased power and decreased data rate [4]. To cover such a large area, physical layer modulation schemes must be adjusted to use Coded PHY LE 500K or even LE 125K, with data rates of 500 Kbps or 125 Kbps, respectively, significantly lower than the traditional 1 Mbps [4]. To form a connection and acknowledge data while transmitting at these longer ranges, transmit power on the BLE devices must be kept high (+20 dBm / 100 mW), which could quickly drain the batteries of BLE devices [4].

Deploying Infrastructure. Another option could be to deploy short range BLE infrastructure. Gateways could act as a bridge between local, short-range, low-power devices and longer range networks. While this approach might work for isolated, dense clusters of BLE devices in range of a single gateway, in cases where BLE devices are sparse, the added maintenance and powering of the gateways would undo many benefits from increased coverage. In other words, deploying new BLE gateways generally for this use case defeats the purpose of using BLE-only devices in the first place, in addition to requiring near duplication of existing cell tower and WiFi access point coverage.

[1]These numbers are an underestimate for a large city, see [11]

3 The Case for Crowd-Sourcing

We propose that a crowd-sourced coverage network could provide the answer. The crowd-sourced approach keeps the communication ranges of BLE-only devices small, reducing energy consumption and improving spatial reuse compared to longer range approaches. Additionally, using existing phones as mobile infrastructure bypasses the need to deploy expensive new infrastructure that must be maintained.

There is also precedent for this idea in the form of prior work on opportunistic networks which use cell phones or other devices as intermediaries. In 2003, the concept of "data mules" was introduced [19]: mobile entities which gather data *from* or send data *to* sparse sensors. Recently, we have seen a resurgence of modern commercial opportunistic networks. Helium [6] uses gateways deployed by individuals in their in homes for LoRa data backhaul; FindMy [3], Tile [9], and Samsung SmartThings [7] have leveraged this pattern for tracking networks, and Amazon Sidewalk [2] uses participating Amazon devices as gateways to extend in-home networks. A common issue with these backhaul networks is a lack of participant privacy [13], and in response, privacy-preserving data backhaul networks have been proposed [23].

4 Crowd-Sourced Downlink Challenges

We identify three challenges facing the design of crowd-sourced, downlink communication.

Status and Location of Devices. In an uplink system, all data is being transferred to a static, known entity, often through a stable cellular or WiFi connection. Conversely, in a downlink system, the location and status of devices are unknown until a gateway is within BLE range, when the gateway must request any relevant downlink data on the fly. Maintainers could be sent out to explicitly service a specific device, but this requires knowing the location of the device, something that becomes difficult in cases where the BLE-only device is itself mobile.

Volatile Connections. Since BLE connection ranges are smaller than WiFi and cellular, mobile gateways move in and out of communication range much faster [4]. This leads to shorter windows of time for data transmission, and increases the chance of packets dropping and devices leaving range before transfers are complete. As a result, robustness in the face of lost data or failed transfers is crucial for downlink BLE-only communication.

Participant Privacy. In an uplink system, gateways usually receive publicly broadcasted data from untrusted edge devices, then perform further processing before sending the data on to a trusted central service. However in a downlink system, gateways must establish connections with these untrusted edge devices in order to transfer downlink data. Additionally, downlink data is destined for specific devices, the locations of which may be known by a centralized service and could be used to deanonymize gateways that request that device's data. Overall, given the number of opportunities for identifying data to be leaked or inferred, implementing a successful downlink system requires careful preservation of participant privacy.

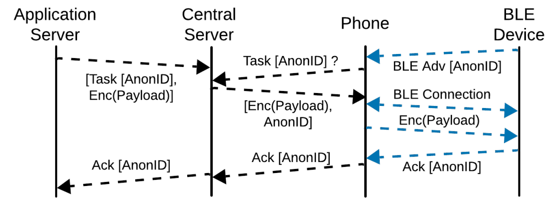


Figure 2: Our proposed process for sending downlink data. First, the application server uploads a task with an anonymized device ID and encrypted data payload to the central server. Later, if a phone receives a BLE advertisement with a valid device ID, the phone queries the central server, and if there is a task, sends the data to the BLE device.

5 Architecture Sketch

Figure 2 outlines the general process of sending downlink data in our prototype system architecture. Application developers and device operators can post "downlink tasks" to a centralized server, with updates or other downlink messages intended to be delivered to the BLE devices. These tasks list the anonymized IDs of the devices to communicate with and the content that needs to be delivered to the device. BLE devices regularly beacon their anonymized IDs in BLE advertisements. Passing phones determine which BLE devices are within range by detecting these advertisements, and query the centralized server to determine whether there is an active downlink task for the device. If there is an active downlink task, the phone can initiate a direct BLE connection with the device, query the server for the downlink data, and transfer the content to the BLE device. When the transfer is complete, the BLE device returns an ACK that the phone passes on to the central server to mark the task as finished.

5.1 System Actors

Devices in our downlink network architecture can be categorized into a handful of device classes.

Application Servers handle posting, disseminating, and updating downlink tasks, in addition to handling any optional reward mechanisms for successfully completing downlink

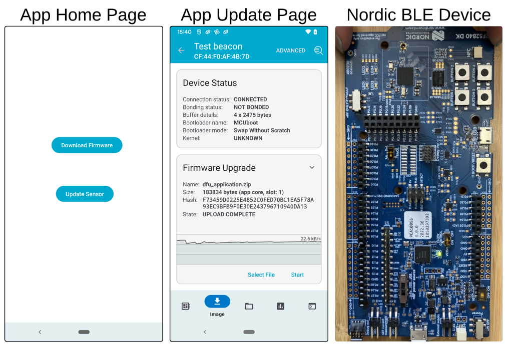


Figure 3: A prototype Android app and BLE device app to test the viability of crowd-sourced downlink. Our app builds on the Nordic device manager to query our central server and provide a firmware update to a BLE devkit. This provides a baseline to evaluate our design concept.

tasks. A particular application server is associated with at least one (and potentially many) BLE device(s). Traffic in the downlink network originates from the application server.

A single ***Central Service*** handles aggregating and routing traffic between application servers and phones in the network. The central service also authenticates edge devices, to confirm that they are valid participants in the network.

Phones in the network are any smartphones that (a) are running the required software to participate in the crowd-sourced downlink network and (b) have consented to participate. They pull the downlink data from the Central Service and transfer it over to the BLE devices. Phones have access to cellular networks and are battery powered.

BLE-Only Devices are distributed, low-power devices that are the target recipients for downlink communication in the system. BLE-only devices may be owned, maintained, and administrated by disparate parties, but can only be sent downlink data from a unique device administrator.[2]

6 Prototype Implementation

To test the viability of providing crowd-sourced downlink traffic to BLE-only devices, we developed a scaled-down prototype system. Our prototype implements the main functionalities: BLE devices beaconing their anonymized IDs, phones that can receive these BLE beacons and transfer downlink data, and a server to handle the creation, update, and completion of downlink tasks.

[2]For example, if the BLE devices are seismic sensors on bridges, they may be owned by the Department of Transportation, physically maintained by Contractors Inc., and administrated by Sensors, LLC. In this case, only Sensors, LLC would be authorized to send downlink data to the cameras, unless they choose to extend that permission to other groups.

6.1 Central Service

The central service is a Python server with an API endpoint exposed over HTTPS. We simulate application server interactions with the central Python server by sending HTTP requests. The API provides the following functionality:

`POST /upload_task` This allows a downlink task to be uploaded to the central server. The uploaded task is a JSON payload containing the encrypted downlink data, the ID of the destination BLE device, and a timestamp with the "expiration date" of the task. When the server receives the request, it adds an "uploaded" timestamp to the downlink task. The task is then stored in a database on the server.

`GET /get_task` Phones query this endpoint (and provide an ID) to check for any downlink messages for a given BLE device. If the ID matches that of a downlink task in the server's database, and the downlink task has not "expired", the server will return the appropriate encrypted downlink data. If there is no matching downlink task, or the downlink task has "expired", the server will return an error code.

`POST /complete_task` Phones use this endpoint to complete a downlink task, providing the completion token from the BLE device. The server then sends back the completion token to the simulated application server, where the application server confirms whether or not it is correct. If the token is validated by the application server, the downlink task is considered "completed," and is removed.

6.2 Phone

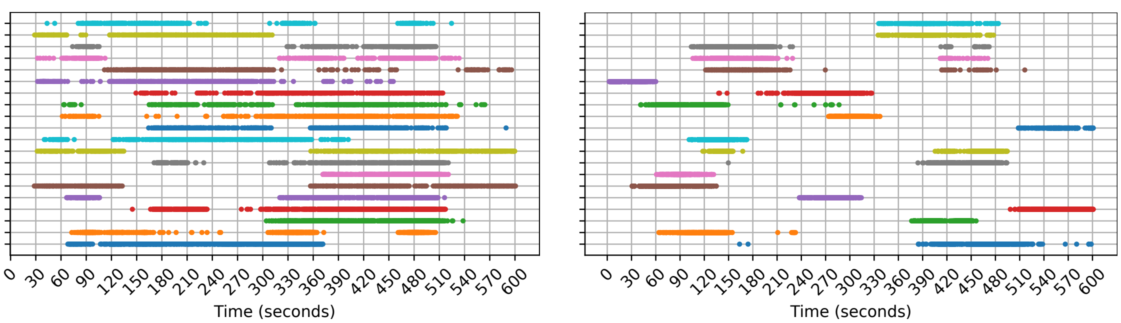
In the prototype system, phones receive BLE advertisements from BLE devices containing the device's ID, and then query the Central Server for any downlink communication for that ID. If there is a downlink task active for the BLE device's ID, the phone downloads it from the central server, initiates a BLE session with the BLE device, and transfers the downlink data. Finally, the phone receives and processes an acknowledgment if the downlink data is accepted by the BLE-only device. All of this functionality is handled by a custom Android application running on the phone.

6.3 BLE-Only Device

BLE devices broadcast advertisements containing their anonymized ID every 100 ms to 150 ms. They then "listen" for any BLE connection requests from phones with downlink data, and initiate a connection. The BLE device receives the downlink data from the phone, and then handles it according to the type of data (e.g., if firmware, install it). Once the BLE device has processed the downlink data, it sends an acknowledgment to the phone.

Table 1: Interaction times and devices seen across different measurements.

Measurement	Avg (s)	Med (s)	Min (s)	Max (s)	Devices Seen
`mall_sitting_0`	4.60	1.11	0.01	310.02	1262
`mall_sitting_1`	5.09	1.41	0.01	482.19	1459
`mall_walking_0`	4.59	1.26	0.02	121.86	747
`mall_walking_1`	4.98	1.50	0.02	429.01	963
`apt_sitting_0`	3.05	0.53	0.05	598.77	40
`apt_sitting_1`	2.95	0.52	0.02	598.92	38
`apt_walking_0`	2.07	0.53	0.01	89.63	888
`apt_walking_1`	1.48	0.45	0.01	80.15	578



(a) Walking in a busy mall **(b) Walking in an apartment complex**

Figure 4: BLE advertisements received over time in various environments. The top 20 most-seen devices for each experiment are shown here, each line is a different device (identified by MAC address). Many encounters exceed a minute, providing ample opportunities for data transfer.

7 Viability Measurements

The success of a crowd-sourced downlink system hinges on whether devices are around to carry downlink data and how quickly that data can be transferred. Phones may be within range of BLE devices for just a few milliseconds (e.g., a biker passing by) or hours at a time (e.g., someone sitting in a coffee shop). To determine whether these interactions can be leveraged to transfer useful amounts of data, we model how our system would behave in real-world environments.

First, we determine how often BLE devices are within range of one another. We use an ESP32 running a measurement program to collect data about nearby BLE devices, performing eight measurements in two different environments: a shopping mall (high traffic) and an apartment complex (medium traffic). In each location, we perform two measurements while stationary and two measurements while walking, to simulate static and mobile BLE device deployments respectively. For each measurement, we record BLE advertisement packets including the sender MAC address, RSSI, and timestamp of each packet.

We calculate the length of time that each device was within range of our BLE transmitter. We define "time in range" (TIR) as the period of time during which we regularly receive advertisement packets. Given an advertising interval of length X seconds for a device, we allow for up to $3X$ seconds to elapse without receiving an advertisement packet before considering an interaction ended, to account for possible interference and dropped packets in noisy environments. X is set to the most-frequently seen advertising interval for each device.[3] Devices may move in and out of BLE range of the ESP32 multiple times, but TIR resets every time a device leaves BLE range, as any data transfer would also have to reset. For the following analyses, we only consider interactions of non-zero duration.

The results of our measurements are shown in Table 1. We observe that in the "best case" (meaning longest interaction time), `mall_sitting_1`, the average TIR was 5.09 seconds, and in the worst case, `apt_walking_1`, the average TIR was 1.48 seconds. There was a minimal difference in TIR between walking and sitting in the mall (a crowded area), but around the apartment complex, walking reduced TIR by up to half. This may have been caused by an apartment complex containing more stationary devices, like appliances or personal

[3]If there are ties, we select the lowest value to err on the side of strictness.

devices that stay within the apartment, than in a mall, where most of the BLE devices are likely phones that move with their owners. If a BLE-only edge device were to move, phones may move with it and keep providing coverage, but stationary devices will not. Despite seeing roughly the same order of magnitude of devices between walking in the mall and walking in the apartment complex, the lack of devices moving with the ESP32 meant TIR fell by up to half. Overall, we see the best TIR when BLE edge devices are stationary and many devices are within range. However, with enough devices nearby, there is lower impact from having a mobile BLE edge device (as seen in Figure 4).

Our empirical measurements also indicate that this downlink system could facilitate useful amounts of data transfer with existing phone movement patterns. In our prototype system that uses BLE connections to transfer downlink data, we observe an average transfer speed of 21.5 kB/s. From the worst and best average TIR measurements of 1.48 s and 5.09 s respectively, this means anywhere from 31.84 kB to 109.44 kB of data can be transferred in the average interaction. If BLE advertisements are used to transfer data rather than connections, with 31 bytes of data per advertisement packet, advertising at the fastest interval of 0.02 seconds, between 2.29 kB to 7.89 kB can be transferred in the average interaction. Commonly used microcontrollers often have less than 1 MB of flash memory [15, 21, 22], so even a full firmware image would require only a handful of average-length interactions using connections. Small downlink data like new configuration values, minimal patches, or bug fixes could even be performed in a single interaction, either connection- or advertisement-based, especially if designs leveraged incremental updates, which we believe is an important capability [12].

8 Research Directions

While we have shown that a downlink BLE network is viable, there are many remaining research challenges to be solved.

Trust and Privacy. In our design scheme, we did not consider malicious parties, but a real-world system would need to handle misuse of the network. Internal actors may not all be trustworthy: an application server might attempt to send data without paying for its delivery, the central server might want to track network participants to sell location data, and phone users might want to get reimbursed for packets they did not deliver. Additionally, external actors may want to perform DOS attacks, or insert malicious traffic. To safely and responsibly deploy such a network, both privacy and security must be considered. To this end, in addition to our baseline prototype, we are working on adding authentication to various steps of the system, in addition to other features to cryptographically prevent forging.

Fragmentation and Reliability. While BLE interactions are often long enough to transfer *some* amount of data, they may not be long enough to transfer the full payload or continuous bytestreams. Therefore, protocols will need to support fragmentation to ensure progress towards eventual completion. Payloads should be able to be fragmented and transferred by multiple different phones, then reconstructed into the final payload at the BLE device. The BLE device must also have the ability to reorder and drop duplicate packets with limited resources.

Mesh and Out-of-Range. Crowd-sourced networks could also be extended to take advantage of BLE mesh technology. By sending packets along multiple hops, the range of BLE could be extended to span a group of people, for example. However, doing this would require implementing routing algorithms that allow data to traverse multiple phones, potentially taking inspiration from DTN routing [20, 24]. This is further complicated by the fact that phones might change locations once a path is established. One potential approach could be to take advantage of sensors such as accelerometers to determine if nodes are mobile or stationary [14]. Additionally, phones might not always have network coverage to reach the Central Service. This could provide an opportunity for opportunistically caching packets on device before interactions occur, much like DTNs advocated [16].

Incentives. Another challenge is providing an incentive system to encourage phone users to participate in a crowd-sourced downlink network. While the benefit of this system is obvious for the owners of BLE devices, there is little to incentivize phone users to sacrifice battery and cellular data to participate in the system. Quickly patching devices out in the world is valuable for applications, and we posit that device owners would likely be willing to pay a small amount for the transmission of downlink traffic. Additionally, we believe that having an incentive structure, where phone users are reimbursed (in some form) for sending data, will result in greater network participation. Providing this capability, however, is challenging, because a traditional scheme for paying a user leaks information about their location based on which devices they have visited.

Different Protocols. While this work focused on serving BLE-only devices, the system design does not hinge on any features unique to BLE. The exact networking protocols used to communicate between the central server, phones, and BLE-only devices could readily be swapped for different protocols, provided they can perform the same functionality. This same paradigm could be used to provide downlink communication to LoRa-only or Zigbee-only devices. Multiple protocols could even be combined to all communicate with and receive data from the same Central Service.

9 Conclusion

In this paper, we demonstrate that crowd-sourcing downlink communication is a promising path forward for connecting and maintaining BLE-only devices. We first highlight why crowd-sourcing connectivity from smartphones is a well-fit solution to the unique challenges posed by BLE-only devices. We then outline the design of a downlink system using smartphones to ferry data between application servers and BLE devices. We develop a prototype version of the system and gather preliminary measurements of transfer speeds between devices, showing that the existing deployment of BLE devices can support such a system. This shows that flipping the script on the successful model of crowd-sourced data retrieval is a viable way to provide downlink data to a whole class of previously isolated devices. However, many challenges remain before this is a safe, secure, usable, and private model for data dissemination.

Acknowledgements

This work was funded by a Qualcomm Innovation Fellowship. We also thank the HotNets anonymous reviewers and shepherd for useful feedback, along with the Lab11, Netsys, and SysNet lab members.

References

[1] 2025. AirTags. https://apple.com/airtag.
[2] 2025. Amazon Sidewalk. www.amazon.com/Amazon-Sidewalk.
[3] 2025. Apple FindMy. https://www.apple.com/icloud/find-my/.
[4] 2025. The Bluetooth Range Estimator. www.bluetooth.com/learn-about-bluetooth/key-attributes/range/.
[5] 2025. Bluetooth technology overview. www.bluetooth.com. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
[6] 2025. Helium. www.helium.com.
[7] 2025. Samsung SmartThings Find. https://smartthingsfind.samsung.com.
[8] 2025. Satellite on iPhone. https://support.apple.com/en-us/122339.
[9] 2025. Tile. https://tile.com.
[10] Alex Bellon, Alex Yen, and Pat Pannuto. 2023. TagAlong: Free, Wide-Area Data-Muling and Services. In *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*. 103–109.
[11] City of Los Angeles Bureau of Street Lighting. 2025. About. https://lalights.lacity.org/about/. Accessed: 2025-07-10.
[12] Andrea De Simone, Giovanna Turvani, and Fabrizio Riente. 2025. Incremental Firmware Update Over-the-Air for Low-Power IoT Devices over LoRaWAN. *arXiv preprint arXiv:2505.13764* (2025).
[13] Tess Despres, Shishir Patil, Alvin Tan, Jean-Luc Watson, and Prabal Dutta. 2022. Where the sidewalk ends: privacy of opportunistic backhaul. In *Proceedings of the 15th European Workshop on Systems Security*.
[14] Prabal Dutta and David Culler. 2009. Mobility changes everything in low-power wireless sensornets. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems* (Monte Verità, Switzerland) *(HotOS'09)*. USENIX Association, USA, 7.
[15] Espressif Systems. 2025. Products: SoCs. https://www.espressif.com/en/products/socs. Accessed: 2025-07-10.
[16] Kevin Fall. 2003. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Karlsruhe, Germany) *(SIGCOMM '03)*. Association for Computing Machinery, New York, NY, USA, 27–34. https://doi.org/10.1145/863955.863960
[17] Chung-Kyun Han, Archan Misra, and Shih-Fen Cheng. 2018. Mobility-driven BLE transmit-power adaptation for participatory data muling. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 962–971.
[18] Unkyu Park and John Heidemann. 2011. Data muling with mobile phones for sensornets. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. 162–175.
[19] Rahul C. Shah, S. Roy, S. C. Jain, and Waylon Brunette. 2003. Data MULEs: modeling a three-tier architecture for sparse sensor networks. *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.* (2003).
[20] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. 2005. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05*.
[21] STMicroelectronics. 2025. STM32F103 - Mainstream Arm Cortex-M3 MCU. https://www.st.com/en/microcontrollers-microprocessors/stm32f103.html. Accessed: 2025-07-10.
[22] Texas Instruments. 2025. MSP430 Microcontrollers Overview. https://www.ti.com/microcontrollers-mcus-processors/msp430-microcontrollers/overview.html. Accessed: 2025-07-10.
[23] Jean-Luc Watson, Tess Despres, Alvin Tan, Shishir G Patil, Prabal Dutta, and Raluca Ada Popa. 2024. Nebula: A Privacy-First Platform for Data Backhaul. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3184–3202.
[24] Zhensheng Zhang. 2006. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *IEEE Communications Surveys & Tutorials* 8 (2006), 24–37.