Make Way for Ducklings: Centering Data Files in Sensor Networks

Tess Despres
UC Berkeley
tdespres@berkeley.edu

Prabal Dutta
UC Berkeley
prabal@berkeley.edu

Sylvia Ratnasamy
UC Berkeley
sylvia@cs.berkeley.edu

Abstract

Remote sensor data can provide valuable insights. For example, data about animal migration patterns can inform conservation efforts, microclimate knowledge can be used to predict weather events, and city usage data can be used to inform infrastructure planning. However, networking sensor systems is an engineering hurdle that must be overcome to gain this crucial information at scale. We argue that to efficiently retrieve data, we should treat data files in sensor systems like the ducks in the classic story "Make Way for Ducklings" by Robert McCloskey. In the story, in order for the ducks to reunite their family, they must cross a bustling downtown Boston. Despite all the complexities of highways, bikes, and pedestrians, the ducks safely make their journey thanks to people clearing a path. In this paper, we argue that we must similarly "clear the path" for data retrieval in sensor systems and that doing so requires hiding the complexity of networking sensor platforms. Specifically, we argue that a desirable developer experience would be to simply write data to a local file on a sensor platform and have the files "magically" appear in a cloud storage location. To make progress towards this vision, we use the SD protocol as a narrow waist to decouple the network stack from sensor platform development. We call this system SDcloud and present a prototype SDcloud implementation as a proof-of-concept for networking sensors. We also discuss additional benefits that focusing on the data files in sensor networks can enable such as data processing applications.

CCS Concepts

• **Computer systems organization** → **Sensor networks**; *Real-time system architecture*; • **Hardware** → **Sensor devices and platforms**; **Wireless integrated network sensors**; *Networking hardware*; *External storage*; • **Networks** → **Sensor networks**.

Keywords

Sensor System, Wireless Networking, SD card, File System, Platform Development

ACM Reference Format:
Tess Despres, Prabal Dutta, and Sylvia Ratnasamy. 2025. Make Way for Ducklings: Centering Data Files in Sensor Networks. In *The 26th International Workshop on Mobile Computing Systems and Applications (HOTMOBILE '25), February 26–27, 2025, La Quinta, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3708468.3711883

HOTMOBILE '25, La Quinta, CA, USA

ACM ISBN 979-8-4007-1403-0/25/02
https://doi.org/10.1145/3708468.3711883

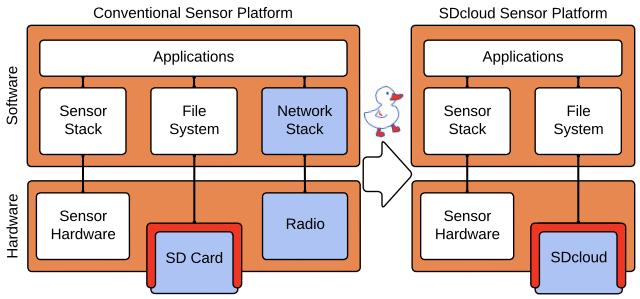

Figure 1: Conventional sensor platforms are typically built with a complex networking stack to reliably communicate to the cloud, in addition to any local storage. Instead, we propose decoupling sensor development from wireless networking by using the SD interface as a narrow waist. This allows developers to collect data and send it to the cloud without the need to explicitly invoke any networking functions, enabling the simpler SDcloud sensor platform.

1 Introduction

Distributed sensor systems can enable environmental monitoring and physical infrastructure monitoring at scale [5, 25, 32]. We have seen tremendous progress towards this goal with innovations in sensor capabilities and novel connectivity options [23, 26, 36]. Despite the promise of such sensor systems, building *sensor platforms* remains a tedious undertaking for sensor system developers. By sensor platform, we mean a combination of components (e.g., sensor, microcontroller, local storage, battery, and radio) which is physically co-located and used for sensing; multiple sensor platforms might all be part of a larger sensor system. We assume, in this paper, that sensor platforms can communicate, either directly or through a gateway, with the cloud to transfer data. Concrete examples of such sensor platforms today include wild fire monitoring [18] and smart city sensing [19].

Building sensor platforms is a challenging proposition for developers because it requires the integration of multiple components for applications with varying data volume requirements, infrastructure availability, and power constraints. Due to the diverse solution space for sensor platforms, sensors often require bespoke networks custom built for a single sensor application. The engineering effort to create custom solutions for each new sensor deployment can be prohibitive, leading to a world full of *network-isolated things* which require manual data retrieval.

We believe that the sensor platform development process would benefit from a focus on the essential task of gathering and analyzing the data, rather than network engineering. This leads us to the

question, *how can we easily retrieve data from distributed sensors?* We posit that from a developer's perspective, an easier experience would be to simply write data into a local file and have the file "magically" appear in a storage location in the cloud. Our key insight is that we can achieve this ideal if we move the network interface to reside *behind* the local storage, taking advantage of the ubiquitous SD standard. To do this, we propose SDcloud, shown in contrast to a typical sensor platform stack in Fig. 1, which uses the SD card standard as a common interface to network sensors. SDcloud interacts with the sensor at the physical layer by plugging into a standard microSD slot. The components inside of SDcloud, in addition to providing local storage, can transparently copy data over from the SD card to the cloud using an SDcloud network stack. This approach offers a familiar abstraction which uses the SD interface as a narrow waist between the sensor and the network.

We argue that the SD interface is a logical place to hide the networking complexity due to SD's mature standardization, and relatively ubiquitous support and use on embedded platforms [10, 15, 19]. Centering the data, in the form of a familiar and well supported file system, makes integrating networking into new sensor platforms easy. Developers can continue to write data to what appears to be a local SD card, with the same libraries that were used without networking. This helps prevent reimplementation because completely different sensor platforms could easily use the same network stack without writing code for a different microcontroller and application. Another benefit of this decoupling is ease of evolution in the networking stack. For example, we envision trying different technologies (e.g., LoRaWAN, WiFi, or even backscatter) could be as simple as swapping out SDcloud modules which support different networking stacks. The SD standard is also backwards compatible, allowing network isolated sensors deployed in the field to benefit from SDcloud by simply plugging in a module without reflashing firmware. Finally, SDcloud can provide a common platform for data processing functions such as filtering, or compressing data before sending, which minimizes the amount of time a power hungry transmitter is turned on.

In the rest of this paper, we motivate SDcloud by discussing the current process of building a sensor platform, highlighting the complexity involved and that SDcloud aims to avoid. We also introduce a proof-of-concept system, discuss applications and limitations to our system, along with open research directions following from SDcloud. We hope this paper sparks dialogue about how to make sensor platforms easier to deploy for a wide range of developers.

1.1 Developing a Sensor Platform

To concretely illustrate the process for developing a sensor platform, we can use the example of adding wireless networking to the popular Bosch BME688 sensor [7], an off-the-shelf, coin cell powered, air quality monitor. This sensor measures relative humidity, barometric pressure, ambient temperature, and gas concentration (VOC). The sensor comes in a compact 3 mm x 3 mm form factor. Bosch sells a development printed circuit board (PCB) populated with eight sensors for logging data and testing sensor configurations. The development board for this sensor does not include a wireless radio, so the first task when networking this sensor is to choose the physical radio technology that is appropriate for our environment and select an associated hardware chip to handle the networking and interface with the radio. For an indoor environment, Bluetooth Low Energy (BLE) and WiFi are both natural choices because existing infrastructure (e.g., routers and cell phones that can provide backhaul) is typically available. Bosch recommends using the Adafruit HUZZAH32 PCB [12] for networking and interfacing with their development board. The Adafruit HUZZAH32 PCB conveniently includes an ESP32 chip which supports both BLE and WiFi radios. Once the development hardware platform is selected, a BLE and/or a WiFi program must be developed. Without including any libraries or customization, BLE and WiFi applications on the ESP32 will be approximately 156 and 112 lines of embedded C code respectively, based on examples provided by Espressif [9]. Finally, a server application must be used to receive and store the data.

While this example requires knowledge of embedded development boards and the ability to write embedded C code, in many ways it illustrates a simple development process for a basic sensor platform. In this case, Bosch has a development hardware platform which connects to the Adafruit HUZZAH32 board out-of-the-box. Additionally, the chip on the Adafruit board, ESP32, is a well supported microcontroller with examples freely available and a robust developer community to provide support and resources. A more typical development path might require physically wiring devices together over a communication protocol such as I^2C or SPI and writing low-level code to interface with these protocols, or developing a custom PCB to interface between sensors, a microcontroller, and a radio. Research has shown that knowledge of embedded systems is a real barrier to setting up networking for crucial distributed sensor systems [22].

Yet one might still argue that buying components, building a small PCB, and programming a microcontroller is simple. However, we point out that this is not enough for a robust, real-world deployment. Many applications, such as asset tracking, include sensitive identifying information which should be encrypted both locally and in transit. Sensor networks can face intermittent connectivity, even when robust infrastructure exists, due to outages. To avoid losing data, systems should implement reliable transport to resend dropped data packets. Flexible connectivity options can also be extremely valuable for deployments. For example, with tightly coupled sensor and network stacks, moving a device from a yard with WiFi to a remote field, requires rewriting the embedded network code to support a long range communication technology such as LoRaWAN. The key to SDcloud is that instead of creating sensor platforms which couple network stacks with the sensor, we decouple sensor development from networking complexities by hiding the networking behind the SD card slot. This vision goes beyond simply replacing one hardware platform with another, because unlike other networked platforms, it is transparent to the sensor and requires no specialized embedded systems knowledge to set up. This efficiently clears the way to pipe data files to a remote file system, and provides a natural compute platform for handling networking, removing the need for sensor developers to deal with the complexities of networking, allowing them to focus on their unique sensing, power, and mechanical concerns.

2 Background and Related Work

In this section, we situate SDcloud with the existing capabilities of SD cards, various networks, and deployed sensor platforms.

2.1 SD Cards

SD cards are non-volatile flash memory devices widely used in embedded devices such as the ESP32 Thing Plus from Sparkfun and MKR Zero boards from Arduino [8, 10]. The small form factor of microSD cards, 15 mm x 11 mm x 1 mm, requires these boards to add only a small slot, which has developed into an industry standard [15]. During the development process, SD cards are often used to store sensor data locally. Data can then be manually retrieved and processed on a separate computer.

In considering potential strategies, we examined how SD cards are typically networked as I/O devices. The standards committee introduced SDIO in 2001 [16]. SDIO allows an I/O aware host device (i.e., a device that supports special SDIO commands) to communicate with another device such as a WiFi, BLE, or GPS module through the SD card physical interface. Since SDIO works as an extension to the normal SD card interface, hosts are able to use SDIO commands to control a range of external modules. SDIO also works with storage cards; combo cards incorporate both SDIO functionality and SD card storage in the same familiar form factor. However, to make use of the SDIO capabilities of a card, the host device (in the case of SDcloud the host would be the sensor) must be I/O aware, meaning it has to know how to send SDIO commands. The SD card must also be SDIO compatible because SDIO commands are a superset of the normally supported SD commands.

We take a different approach by supporting wireless connectivity with non-I/O aware hosts, such as sensors which are only capable of interacting with a normal SD card. This means that the sensor does not require any a priori knowledge of wireless capabilities to have data transferred using SDcloud. There have been a number of industry solutions which take advantage of the SD interface to network SD cards such as the Toshiba FlashAir and Eye-Fi, which were designed specifically to transfer files from cameras to a nearby laptop over WiFi [2]. 3D printers have also made use of networked SD cards to transfer files to the printer. These devices, while useful and an inspiration for SDcloud, are designed for a single purpose and do not support bidirectional communication or applications.

The SD protocol works by sending a six byte command to the SD card, which responds according to the command format. Commands include set mode, register reads, read block, and write block among others. Since this protocol is standardized, the commands and responses can work with multiple different digital communication modes including SD mode and SPI mode. The SPI interface is well supported on many microcontrollers, making it an ideal interface on which to build SDcloud.

2.2 Distributed File Systems

Distributed file systems, which are file systems distributed across multiple locations, can allow for remote access to files from any device [17, 21, 35]. SDcloud takes inspiration from distributed file systems by caching commands on the SDcloud hardware module, and providing location transparent access to the sensor client. However, distributed file systems solve a distinct systems challenge, in which multiple hosts access shared files which are distributed over many devices. In SDcloud, there is only one client which accesses files and one copy—the networked file copy.

2.3 Alternative Networks

There has been a resurgence of both commercial and research focused networks to support distributed sensors. For example, Amazon Sidewalk [1] extends Wi-Fi enabled Amazon devices to IoT devices in the surrounding area of a home, by using Amazon devices as gateways, Helium [3], known as "The People's Network" uses LoRa [4], to connect devices to gateways in people's homes, and Hubble Network [11] promises to connect off-the-shelf BLE devices to satellites. There have also been academic proposals to use people's phones to opportunistically backhaul sensor data [20, 33, 38]. These networks offer compelling and powerful communication capabilities at a lower cost point (i.e., in terms of power and price) compared to cellular IoT, but while these creative networks are exciting and offer new connectivity models, they are underutilized since connecting sensor devices to them is still difficult. One reason that connecting sensors is difficult is because of how tightly coupled the networking stack is with the sensor application. To integrate a sensor with these networks to create a platform, developers must ensure they have the appropriate physical radio for the network, and a microcontroller that is supported by the network. Even with the correct hardware and supported libraries, companies might require developers to register and provision sensors before they can be deployed [13].

2.4 Sensor Deployments

The research community is not new to deploying wireless sensor networks for applications ranging from tracking animal populations [25, 27] and monitoring ecosystems [28, 31], to sensing for city-based infrastructures [24, 37]. The ecology conservation community has cleverly embraced open source sensor platforms for deployments to address the prohibitive cost of commercial data loggers [29, 30]. A survey of 248 conservation technology users noted that while networked sensors had huge perceived potential to address environmental challenges, tool interoperability and scaling data analysis remained challenging [34]. Researchers working closely with communities to build sensor networks have noted that adding technologies, such as new types of wireless connectivity, can often be more effort than the benefits are worth, compared with field-tested methods such as manual data retrieval [22]. While SDcloud does not solve all of these problems, we believe that moving towards easily configurable, modular embedded platforms can help enable more affordable deployments with greater ease.

2.5 The Missing Link

There is a key missing link between unique and creative networks and bespoke sensor designs. Some firms have tried to address the gap in sensor network capabilities, by offering connectivity modules [6, 14]. However, these modules connect over physical layer protocols such as UART, SPI, or I^2C requiring developers to write low-level, embedded code on their sensor platforms to interface between the sensor and the connectivity module. While drivers might exist for these communication protocols, the developer must

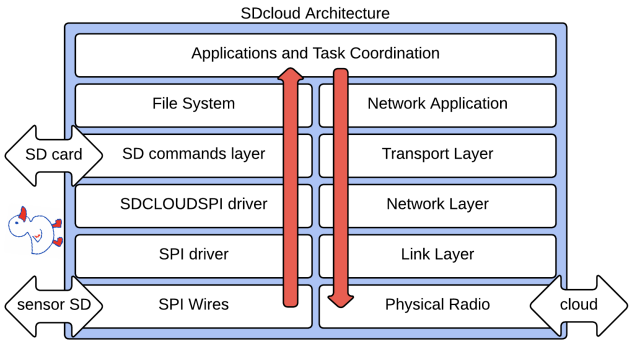

Figure 2: The SDcloud architecture relies on multiple layers between the physical interface with the sensor over the SD card slot and the physical radio which communicates with the cloud. The path that data travels from sensor to cloud is shown with vertical arrows.

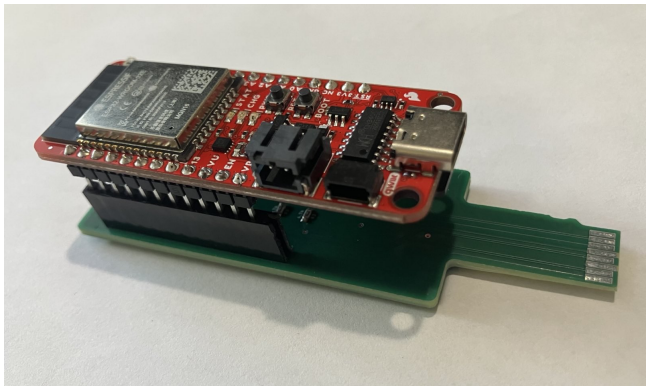

Figure 3: Our SDcloud prototype includes a custom baseboard PCB and an ESP32 Sparkfun Thing Plus where the SDcloud architecture is implemented in embedded C code using FreeRTOS.

still properly pack data into packets and send the data according to the specifications of the sensor connectivity module. We propose SDcloud can act as a bridge between these two paradigms.

3 System

In this section, we outline our first cut at a design of an SDcloud system along with the challenges we faced. We also discuss the details of our prototype implementation. Our architecture uses a microcontroller in between the sensor and a backup SD card to interface with the sensor over the SD protocol and manage wireless networking. The SDcloud architecture is shown in Fig. 2. We use a layered approach to intercept SD commands and data blocks using the SPI interface. We translate these electrical signals into SD commands and data blocks which are then passed to our onboard SD card. We periodically pull files from the SD card and upload them to a file server which can be accessed remotely.

We strategically chose to sit in the middle between the sensor and SD card as opposed to wiring the sensor directly to the SD card. Connecting the sensor directly to the SD card would limit SDcloud to either snooping on the wires to discover new data or periodically scanning the SD card. Snooping on the physical lines without being in the middle would not allow SDcloud to inject control commands or adjust the timing. Injecting commands is crucial to make sure that SDcloud can keep up with sensor traffic. Therefore, for more flexibility and configurability, we choose an architecture that places our microcontroller between the sensor, SD card, and physical radio. This allows SDcloud to simulate data processing while the microcontroller transfers data to the cloud, does computation, or communicates with the SD card all by holding the data line low. We can be dishonest to the sensor about the true capabilities of the SD card. For example, we clock our SPI communication to the SD card faster than the communication to the sensor. We could also hide a portion of the SD card from the sensor and reserve it for SDcloud applications as discussed in Sec. 4.

Since the SD protocol is highly standardized, the possible sensor behavior is bounded, but we also must respond appropriately to commands as they arise. This is the key challenge of SDcloud. SD card controllers are typically purpose-built chips embedded within the SD card. These chips interface with the SD host and the flash memory, keeping to a rigid timing schedule by responding correctly to SD commands within microseconds.

3.1 Implementation

We built our own PCB to act as a baseboard for the ESP32 Thing Plus and interface with the sensor SD card slot. This hardware prototype is shown in Fig. 3.

Having our own baseboard helps control for signal integrity issues that long wires could introduce and works with the standard microSD form factor of our sensor prototype (another ESP32 Sparkfun Thing Plus). For timing, command responses must occur within the next eight bytes after a command, as defined by the clock. Our sensor prototype clocks the communication with SDcloud at 400 kHz, meaning we have only 0.16 ms to respond. To meet this tight timing constraint, we pre-cache the initialization sequence, so that the command responses are ready to send to the sensor even before the command is received. For dynamic reads and writes, there are also timing constraints, which are not practical to pre-cache. Based on the SD protocol, reads and writes that do not recieve a response within 100 ms and 500 ms respectively, will fail. Sensors can implement longer timeouts, but must at least comply with the minimum set by the standard. We can take advantage of this timeout window to do SDcloud processing. For example, on a read or write, we tell the sensor to wait by holding the physical line low while we interact with the onboard SD card in the background to read or write the appropriate data.

There are many possible wireless interfaces which SDcloud could support, such as LoRa, BLE, WiFi, or even potentially cellular. For our prototype, we focused on WiFi using HTTP due to ease of development and ubiquity. This also allowed us to easily test our device in lab. We discuss adding other network stacks and the modularity of changing out network stacks in Sec. 4.

We used the ESP32 Thing Plus from Sparkfun [8] as the core of SDcloud, and we have two tasks running on the microcontroller.

The first task is a peripheral SPI interface which waits for and responds to commands from the sensor. The second task is a central SPI interface which sends commands to the SD card. These two tasks are pinned to separate cores and given high priority. The ESP32 also acts as a file server and writes files from the SD card using HTTP.

4 Limitations, Challenges, and Future Work

Currently, we know how the sensor behaves because it is implemented as another ESP32 Thing Plus development board which reads and writes fake sensor data files to the SD card. This allows us to make a number of simplifying assumptions, to validate the feasibility of SDcloud. For example, we have knowledge about the timing of commands that the sensor sends. We also restrict the clock rate of the sensor to 400 kHz, which is typically only used to probe the SD card during initialization, before it begins running at a frequency of 20 MHz or higher for data transfers. Meeting the timing requirements of the SD protocol is a key challenge. In the future, we would like to avoid artificially restricting the clock rate and dynamically respond to the sensor with the SDcloud capabilities. This would mimic how SD cards with different capabilities adjust the timing of the bus by telling the sensor their capabilities.

Additionally, we focused on only the SPI interface because it is commonly supported and used in embedded systems. The drawback of this approach is that not all sensors or SD host devices use SPI, some use SD mode. We plan to develop an SD mode driver to complement our SPI mode driver to parse signals appropriately based on the mode used by the sensor.

The SD protocol includes over 60 commands. In our prototype, we only support the commands that our sensor uses. We believe that these are the most common commands including read, write, and various register reads. These commands, though, may not cover a large enough number of devices. To address this we plan to implement responses for the minimum required set of commands for an actual SD card. Pre-caching commands is a good solution for the initialization sequence because it allows us to have responses immediately ready for the initialization sequence, but if the initialization sequence is not ordered as we expect, we could respond with incorrect byte sequences. We plan to address this by pre-building a dictionary of responses for possible commands that might be called during initialization. To help with this effort, we are in the process of doing a measurement study to discover what the SD command sequences are for a variety of different devices.

Our prototype exists on a board which supports WiFi and BLE. We have implemented connectivity over WiFi and plan to also implement BLE. Because flexibility is key to creating easy deployments, we would also like to support LoRa and cellular IoT. This will require integrating new radios into SDcloud.

Another key challenge in many sensor networks is power management. In our prototype, we do not address this challenge leaving it for future work. We imagine that SDcloud can be self-powered using a small battery so as not to drain the sensor battery, but we have yet to explore different power constraints and tradeoffs in our hardware prototype, especially since sensors may power off the SD card slot when it is not being used.

SDcloud could also store small applications on the SD card because we control both the interface to the sensor and the SD card. We imagine that these applications could be configurable on the server side, creating a control path from server to sensor, along with the data retrieval path from sensor to server. Applications could determine how frequently or under what conditions to send data. Since turning on the radio on a sensor node is a power hungry operation, doing data compression, filtering or anomaly detection on the node could provide important power and bandwidth savings. Applications could also potentially span multiple SDcloud sensor nodes and coordinate based on data shared between the nodes to decide if any data should be transferred or to share status information. Understanding the status of a sensor, and if it is active, can be reassuring and helpful even if data is not needed. Of course, multiple applications raise the specter of conflicting data accesses, which must be addressed.

Creating a lab prototype of SDcloud is just the first step. We plan to test our system in the field on network-isolated sensors, recognizing that new tools are only as useful as their end applications are to real people.

5 Conclusion

In this paper, we center sensor platform development around the SD protocol as a narrow waist interface. This reframes development and data retrieval to be about the data files themselves (i.e., the ducklings). We provide an architectural framework and prototype implementation of SDcloud. We hope that this effort can help make networking sensor systems easier for developers. More generally, we believe that creating easy-to-use sensor platforms should be a top level priority for the research community, and we hope this work generates more discussion on the topic.

Acknowledgments

Thank you to Lab11 and NetSys students and faculty for providing feedback on this paper. Thank you specifically to Paul de La Sayette and Jash Gujarathi for examining SD command sequences, and to Micah Murray for spending time rubberduck debugging. Finally, thank you to FabLab Neukölln in Berlin for providing temporary lab space and materials.

References

[1] 2023. Amazon Sidewalk. https://sidewalk.amazon.
[2] 2023. The best WiFi SD Cards of 2023. https://www.mbreviews.com/best-wifi-sd-card/#eye-fi-wifi-sd-card.
[3] 2023. Helium Network. https://www.helium.com/.
[4] 2023. LoRa Documentation. https://lora.readthedocs.io/.
[5] 2023. Purple Air. https://www.purpleair.com/.
[6] 2024. Blues. https://blues.com/.
[7] 2024. BME688 Sensor. https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme688/.
[8] 2024. ESP32 Thing Plus. https://learn.sparkfun.com/tutorials/esp32-thing-plus-usb-c-hookup-guide/.
[9] 2024. Espressif Github Examples. https://github.com/espressif/arduino-esp32/tree/master/libraries.
[10] 2024. Guide to Arduino and SD. https://docs.arduino.cc/learn/programming/sd-guide/.
[11] 2024. Hubble Network. https://hubblenetwork.com/.
[12] 2024. HUZZAH32. https://www.adafruit.com/product/3405.
[13] 2024. nRF Connect SDK - Amazon Sidewalk. https://docs.nordicsemi.com/bundle/sidewalk_2.6.0/page/index.html.
[14] 2024. Particle. https://www.particle.io/.
[15] 2024. SD protocol specifications. https://www.sdcard.org/downloads/pls/.

[16] 2024. SDIO/iSDIO. https://www.sdcard.org/developers/sd-standard-overview/sdio-isdio/.
[17] 2025. Gluster. https://www.gluster.org/.
[18] 2025. Gridware Gridscope. https://www.gridware.io/product.
[19] Joshua Adkins, Bradford Campbell, Branden Ghena, Neal Jackson, Pat Pannuto, Samuel Rohrer, and Prabal Dutta. 2017. The Signpost Platform for City-Scale Sensing. *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (2017), 188–199.
[20] Alex Bellon et al. 2023. TagAlong: Free, Wide-Area Data-Muling and Services. In *HotMobile*.
[21] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Symposium on Operating Systems Principles*.
[22] Eric Greenlee, Blaine Rothrock, Hyeonwook Kim, Ellen Zegura, and Josiah Hester. 2024. " The Devil You Know": Barriers and Opportunities for Co-Designing Microclimate Sensors. *ACM Journal on Computing and Sustainable Societies* (2024).
[23] Josiah D. Hester and Jacob M. Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (2017).
[24] Dhananjay Jagtap, Nishant Bhaskar, and Pat Pannuto. 2021. Century-scale smart infrastructure. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Ann Arbor, Michigan) *(HotOS '21)*. Association for Computing Machinery, New York, NY, USA, 72–78. doi:10.1145/3458336.3465275
[25] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel I. Rubenstein. 2002. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *ASPLOS X*.
[26] Zerina Kapetanovic, Shanti Garman, Dara Stotland, and Joshua R. Smith. 2023. Cosmic Backscatter: New Ways to Communicate via Modulated Noise. *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (2023).
[27] Roland Kays and Martin Wikelski. 2023. The Internet of Animals: what it is, what it could be. *Trends in Ecology & Evolution* 38, 9 (2023), 859–869.
[28] Alan M. Mainwaring, David E. Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. 2002. Wireless sensor networks for habitat monitoring. In *ACM International Conference on Wireless Sensor Networks and Applications*.
[29] James G Mickley, Timothy E Moore, Carl D Schlichting, Amber DeRobertis, Emilia N Pfisterer, and Robert Bagchi. 2019. Measuring microenvironments for global change: DIY environmental microcontroller units (EMUs). *Methods in Ecology and Evolution* 10, 4 (2019), 578–584.
[30] Lina K Mühlbauer, Giorgio Zavattoni, Risto Virtanen, Martin Grube, Bettina Weber, and Adam Thomas Clark. 2023. Arduinos in the wild: A novel, low-cost sensor network for high-resolution microclimate monitoring in remote ecosystems. *Ecological Solutions and Evidence* 4, 3 (2023), e12255.
[31] Michael R. Prior-Jones, Elizabeth A. Bagshaw, Jonathan Lees, Lindsay R. Clare, Steve G. Burrow, Mauro A. Werder, Nanna Bjørnholt Karlsson, Dorthe Dahl-Jensen, Thomas Russell Chudley, Poul Christoffersen, Jemma Louise Wadham, Samuel Huckerby Doyle, and Bryn Hubbard. 2020. Cryoegg: development and field trials of a wireless subglacial probe for deep, fast-moving ice. *Journal of Glaciology* (2020).
[32] Davide Sartori and Davide Brunelli. 2016. A smart sensor for precision agriculture powered by microbial fuel cells. In *2016 IEEE Sensors Applications Symposium (SAS)*. 1–6.
[33] Rahul C Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. 2003. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad hoc networks* 1, 2-3 (2003), 215–233.
[34] Talia Speaker, Stephanie O'Donnell, George Wittemyer, Brett Bruyere, Colby Loucks, Anthony Dancer, Marianne Carter, Eric Fegraus, Jonathan Palmer, Ellie Warren, et al. 2022. A global community-sourced assessment of the state of conservation technology. *Conservation Biology* 36, 3 (2022), e13871.
[35] Chandramohan A. Thekkath, Timothy P. Mann, and Edward K. F. Lee. 1997. Frangipani: a scalable distributed file system. *Proceedings of the sixteenth ACM symposium on Operating systems principles* (1997).
[36] Ambuj Varshney, Wenqing Yan, and Prabal Dutta. 2022. Judo: addressing the energy asymmetry of wireless embedded systems through tunnel diode based wireless transmitters. *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services* (2022).
[37] Sriranga Vishnu, S. R. Jino Ramson, S. Senith, Theodoros Anagnostopoulos, Adnan M. Abu-Mahfouz, Xiaozhe Fan, S. Srinivasan, and A. Alfred Kirubaraj. 2021. IoT-Enabled Solid Waste Management in Smart Cities. *Smart Cities* (2021).
[38] Jean-Luc Watson, Tess Despres, Alvin Tan, Shishir G. Patil, Prabal Dutta, and Raluca A. Popa. 2024. Nebula: A Privacy-First Platform for Data Backhaul. *2024 IEEE Symposium on Security and Privacy (SP)* (2024), 3184–3202.