Liberating the Siloed Gateway: Application-Agnostic Connectivity and Interaction for the Internet of Things

By

Thomas P. Zachariah

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Spring 2023

Liberating the Siloed Gateway:
Application-Agnostic Connectivity and Interaction for the Internet of Things

Abstract

Liberating the Siloed Gateway: Application-Agnostic Connectivity and Interaction for the Internet of Things

by

Thomas P. Zachariah

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Associate Professor Prabal Dutta, Chair

At its inception, the Internet of Things (IoT) was predicted to connect trillions of inexpensive smart devices to the Internet in mere decades, embedding paradigm-shifting utility, communication, and convenience into everyday things. Since then, a number of connected speakers, thermostats, doorbells, lights, and wearables have entered the market and our homes, but devices and sensors further constrained by processing power, memory, energy, and networking capabilities struggle to find their way to market. This is due, in part, to the prevailing model in industry of requiring a different application-layer gateway — often in the form of a device-specific router box or smartphone app — to link each unique IoT device to the Internet and allow users to interact with it. Unfortunately, this siloed approach cannot sustain the true scale, density, and capability of the IoT vision.

To break free from such siloed gateway standards which restrict scalability, affordability, and accessibility for constrained IoT devices, we propose a set of application-agnostic approaches to connectivity and interaction. For connectivity, we design a general-purpose network architecture consisting of ubiquitous short-range gateways that openly facilitate data transport for constrained devices to the Internet. By simplifying the gateway to its most essential parts, communications and processing, we can implement the architecture on inexpensive off-the-shelf system-on-chips and on existing smartphone infrastructure. These static and mobile approaches provide coverage for both reliable data throughput scenarios and ephemeral connectivity needs. For interaction, we design a browsing architecture that provides a seamless, scalable approach to discovering and interacting with nearby Things. The system takes advantage of multiple network patterns and modern web technologies to automatically detect devices and supply users with rich device interfaces that can enable interaction directly over local networking protocols. We implement and deploy a mobile-based browsing platform to facilitate this and provide tools for device developers. We further explore techniques for more physically-tied interaction, including extension to mixed reality.

The proposed approach to IoT connectivity and interaction opens up exciting possibilities for the future of the Internet of Things. By breaking down the barriers imposed by siloed gateway standards and introducing a set of approaches that form a general-purpose alternative, this work paves the way for the creation of more innovative and versatile IoT devices and interfaces. The design principles presented provide a simple, yet dynamic framework for enabling Internet connectivity for IoT devices, and for users to interact with them in a more intuitive and seamless way. These architectures represent significant steps forward in realizing the full potential of the Internet of Things, making it more accessible to everyone, and further advancing the development of the smart and connected world of the future.

For my family — especially my parents & siblings.

For my friends — past, present & future.

For my heart — Sybil.

For you.

Contents

List of Figures

List of Tables

Acknowledgments

The path to this dissertation has been a long and winding one, and I have been fortunate to have had the support of many people along the way.

I am immensely grateful for my advisor, Prabal Dutta, who encouraged pushing boundaries and allowed me to follow random tangents in my research pursuits. At the same time, he provided a deft guiding hand and a keen sensibility for the bigger picture when I would get lost in the weeds. And through many ups and downs over the years, both academic and personal, he has been incredibly helpful and understanding throughout.

I would like to thank the members of my dissertation committee: Edward Lee, Kimiko Ryokai, and Scott Shenker. They have graciously given their time and have provided invaluable feedback during the thesis process. I would also like to thank Shirley Salanio who has made life so much easier administratively during my transition to and time in Berkeley.

My interest in research was sparked in a high school course taught by Altair Maine, where we conducted independent studies in STEM. I was further encouraged to pursue this path by professors at Loyola Marymount University, including Barbara Marino, David Berube, Jeremy Pal, John Dionisio, and Gustavo Vejarano, as well as colleagues at Virginia Tech, Hyomin Kim and Kshitija Deshpande. At University of Michigan, Karem Sakallah and Mark Brehob were additional sources of inspiration and support. I am grateful to all of them for their encouragement and guidance.

In the midst of this wild journey, I somehow tripped and fell in love. Sybil has brought an immense amount of joy to my daily life and has been unjustifiably patient through this entire experience. I owe so much to her. The best part of reaching this milestone was completing it with her.

I would also love to express my unending gratitude to my family, who have all been my constant support during this endeavor, especially my parents — Anna and Thomas, my in-laws — Saramma and Jacob, and my siblings — Pushpa, Anu, and Solo (/Cait). These last few years have also seen the passing of three loving and wise women who helped raise me and foster my love of learning: Mariamma Zachariah, Mariamma Onattu, and Vimala Philip. I miss them dearly and am grateful to have experienced their radiant and nurturing presence in my life. I would also like to thank my adopted families in Michigan and Northern California — people in the community who, without question, took me in and treated me as family almost immediately when I arrived for grad school in both locations. I love you all.

By a set of bizarre, but welcome coincidences, two childhood friends happened to move, at the exact right times, into the two towns that I have had the pleasure of calling home during my graduate studies: Ke'ale Louie (Ann Arbor) and Rahim Ramzan (Berkeley). Once upon a time for a Spanish class project at one North Hollywood High School, we three would create a film consisting almost entirely of us being utter, unabashed fools dressed in pirate costumes. Somehow, we aced it. Apparently, years later, we foolish buccaneers would set sail together to new, far-off lands. Could not have picked finer roommates for this grand voyage. Gracias mis piratas.

I have encountered some truly incredible and brilliant people during my time in grad school. Walter Zarate, Ram Srivatsa, Kartik Joshi, and Salessawi Ferede were key figures of support and friendship at Michigan. We all met and naturally clicked in a computer architecture course, and would go on to form a project team named using the only acronym that seemed to work for us, Team TWRKS. In Lab11, I have had a number of collaborators who contributed to work that partly forms the basis for this dissertation, including Noah Klugman, Brad Campbell, Josh Adkins, Neal Jackson, Meghan Clark, and Branden Ghena. Additionally, as the fellow lab member relegated to the dungeons at Michigan (which was actually a pretty nice office) due to overflow, I got to share space with Rohit Ramesh, with whom many cherished hours were had whiteboarding complex problems and discussing the finer points of life, the universe, and everything. I spent copious amounts of time talking, eating, laughing, and travelling the world with the rest of lab, including Andreas Biri, Sam DeBruin, Bernhard Grosswindhager, Will Huang, Ben Kempke, Ye-Sheng Kuo, Pat Pannuto, Shishir Patil, Matt Podolsky, Ambuj Varshney, and Jean-Luc Watson. I am honored to have worked with and just messed around with all of them.

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the award number DE-EE0008220. This material is also based upon work partially supported by the National Science Foundation under grants grants CNS-1111541, CNS-1239031, CNS-1350967, and CNS-1824277, as well as by the NSF/Intel Partnership on CPS Security and Privacy under grant CNS-1822332. Finally, this work has been supported by generous gifts from Intel, Qualcomm, and Texas Instruments.

Part I

The Gateway Problem

Chapter 1

Introduction

The Internet of Things (IoT) is a growing network of physical items that are embedded with electronics, software, sensors, and connectivity. Examples of such devices like "smart" light bulbs and power meters are pictured in Figure 1.1. The Internet of Things represents the extension of the Internet into the physical world, connecting devices and enabling them to communicate with each other and with their users. The oft-cited goal of IoT is to create a more interconnected, automated, and data-driven world, where devices can be monitored, controlled, and optimized wirelessly, and where data can be analyzed to improve efficiency, productivity, and convenience in various domains, including home, healthcare, transportation, agriculture, and manufacturing.

Adapting computational resources to accommodate for such a wide array of IoT applications often presents a set of stringent operating requirements, such as a small form factor and long battery lifetime. This means the success of the Internet of Things relies heavily on the ability to connect devices that are constrained by processing power, energy, memory, and, most notably, networking capabilities. While we have seen an increasing number of smart devices such as thermostats, lights, and wearables on the market, there remains a need for more accessible and affordable devices that can further handle these constraints. If devices in the Internet of Things must operate under such limitations, a natural question arises: How do we connect Things to the Internet?

Mobile computers, including laptops, tablets, and smartphones, have experienced unparalleled success due in no small part to an abundance of wireless connectivity. Widespread Wi-Fi and cellular networks provide universal and transparent access to the Internet and cloud-powered applications. This has driven the success of mobile computing.

The coming wave of tiny, embedded, low-power, wireless, mobile, and wearable devices, however, does not currently enjoy the same level of ubiquitous and universal access to the Internet. Due to battery constraints and lifetime considerations, these devices tend to rely on low-power wireless communications like Bluetooth Low Energy (BLE), Zigbee, and Thread instead of more well-connected, but also more power intensive, Wi-Fi and 4G/5G cellular radios, despite their increasing ubiquity.

Figure 1.1: IoT Devices. Objects of a variety of form factors are embedded with an (often constrained) set of computational resources to create peripheral devices that enable "smart" capabilities in a variety of applications, such as lighting, power monitoring/control, personal health tracking, and environmental sensing.

The Internet of Things has grown in recent years, permeating the consumer and industrial market sectors. There are now over 20 billion devices connected to the Internet across the globe today [87]. This number, however, is less than half of the estimate industry leaders predicted for the anticipated IoT-infused market a decade ago [37, 46]. This outcome is due, in part, to the *gateway problem*.

1.1 The Gateway Problem

To connect to the Internet, resource-limited IoT devices require an *application layer gateway*—a system capable of translating data from the low-power link to the Internet at large. However, there remains a lack of affordable, application-agnostic infrastructure through which such devices could access the Internet. Current implementations of these low-power links do not provide an *Internet gateway*, but rather, as Figure 1.2 depicts, a narrow connection to a device-specific application that must be installed on a smartphone or laptop, or to a completely separate device-specific hardware router.

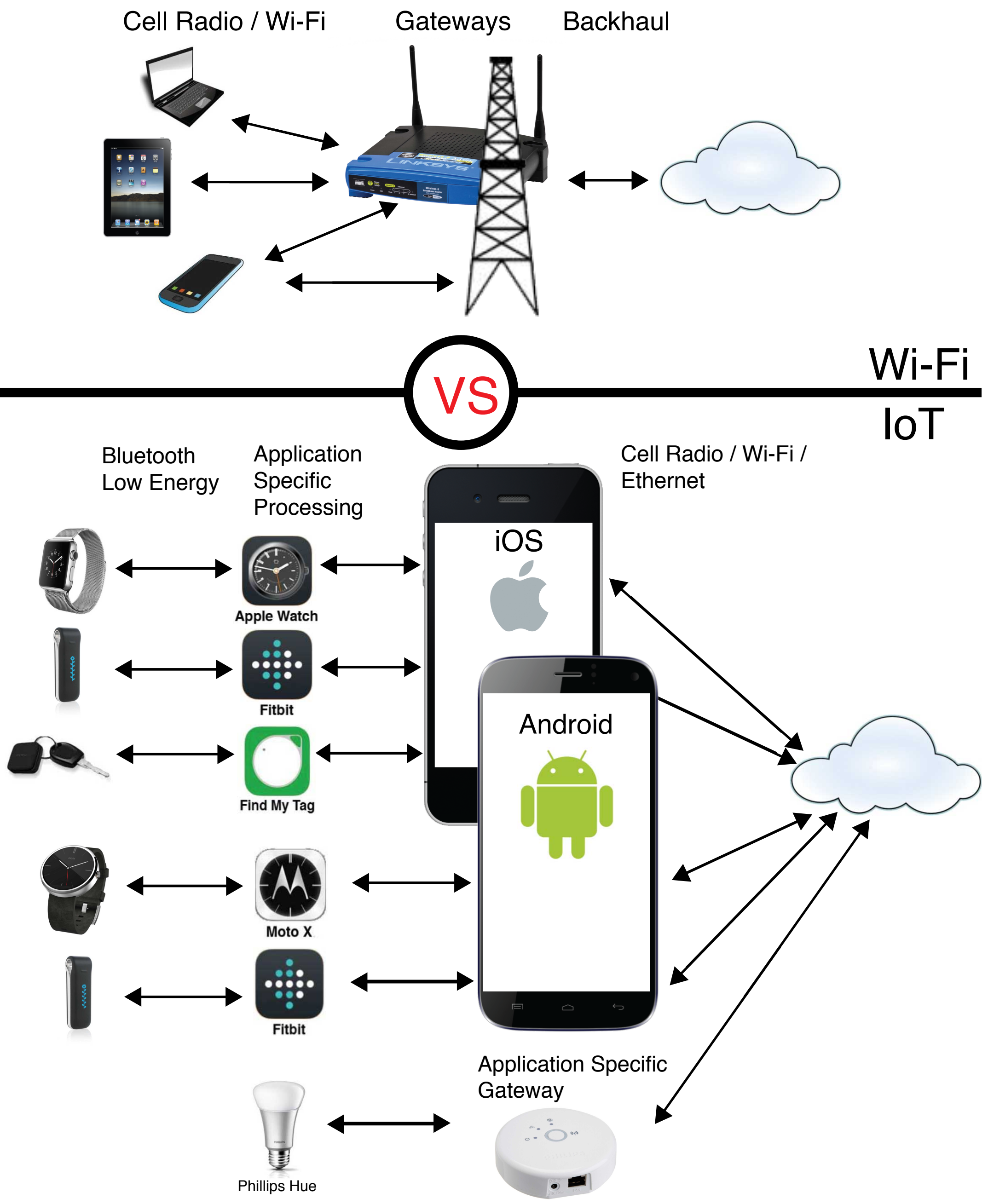
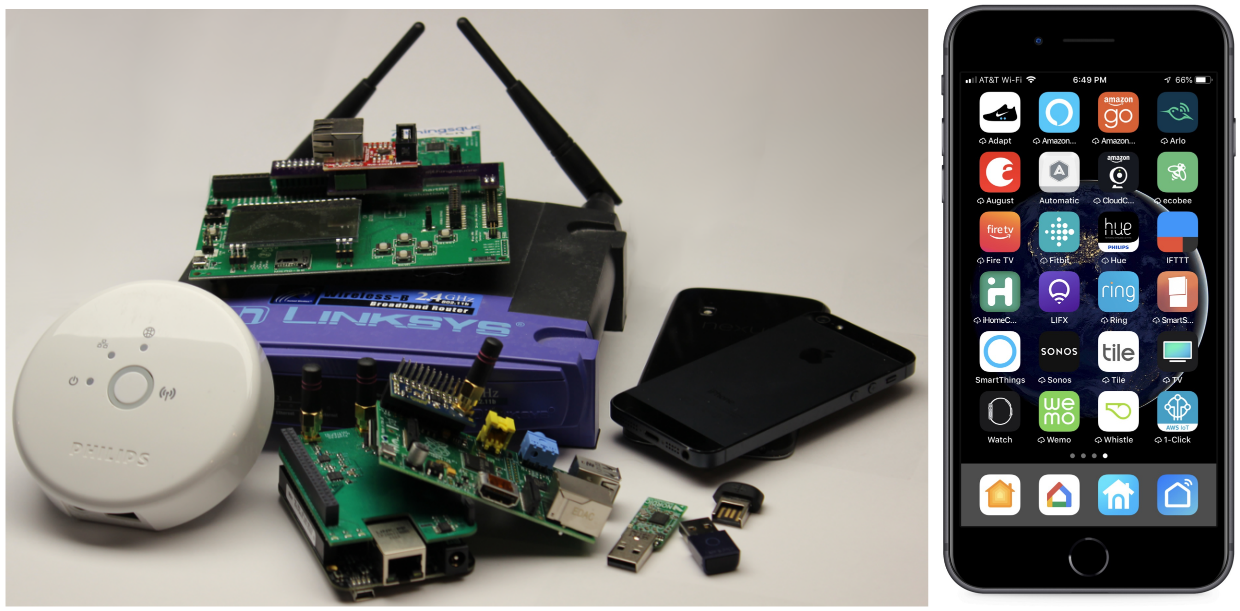


Figure 1.2: Wi-Fi Routers vs IoT Gateways. Currently, a separate physical router or smartphone application must be provided in order to enable gateway services for each type of IoT device deployed. This contrasts with any mobile computer's ability to connect to the Internet via a single Wi-Fi router.



(a) IoT Gateways **(b)** IoT Apps

Figure 1.3: Application-specific gateway approach. Currently, each of the devices in Figure 1.1 requires its own gateway as shown in (a) and/or an application like those shown on the smartphone in (b) in order to function. Each gateway in (a) and each application in (b) does not support more than a single type of device. Each gateway in (a) connects directly to the Internet through either a computer, Wi-Fi, or wired Ethernet connection.

The industry standard of using application-layer gateways both in software and hardware, only provides application-specific connectivity to the Internet—an issue especially for constrained IoT devices, as illustrated in Figure 1.4. Opening a new webpage on a laptop does not require a new application on the Wi-Fi router, but connecting a new IoT device does require a new smartphone app, a new laptop dongle, or a new basestation device, such as those seen in Figure 1.3. It is evident that gateways are the Achilles heel of low-power connectivity for the Internet of Things.

A number of factors have helped alleviate some of the past critical issues with IoT networks. For instance, alliances between several organizations across the industry have formed to agree on common standards for data protocols and user accessibility, limiting the number of custom solutions required [41, 67, 152]. Furthermore, popular cloud services and frameworks now facilitate orchestration and automation between multiple device classes, which are all assumed to have reliable network connectivity [18, 85]. The introduction of smart speakers presents an additional access point to several brands of devices in the consumer space through user-prompted controls and "scenes" [8, 19, 63].

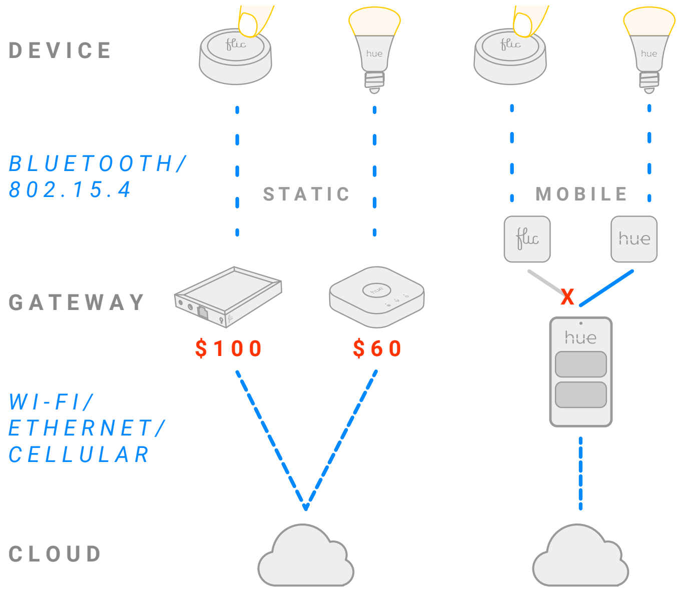


Figure 1.4: The Gateway Problem. Current state-of-the-art for reliable low-power IoT connectivity still typically incorporates a siloed, over-provisioned, and expensive stationary gateway for each brand or class of device. The other popular data transport mechanism is the use of device-specific apps on the owners' phones, which can opportunistically form a bridge to the Internet as a background process, but only through meager allowances of time—dictated by the OS—to receive, transmit and process data. [49, 132]

But devices—particularly those on the edge that rely on low-power operation—still often require their own custom infrastructure to establish connection to the Internet and to enable user interaction. Current application-specific hardware gateways are expensive one-off systems that implement different custom communication protocols for each device. The continuing standard of requiring users to obtain one for each brand of device — or even just installing a new app for each device — is untenable if the IoT market is to grow.

A worldwide collection of Internet-connected smartphones and inexpensive off-the-shelf system-on-chips (SoCs) may provide an unprecedented opportunity to deliver last-inch connectivity and interaction for the billions of IoT devices expected to emerge in the next few years, crucially, without requiring each phone to load every application-specific gateway app or each IoT device to have an expensive application-specific gateway router within proximity. On one end of the spectrum, networks built atop a few inexpensive SoCs could provide reliable Internet access to stationary sensors tasked with monitoring homes, offices, cities, or other areas without the need for device-specific gateway hardware. On the other, mobile or remote nodes could piggyback on passing smartphones to offload or receive data instead of requiring extensive mesh networks to relay data back to a set of dedicated Internet-connected gateways. Indeed, we have witnessed siloed versions of such approaches used with Fitbit [48], Apple Airtag [12], and Tile [153] devices.

1.2 Thesis Statement

We can use an application-agnostic approach to break free from siloed gateway standards which restrict scalability, affordability, and accessibility for constrained IoT devices by creating:

(1) general-purpose network architectures for connectivity to the Internet, and

(2) browsing platforms for seamless discovery of and interaction with Things.

1.3 Contributions

This work explores solutions for the major hindrances caused by siloed gateway standards which are impacting the growth of the Internet of Things, especially for the many envisioned applications and scenarios requiring constrained devices. The dissertation presents the design of open, application-agnostic architectures enabling connectivity and interaction for such IoT devices. The feasibility and utility of these designs is demonstrated with a series of implementations, often using inexpensive off-the-shelf components or leveraging existing mobile infrastructure. This document incorporates and references writing from a number of published works. Notably, the impetus for this work — the identification of the central issue discussed in this chapter — was originally presented in "The Internet of Things Has a Gateway Problem", published at the 16th International Workshop on Mobile Computing Systems and Applications (ACM HotMobile 2015) [172]. The following is a breakdown of the remaining chapters.

Chapter 2 presents a brief overview of the architectures proposed in this work. This includes design and implementation of general-purpose network architectures for data transfer and processing [169, 172], and browsing architectures for seamless device discovery and interaction [166, 168]. The chapter also covers some background on network protocols and topology, particularly with regard to Bluetooth Low Energy. This references material presented in "ReliaBLE: Towards Reliable Communication via Bluetooth Low Energy Advertisement Networks", published at the 2022 International Conference on Embedded Wireless Systems and Networks (EWSN 2022) [171].

Chapter 3 explores design and implementation of a general-purpose network architecture built atop inexpensive and ubiquitous off-the-shelf system-on-chips. This covers material presented in "The Internet of Things Still Has a Gateway Problem", published at the the 23rd International Workshop on Mobile Computing Systems and Applications (ACM HotMobile 2022) [169].

Chapter 4 explores design and implementation of a general-purpose network architecture leveraging existing mobile infrastructure via smartphone integration. This covers and expands on initial concepts presented in "The Internet of Things Has a Gateway Problem", published at ACM HotMobile 2015 [172].

Chapter 5 presents design, implementation, and deployment of a browsing platform for seamless discovery and interaction for IoT devices. This work is published as “Browsing the Web of Connectable Things” at the 2020 International Conference on Embedded Wireless Systems and Networks (EWSN 2020) [166].

Chapter 6 considers extension of the browsing model to mobile-based mixed reality. Part of this work is explored in “Browsing the Web of Things in Mobile Augmented Reality”, published at the 20th International Workshop on Mobile Computing Systems and Applications (ACM HotMobile 2019) [168].

Chapter 7 summarizes the implications of this work, discusses possible future directions, and concludes.

Chapter 2

Architectural Overview

To begin to break free from siloed gateway standards impacting constrained devices in the Internet of Things (IoT), we consider the functionality they provide. Each siloed gateway is effectively a re-implementation of the same core functions. The gateway is set up to discover and detect its own corresponding devices. It then collects data from the device over the low-power link, and possibly performs some local processing of that data. Additionally, when the gateway is implemented in a device-specific smartphone app, it has the ability to provide the user with a rich user interface that can enable interactions with the device by communicating directly over the low-power link. Finally, the gateway provides a bridge for communication to the Internet through its higher-power Wi-Fi, Ethernet, or cellular link, typically enabling data transfer to and from an associated cloud service.

In this work, we design an alternative set of architectures for device connectivity and interaction that can satisfactorily handle these core gateway functions using an application-agnostic approach.

For connectivity, we design a general-purpose network architecture consisting of ubiquitous short-range gateways that openly facilitate data transport for constrained devices to the Internet. In the design, we simplify the gateway to its most essential parts: communications and processing. This allows us to implement the architecture on inexpensive off-the-shelf system-on-chips (SoCs) and on existing smartphone infrastructure, providing coverage for both reliable data throughput scenarios and delay-tolerant/ephemeral connectivity needs.

For interaction, we design a browsing architecture that provides a seamless, scalable approach to discovering and interacting with nearby Things. The system takes advantage of multiple network patterns and modern web technologies to automatically detect devices and supply users with rich device interfaces that can enable interaction directly over local networking protocols. This serves as the basis for a mobile-based browsing platform and set of developer tools we implement and deploy. We also explore application of the approach in mixed reality for more physically-tied interaction.

The concepts of the architecture can be applied broadly to support several network technologies. In this work, we focus primarily on Bluetooth Low Energy (BLE), due to its popular use in constrained IoT devices and compatibility with mobile OSes.

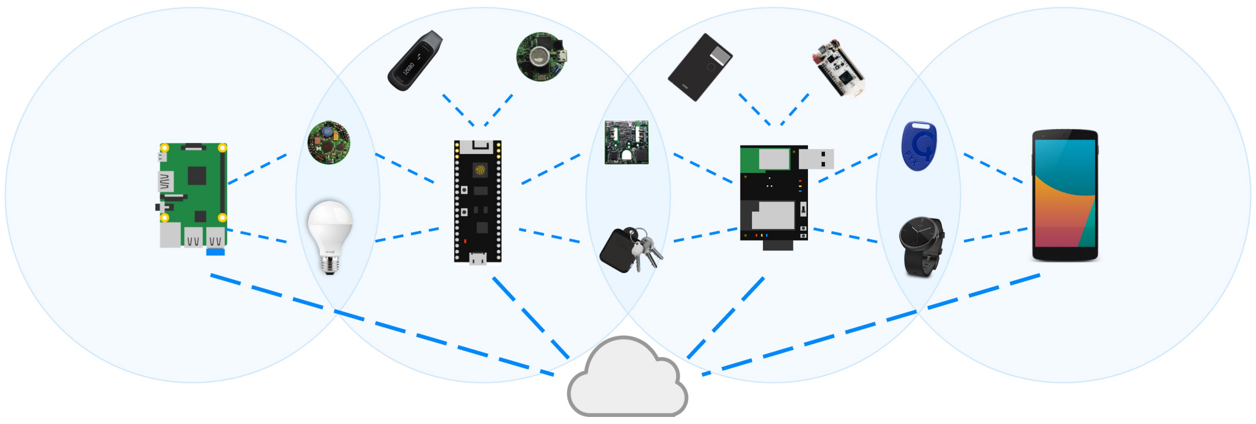


Figure 2.1: Gateway model for device connectivity. In the proposed architecture, we specify the use of simplified gateways that openly facilitate data transport for constrained devices to the Internet. By focusing design on the essential functions involving communications and processing, the model allows for gateway implementation on inexpensive off-the-shelf system-on-chips and on existing smartphone infrastructure, providing coverage for both reliable data throughput scenarios and ephemeral connectivity needs.

2.1 Connectivity

To enable connectivity, we specify the relationships and protocols linking three major components in the IoT network: the device, the gateway, and the cloud. A model of these links is depicted in Figure 2.1.

Devices would make themselves known to nearby gateways by sending periodic broadcasts over the low-power radio. These packets would contain, at minimum, some form of identifying device information. Additionally, they can include basic parameters specifying what type of gateway services the device would like to use and the location of the endpoint to communicate with. These broadcasts can also incorporate a small amount of the device's own service data, such as readings from a sensor.

Gateways would continuously listen for such broadcasts. When one is detected, the gateway can facilitate the services requested, which may include some form of data processing and one- or two-way communication with a cloud endpoint. The main transport mechanism we design is a simple proxy of the device's communication profile, translating the low-power link data/commands to an IP-compatible data structure when sending from the device to the cloud, and vice-versa when sending from cloud to device. We also discuss the technologies that can enable end-to-end IP-based communication, and the associated trade-offs.

Focusing on just the essential functions involving communications and processing opens up several implementation options for gateway platforms. We present a static approach using inexpensive off-the-shelf system-on-chips, which is capable of handling scenarios requiring reliable data throughput. We also present a mobile approach using existing smartphone infrastructure, which can provide support for remote and ephemeral connectivity needs.

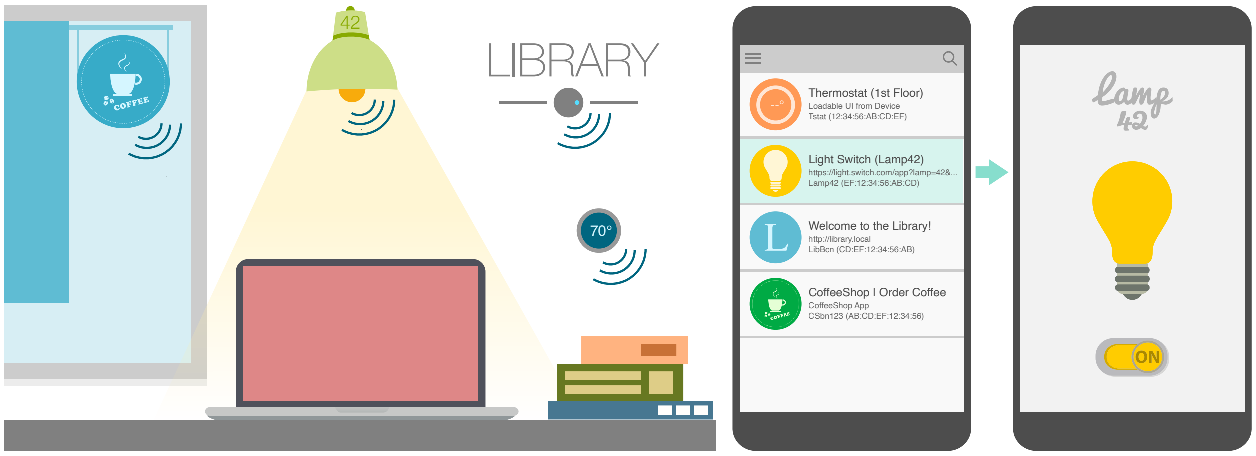

COFFEE
42
LIBRARY
70°
Thermostat (1st Floor)
Loadable UI from Device
Tstat (12:34:56:AB:CD:EF)
Light Switch (Lamp42)
https://light.switch.com/app?lamp=42&...
Lamp42 (EF:12:34:56:AB:CD)
Welcome to the Library!
http://library.local
LibBcn (CD:EF:12:34:56:AB)
CoffeeShop I Order Coffee
CoffeeShop App
CSbn123 (AB:CD:EF:12:34:56)
Lamp 42
ON

Figure 2.2: Browsing model for device interaction. Smart devices (light bulb, thermostat) and beacons make their presence known by broadcasting information about themselves. The browser, which can run on a smartphone, scans the local vicinity and displays to the user a selection of available user interfaces for nearby devices. When the user makes a selection, the interface for the chosen device is opened from a specified location hosted locally or online, presenting a user interface that can facilitate direct interaction with the device using browser-extended APIs for local network communication protocols.

2.2 Interaction

In the design of the browsing architecture, we extend the device broadcasting scheme to enable device discovery and interaction for the user. The illustration in Figure 2.2 depicts this model.

In this model, the device broadcast would typically include reference to a user interface location — most commonly a URL for a remote web page. This location can also be a page hosted on the local network or content stored on the device. If the phone detects that a corresponding native app is already installed, the browser can link to it as the default interface for the device. If no location is specified, the browser may be able to auto-generate an interface if the device implements known standard services.

By detecting these broadcasts, the browser can present interfaces for the nearby devices. When the user makes a selection, the interface for the chosen device is opened from the determined location. The interfaces can be built using standard web technologies, and the browser can extend APIs that enable communication over the phone's low-power radio to facilitate direct interaction with the device.

The standard implementation would present interfaces as a list. We also envision a version of the browser that displays the selection of interfaces in mixed reality to more physically tie the interface to the device. The browser may additionally extend APIs to allow interfaces to use mixed reality as a mode of interaction with their devices.

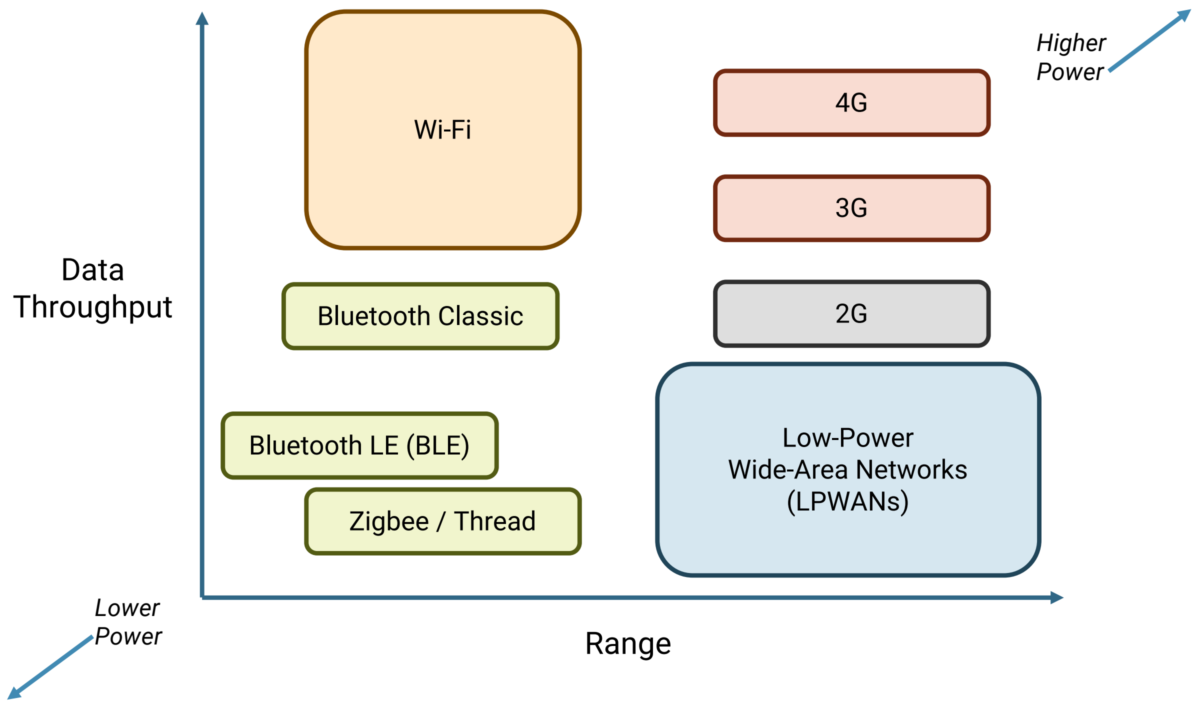


Figure 2.3: Comparison of radios technologies. The graph depicts relative trade-offs in data, power, and range between major wireless radio technologies [57]. Bluetooth Low Energy (BLE) is a popular choice in constrained IoT devices due to its low power benefits and its compatibility with smartphone platforms.

2.3 Network Technologies

Bluetooth Low Energy (BLE) is a wireless communication technology that enables short-range communication between devices with low power consumption, whose protocol was defined by the Bluetooth Special Interest Group in 2010 [30]. Figure 2.3 shows how BLE compares to other radio technologies. While BLE shares a name and some similarities with classic Bluetooth networks, it is a distinct protocol. It achieves lower power consumption through a number of mechanisms, including low duty cycle operation, shorter packet lengths, and reduced complexity in the protocol stack.

Low-Power Wide-Area Network (LPWAN) and 802.15.4 (Zigbee/Thread) radios are alternative low power link options that are suitable for long- and short-range applications, respectively [36, 152, 176]. These systems show significant promise for IoT applications, and are gaining traction within industry [41, 59]. The approaches presented in the work are motivated by aspects of each of their protocols, and would largely be applicable to these technologies. However, due to the high adoption rate of BLE in IoT devices and its ubiquitous presence in nearly all modern user-facing computers and smartphones, we choose to implement our approaches primarily using Bluetooth Low Energy.

BLE networks are typically implemented with a single-hop, star-topology, in which there exists a central device (the gateway, in our case) that scans for and connects to one or several peripheral devices. The device broadcasting schemes used in the connectivity and interaction architectures can be facilitated using the BLE advertisements — simple, periodic, broadcast messages. Our approaches can support networks of devices that exclusively use advertisements for communication needs, a choice that is sometimes made due to resource constraints [167, 171]. Higher throughput data transport can be achieved by entering a two-way connection mode with the peripheral device.

Part II

Connectivity

Chapter 3

Static Gateways

The gateway problem has created a gap in the availability of good quality, affordable bridging mechanisms for low-powered end devices in the Internet of Things (IoT). To begin to address this, we propose the use of ubiquitous, low-cost static gateways in scenarios and environments requiring long-term reliable throughput, especially with connection-less data transport for low-power devices.

In pursuit of this effort, we simplify the gateway to its most essential parts — communications and processing — and we consider inexpensive off-the-shelf components that might satisfactorily work to fulfill this role. Notably the ESP32, a $3 system-on-chip (SoC) which has recently grown in popularity, incorporates Wi-Fi and Bluetooth Low Energy (BLE) radios as a convenient single package with a built-in dual-core processor and memory [47].

We explore and optimize the design and performance of gateway functions using single and multi-microcontroller implementations, particularly for reliable opportunistic data forwarding, as well as support for application needs like simple device provisioning, high-rate two-way communication, and Internet Protocol (IP) compatibility.

3.1 Background & Related Work

Previous studies identify ESP32 as a desirable option for IoT applications, but focus on suitability as part of the end device [113]. While its relative low power is highlighted, its typical consumption is orders of magnitude higher than a standalone BLE SoC—the difference between lasting years on a battery instead of days [56, 90, 111]. ESP32 is, perhaps, better suited as a wall-powered IoT gateway.

For optimizing gateway performance, we look to previous work that investigates BLE IoT networks and provides strategies to set parameters for maximizing throughput [129], expand coverage with multiple gateways [57, 171], adapt to changes in the spectral environment [75, 110], efficiently filter and transform traffic at the network layer [115, 175], and create systems built on multi-SoC designs [58].

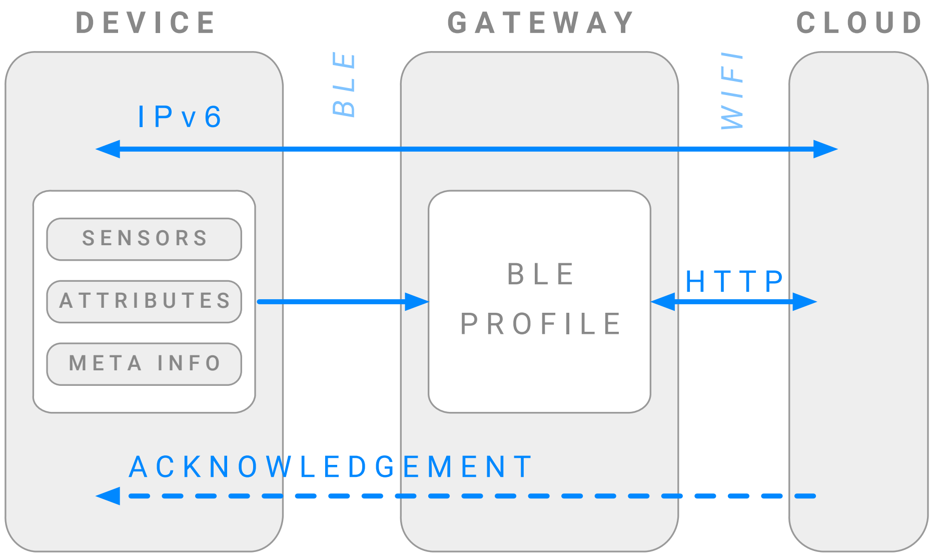


Figure 3.1: Architecture. We apply a data transport approach that considers two primary transmission mechanisms: (1) via IPv6, using the gateway as an IPv6 router and treating the peripheral as an IP-connected end host, and (2) via proxy, using the gateway to forward the device's BLE profile to the cloud.

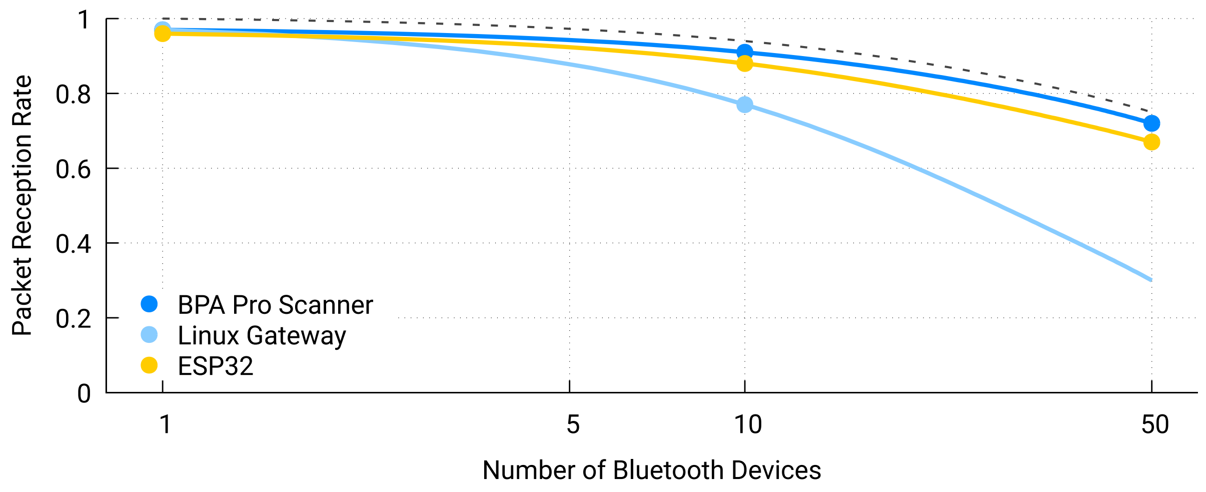
Examples of broader static gateway approaches exist, providing insight into the feasibility of supporting larger scale networks, but still use a closed, application-specific approach [9]. BLE mesh network techniques have been explored by academic, commercial, and standards organizations, including a strategy that builds a mesh network on top of BLE advertisements [42, 92]. A study on IPv6 over BLE explores how to establish IP-based connections with 4–8 Bluetooth devices through a central gateway system [149]. These approaches could provide support for wider coverage and IP compatibility in our gateway design.

3.2 Network Overview

To facilitate data transport between devices and the cloud on the simplified static gateway, we propose applying an approach that specifies two general and reusable techniques for transferring data that can support many applications over a system like ESP32. An overview of the architecture is shown in Figure 3.1.

3.2.1 BLE Profile Proxy

In this data transport mechanism, the gateway acts as a proxy for the information contained in the BLE data structures on the peripheral. At a high level, the gateway relays the advertisements, services, characteristics, and attributes shared with it from the BLE device to a remote endpoint in the cloud.



# Devices	ESP32	BPA(avg)	Linux
1	0.96052	0.96593	0.96696
10	0.88245	0.91357	0.77021
50	0.67458	0.71788	

Figure 3.2: BLE packet reception rates. A comparison between the PRRs of a $3 ESP32, a Linux-based gateway, and a professional $1000 Teledyne scanner (channel average) during 10 minutes of scanning with BLE devices sending unique packets every 100 ms. The dashed line is ideal reception accounting for probability of loss due to packet collision [171].

This is compatible with most existing BLE device setups, as the data organization between the peripheral and central nodes in current, application-specific BLE interaction will not fundamentally change. The approach supports opportunistic data forwarding for devices that primarily broadcast data via BLE advertisements—common for especially low power devices. Also, acknowledgements from the server can include commands to initiate connections and act as a proxy for two-way communication using standard BLE protocols.

3.2.2 End-to-End IPv6 Routing

The other data transport mechanism we consider is IPv6 packet transfer over BLE or Thread 6LoWPAN. This allows each IoT device to behave as any other IP end host and take advantage of the flexibility and convenience of working at the network layer. To support this, the peripheral and gateway devices must both include a 6LoWPAN network stack.

While 6LoWPAN specification exists for Bluetooth [123] and implementations of the network stack are available [149], it is still not yet commonly utilized by most BLE peripherals. 6LoWPAN sees perhaps greater utilization on 802.15.4 SoCs like the Nordic nRF82540 [126] using Thread [152].

State	Power	Current
Wi-Fi & BLE off	.19 W	37 mA
BLE scanning (w/ Wi-Fi idle)	.54 W	107 mA
HTTP POST (w/ BLE scan off)	.44 W	89 mA
HTTP POST (w/ BLE scan on)	.60 W	119 mA

Table 3.1: Power and current at various BLE and Wi-Fi states on ESP32.

The inclusion of both a BLE and Thread-capable SoC in a gateway device allows the flexibility to support two different 6LoWPAN networks. While we consider and incorporate designs to support this, we leave the implementation and evaluation of 6LoWPAN routing to future studies.

3.3 ESP32 Characteristics

ESP32 is an SoC from Espressif that includes Wi-Fi, BLE, two Xtensa LX6 microprocessor cores, and 520 kB internal SRAM, and can connect up to 64 MB external flash [47]. We explore the characteristics of the ESP32 to gauge its operational parameters for performing gateway operations.

3.3.1 Bluetooth Broadcast Packet Reception

To test the ESP32's ability to receive BLE advertisement data, we run a series of 10-minute scans in an isolated environment, with no external BLE interference. We compare with the average results of a professional $1000 Teledyne BPA scanner which runs on a single BLE channel at a time (3 total), as well as the results of a Linux-based gateway running Noble, a NodeJS Bluetooth scanning library. With devices each sending a unique packet every 100 ms, the ESP32 achieves a packet reception rate (PRR) ranging from 96% for a single device to 67% for 50, as shown in Figure 3.2. This performs better than the Linux-based gateway and is comparable to the average performance of dedicated BPA scanners.

3.3.2 Power Draw

We take power readings using a Drok USB meter [45] at different states while running a simple gateway application that performs an active BLE scan and sends raw data via HTTP request over Wi-Fi. The readings are shown in Table 3.1. Most consumer Wi-Fi routers require between 4–10 W in normal operation [151]. Under the same power constraints, about 8–20 ESP32 devices could continuously run the gateway application. This number of ESP32-based gateways is more than sufficient to achieve an amount of coverage similar to that of a consumer Wi-Fi router.

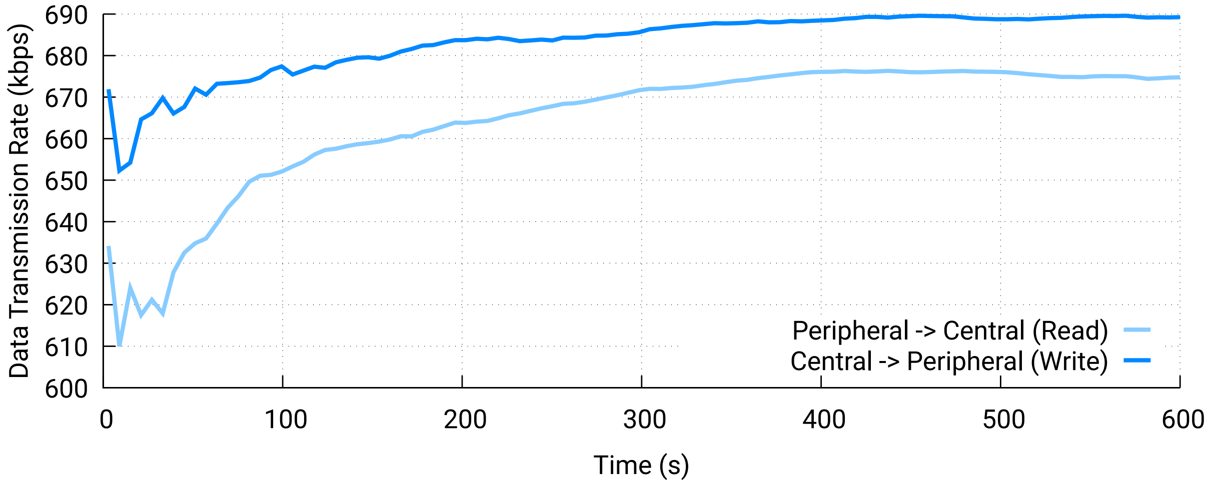


Figure 3.3: BLE read/write transmission rates on ESP32.

3.3.3 Bluetooth Connected Data Transport

To test read and write performance in a Bluetooth connection, we set up two ESP32s, with one in central role and the other as peripheral. Both are set to use the maximum transmission unit (MTU) of 517 bytes per read/write transaction. The central device writes data to the peripheral for 10 minutes. Then, in the 10 minutes that follow, the peripheral sends notifications to the central indicating it has data to be read. When the central receives each notification, it initiates a read.

In the test, the peripheral takes 770 ms to connect and configure itself, and the central takes 830 ms. The read/write transmission rate is plotted in Figure 3.3. The average transfer during read is 664 kbps, and write is 683 kbps.

3.3.4 Radio Coexistence

We note that the ESP32 makes a compromise to enable coexistence between its BLE and Wi-Fi radios. Both systems share the single on-board 2.4GHz radio module and antenna connection to perform their respective tasks. When both BLE and Wi-Fi are required by software, no one radio can continuously run for an extended period time. If an application runs both simultaneously, the ESP32 divvies utilization of the radio module between the two. While this still effectively facilitates connections and negotiated traffic quite reliably due to built-in delay tolerance, it proves inadequate in performing comprehensive retrieval of broadcast BLE data. When the Wi-Fi radio is running, the BLE radio only receives approximately 50% of advertisement packets with default settings. Radio priority can also be specified programatically when both radios are in use. Giving BLE priority increases PRR to around 66%.

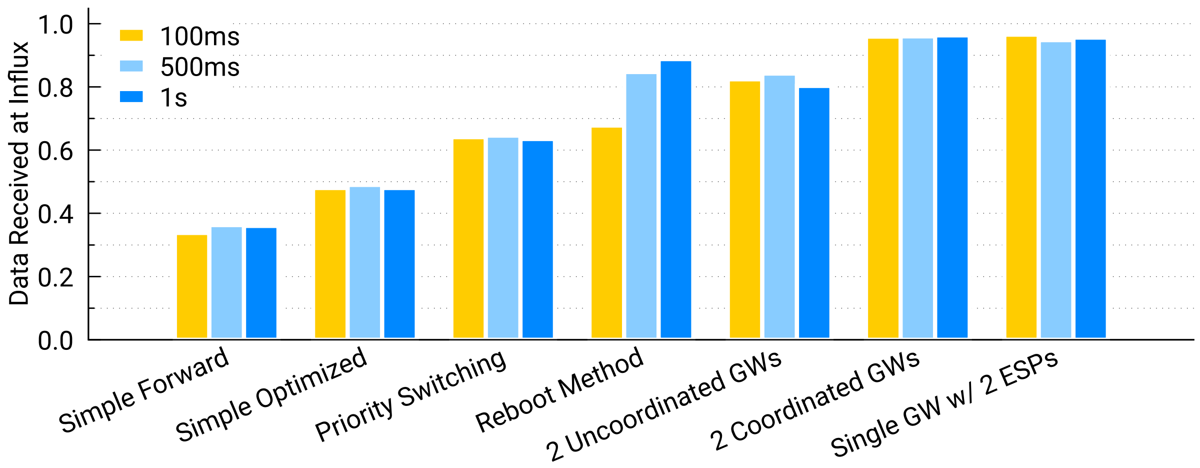


Figure 3.4: Forwarding performance with different approaches. Tested with unique advertisement packets sent every 100ms, 500ms, and 1s over 10 minute periods.

3.4 Static Gateway Analysis

We test gateway operations on the ESP32 with a deployment of low-power BLE devices and evaluate its performance. As a driving application, we explore and analyze services that facilitate data transport functions for PowerBlade plug-through power meters [43]. These low-footprint devices monitor loads plugged into power outlets and use Bluetooth to relay power measurements.

3.4.1 Forwarding

A key indication of gateway reliability for low-power devices is performance while forwarding data from advertised BLE packets to the Internet over Wi-Fi. We test with a basic implementation that forwards advertisement data to InfluxDB, a time-series database endpoint in the cloud [86].

The devices for these tests send unique advertisement packets at intervals of 1 s, 500 ms, and 100 ms for 10 minutes each. At these advertising rates, the test device effectively simulates broadcast of the once-per-second power measurement payloads from 1, 2, and 10 PowerBlade devices respectively.

We improve forwarding using various strategies that change how the BLE/Wi-Fi coexistence ultimately impacts data reception at the cloud. The performance of each method is shown in Figure 3.4.

3.4.1.1 Simple Forwarding

The gateway scans for BLE packets from the the test devices and sends parsed data to Influx via HTTP POST. Without modifying default configuration values, approximately 35% of packets are received at the cloud endpoint for all of the tested advertisement intervals.

3.4.1.2 Simple Optimized

Performance improves when some of the default configuration values are modified. We increase the scan window and interval to 100ms. Data is sent to the cloud in batches of up to 160 advertisement packets. A second of delay is added after any HTTP request to allow time for other background processes to take place, and reduce failures. This increases data reception performance to nearly 50%.

3.4.1.3 Priority Switching

The gateway can only achieve 50% because of the way coexistence of Wi-Fi and BLE, and sharing of radio hardware, is handled by the ESP32, as noted in Section 3.3.4. Programatically switching priority of the radios as needed improves reception to about 66%.

3.4.1.4 Reboot Method

As we seem to reach the limit of simultaneous radio performance with the priority switching technique, we consider an approach that instead handles BLE and Wi-Fi tasks sequentially so that neither cannibalizes the other's performance while running. The ESP32 performs a BLE scan at startup. It waits until the batch limit is reached to start up Wi-Fi, connect to the network, and send data. Then it reboots to deactivate Wi-Fi, restart BLE, and repeat the process. Because ESP32 does not load a bloated OS on boot, the reset is less than half a second.

At initial glance this seems to perform relatively well, particularly for fewer advertisements. However, as the frequency of advertisements increases, the gateway struggles to support the volume due to the scanning time lost during the switch. At 100 ms, it just barely keeps up with the performance of the priority switching method.

3.4.2 Multiple Forwarders

Next, we explore how coverage increases when multiple gateways receive data from the same peripherals.

3.4.2.1 Uncoordinated Gateways

We test the coverage of multiple gateways, using the priority switching method, without any facilitated coordination. For two gateways situated two feet apart with the test device a foot away from both, this uncoordinated approach yields about 80% reception. This setup, however, leads to recurring periods of redundant reception when both are receiving BLE data and—more worrisome—data loss when both are in Wi-Fi mode.

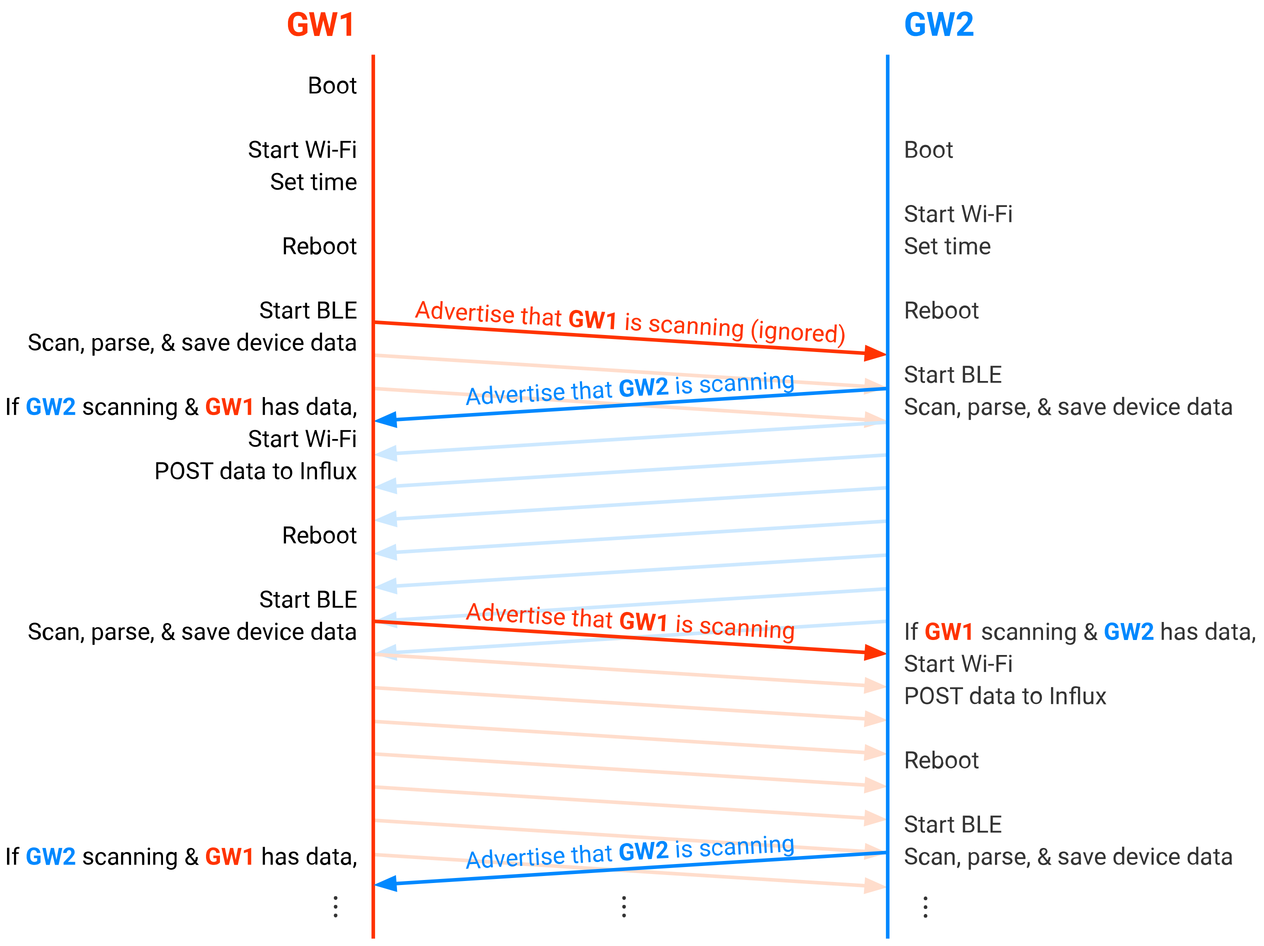

GW1
GW2
Boot
Start Wi-Fi
Set time
Reboot
Start BLE
Scan, parse, & save device data
If GW2 scanning & GW1 has data,
Start Wi-Fi
POST data to Influx
Reboot
Start BLE
Scan, parse, & save device data
If GW2 scanning & GW1 has data,
Advertise that GW1 is scanning (ignored)
Advertise that GW2 is scanning
Advertise that GW1 is scanning
Advertise that GW2 is scanning
Boot
Start Wi-Fi
Set time
Reboot
Start BLE
Scan, parse, & save device data
If GW1 scanning & GW2 has data,
Start Wi-Fi
POST data to Influx
Reboot
Start BLE
Scan, parse, & save device data

Figure 3.5: Timing diagram of the coordinated approach. Depicts communication between two gateways (GW1 & GW2).

3.4.2.2 Coordinated Gateways

In this approach, two ESP32s coordinate to alternately scan BLE packets from the test device and HTTP POST data to Influx. This extends from the reboot approach in which BLE tasks and Wi-Fi tasks are performed sequentially. Instead of waiting for batch limit to be reached, the BLE-scanning gateway waits for a packet from the other gateway that indicates that it has begun scanning. Once this signal is received the first gateway can halt scanning, startup the Wi-Fi radio, and send its data. It then reboots, restarts BLE scanning, and broadcasts a packet to inform the other gateway that its scanning has begun. Figure 3.5 depicts a timing diagram of this process. Using the same setup as before, this technique yields roughly 96% reception.

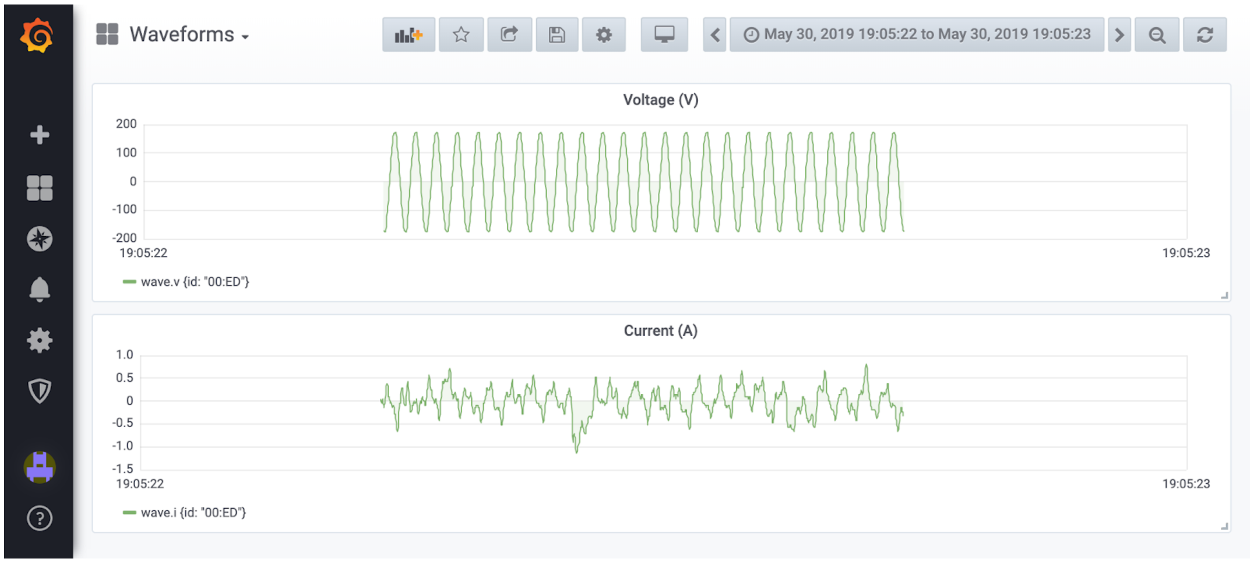


Figure 3.6: Voltage and current waveforms at Influx, viewed on Grafana. The ESP32 can connect to a PowerBlade, retrieve calibration values, and read 1260 values of voltage and change of current—representing 0.5s of data—from the raw sample service. The gateway then adjusts the data using the calibration figures, calculates the real-time current values, timestamps every point, and sends the final voltage and current waveform data to an Influx database via HTTP POST. The entire process takes around 82s.

3.4.3 Connecting

For occasional scenarios requiring larger rates of data transfer from the device or communication from the cloud to the device, the gateway can facilitate high-throughput two-way transmission via BLE connection proxy. We test collection of high-fidelity readings from BLE services on PowerBlade.

The PowerBlade device has a raw sample collection service which provides a half-second sample of values (1260 points) for voltage and change of current. The device also includes a calibration service that provides the constants to adjusts the data points to known units. To retrieve a sample, the ESP32 scans for and connects to the device. Once a connection is established, the gateway reads the values from PowerBlade's calibration service. Next, it requests data from the device's raw sample service by writing '1' to the startup characteristic. The ESP32 then reads a chunk of data from the data characteristic. It repeats these request and read operations for 10 iterations. The gateway then disconnects, timestamps and adjusts the data to proper voltage and current values, and sends the data to InfluxDB via HTTP POST in two chunks. This full operation takes about 82 seconds (60 s for BLE, 22 s for Wi-Fi). Figure 3.6 displays data received at InfluxDB using this method, viewed in graphical form on Grafana.

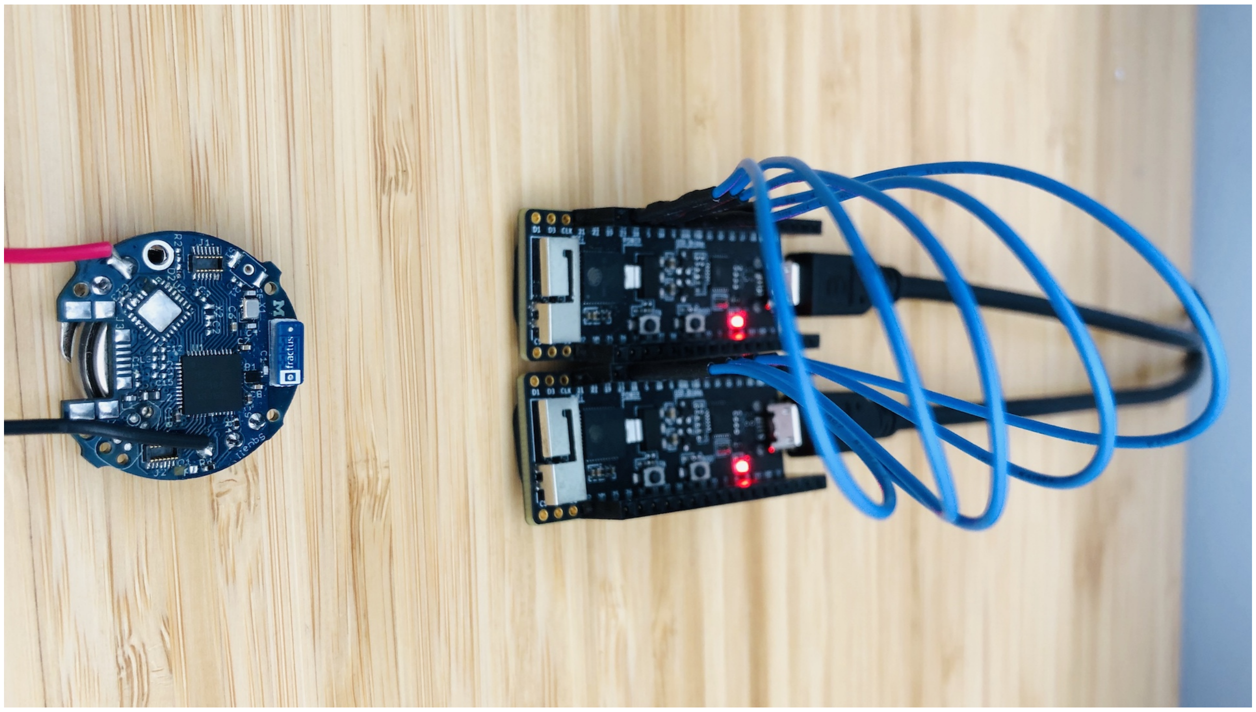


Figure 3.7: Dual-ESP gateway setup. Two ESP32 boards are connected via SPI. One ESP32 is dedicated to performing BLE scans, while the other solely sends the scanned data over Wi-Fi. This effectively yields near-optimal forwarding performance, roughly equivalent to the BLE scanner's PRR.

3.4.4 Multi-SoC Gateway

The promising results of the coordinated forwarding approach indicates that distributing BLE and Wi-Fi roles to dedicated SoCs improves performance. As a result, we explore creation of a single gateway from multiple SoCs. Table 3.2 compares the single- and multi-SoC setups.

3.4.4.1 Dual-ESP Gateway

This approach uses two dedicated ESP32s that communicate via SPI to operate as a single gateway. The BLE-focused ESP scans for BLE packets from peripherals. The Wi-Fi-focused ESP forwards parsed data over Wi-Fi to an Influx database. This yields a reception rate of about 96% while also avoiding scaling issues that occur in denser environments with the reboot method. The setup is pictured in Figure 3.7.

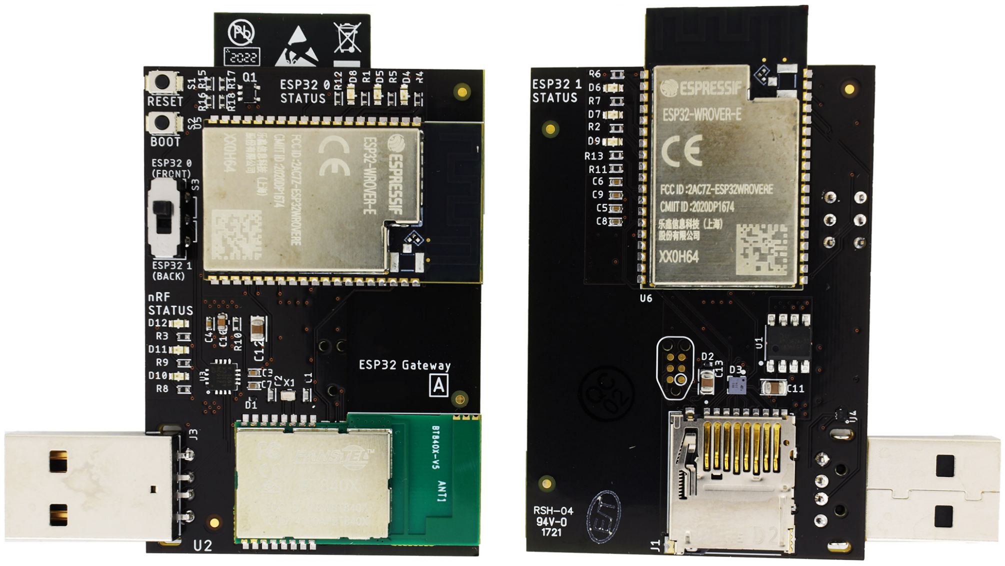


Figure 3.8: Gateway hardware. Our design distributes BLE, Wi-Fi, and 802.15.4 roles between two ESP32s and an nRF52840.

3.4.4.2 ESPxNRF Gateway

Based on these results, we design a custom high-reception gateway system, as shown in Figure 3.8. Using modularization to optimize performance and distribute gateway roles, the design houses two ESP32 modules, SD storage, and a Nordic nRF52840 — a supplemental chipset with Bluetooth Low Energy and 802.15.4 [126]. Using the two ESP32s for forwarding performs the same as the dual-ESP setup. Inclusion of 802.15.4 opens the door for Thread-based applications, including IPv6 connectivity [67, 152].

3.5 Discussion

In this section, we discuss lingering research questions and trade-offs that should be more deeply explored when considering use of the low-cost static gateway approach outlined in this chapter.

3.5.1 Design

Our gateway analysis reviews a range of techniques for facilitating communication between device and cloud, each of which present a set of trade-offs. Table 3.2 presents a tabulated comparison. The initial forwarding technique that runs on a single ESP32 is capable of receiving roughly two-thirds of broadcast advertisements from devices within moderate range. This is likely suitable enough in more delay-tolerant deployments which only require pings at frequencies of minutes, hours, or days. The multiple forwarder and multi-SoC setups are de-

SoCs	Price	Approach
1x ESP32	<\$5	Shared BLE & Wi-Fi
2x ESP32	<\$10	Distributed BLE & Wi-Fi
2x ESP32 + 1x nRF52840	<\$20	Distributed BLE, Wi-Fi, 802.15.4

Table 3.2: Single & multi-SoC gateway approaches.

signed to handle more frequent forwarding in more dense deployments. Our hardware design is driven by the needs of the PowerBlade deployment, which produces frequent output from devices at potentially every outlet in a household. These setups remain relatively low cost and low power even when multiple SoCs are used. Though it may be limited in resources, it runs on a simpler processing loop, and is capable of quickly recovering from crashes as it does not need to load a bloated OS every time it starts. The benefits for the low-cost static approach may fall off, however, when deployments demand highly-responsive, large-volume throughput. But such demands are often excessive for low-power IoT device deployment scenarios.

3.5.2 Security

When BLE devices broadcast data in advertisements as they do in the forwarding approach, that data is accessible to any scanning BLE device within range. It is important to consider this inherent risk in any deployment of BLE devices, as advertisements are a key component of the Bluetooth protocol. At minimum, sensitive data can be encrypted by the device when broadcast and translated by the trusted cloud endpoint or the gateway itself, if provisioned properly for it. Alternatively, devices can broadcast requests for the gateway to establish secure BLE connections if larger amounts of sensitive data need to be sent or if a more secure transaction needs to be facilitated via profile proxy between the device and cloud.

3.5.3 Industry

It is understandable why the prevailing IoT approach adopted by industry has favored expensive brand-specific gateways. At face value, it makes sense as a short-term business decision as a means for a manufacturer to generate additional revenue, guarantee buy-in by its users, and control the full pipeline between device and cloud. However, the cracks in this approach grow more apparent — made evident by the relative stagnation of consumer IoT markets and pushes for standardization between brands. The low-cost static gateway approach reduces the barrier to user entry, which reduces the barrier to device deployment, connectivity, and orchestration. One potential way to support an industry transition to this approach, is implementing a gateway-as-a-service which could run many virtual brand-specific gateways on a single physical gateway.

3.6 Summary

The gateway is a major pain-point in current state-of-the-art IoT architectures, particularly with achieving reliable data transport for resource-constrained edge devices. We suggest an approach that anchors networking infrastructure for such systems on low-cost, pared-down open static gateways. We first test the approach on a standalone ESP32 BLE/Wi-Fi SoC, and fine-tune to reduce contention and improve performance, particularly for connection-less data forwarding scenarios in densely populated environments. For high-reliability scenarios we develop a custom gateway design which distributes gateway tasks among two ESP32 modules and an additional BLE/802.15.4 SoC. These setups can proxy as an Internet-connected BLE profile or translate to IP using 6LoWPAN. If deployed widely in requisite environments, our approach could provide inexpensive and reliable connectivity for a host of devices in a currently-neglected category of constrained and low-power systems, perhaps reigniting the growth of a more densely populated and useful Internet of Things.

Chapter 4

Mobile Gateways

To expand network access for low power IoT devices beyond the bounds of static coverage, we consider the established global network of well-connected smartphones, which may provide a promising foundation for ubiquitous, low-power, last-inch networking, particularly when more ephemeral and delay-tolerant communication suffices. An example of such a model is depicted in Figure 4.1. It is already common practice in industry to facilitate configuration and administration of devices like smart home appliances in smartphone apps. However, this siloed, segmented, and application-specific approach creates walled gardens that drastically limit the wireless connectivity potential for the substantive class of constrained IoT devices.

Addressing this problem requires a networking architecture for low-power wireless devices that better leverages the opportunities provided by mobile infrastructure. Such an architecture would need to provide convenient and transparent access to the Internet for low-power devices while offering data integrity, security, throughput, and lifetime for the phone and device.

The proposed approach uses Bluetooth Low Energy (BLE), common on modern smartphones, as the primary link between low-power peripherals and capable smartphones. In contrast to the application-specific design of device-phone interactions, however, we extend the open gateway model introduced in the previous chapter. First, we envision that any BLE device could leverage any smartphone as a temporary Internet Protocol (IP) router and act as a normal IP end host. Second, any phone could proxy a Bluetooth profile to the cloud on behalf of a device. The former allows for a high degree of flexibility while the latter may be better suited to the power and processing constraints of the device. Both can be implemented as part of an independent app or operating system (OS) service on the phone. This chapter describes an implementation of the profile proxy approach as a background Android application, *Gateway*.

Current applications cannot be entirely replaced by transparent gateways, however. The asymmetry in capabilities between smartphones and peripherals leads to some application-specific functionality, like location information or user interfaces, being handled by the phone. To support some such usage scenarios, we further extend the architecture to allow devices to request certain services from the paired smartphone, such as supplementing data sent with

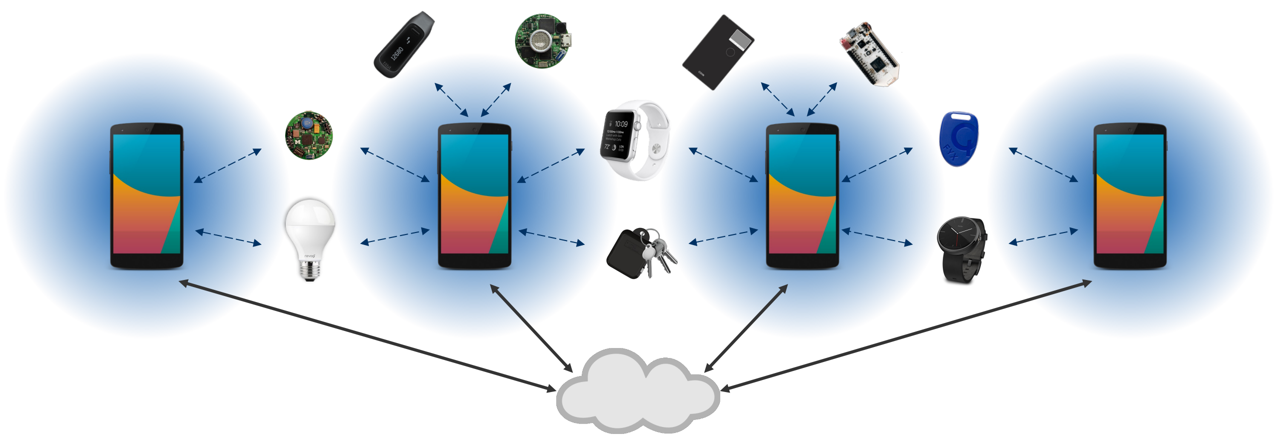


Figure 4.1: Mobile Gateway Model. In a mobile gateway architecture, the existing network of smartphones could collectively act as an open, ambient bridge of connectivity between constrained IoT devices (via Bluetooth) and the cloud (via cellular/Wi-Fi).

information like the phone's location, the current time, or measurements from on-phone sensors. Services like these may be critical to the application but difficult for a cost-and energy-constrained peripheral device to acquire on its own. Additionally, we introduce a service to present smartphone users with an associated dynamically loadable interface to interact with the device, without requiring a device-specific app. An early implementation of such a service, *Summon*, is included in the *Gateway* application. This extension of the architecture suggest a possible new role for the smartphone—as an opportunistic context server for nearby devices.

With this approach, the gateway is decoupled from apps, as depicted in Figure 4.2. It eliminates the need for for siloed networking components within apps, and instead allows any nearby phone to collect data for them using a core gateway service. The app's unique and novel user interface (UI) could be retained and continue to pull data from the cloud, or in some cases, the need for a native app can be eliminated in favor of linked dynamically loadable interfaces, built using standard web technologies.

In this chapter, we make a case for the architecture by describing a number of motivating applications and propose an open mobile gateway architecture. We explore the design, implementation, and evaluation of *Gateway*, an implementation of BLE profile proxy via smartphones, and *Summon*, an included service that enables proximity-driven availability and opening of full-featured, interactive, and remote interfaces for BLE devices using web-technologies.

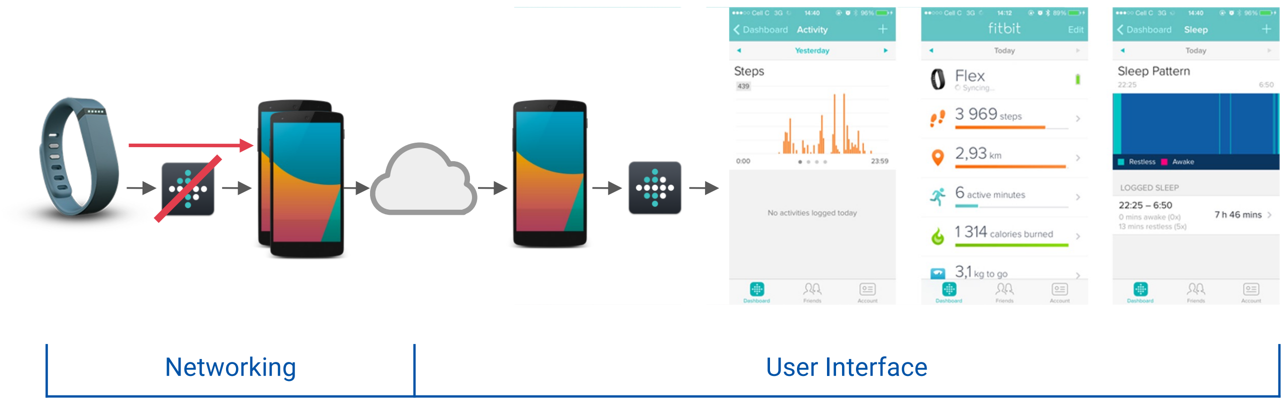


Figure 4.2: Approach: Decoupling core gateway services from mobile apps. With this approach, a separate (ideally, OS-run) general gateway service on smartphones facilitates networking between potentially any device and the cloud. The UI can continue to retrieve data from the cloud. The UI also may, itself, be retrieved from the cloud, at a location specified by the device.

4.1 Background & Related Work

The proposed approach is motivated by previous related work that spans delay tolerant networking, data muling, existing IoT ecosystems, BLE proximity services, and web-based interfaces.

4.1.1 Delay Tolerant Networking.

The use of mobile phones as gateways leads to challenges stemming from the lack of continuous network connectivity. Mobile wireless ad hoc networks allow for the continuation of previously disrupted communication when the mobile node is in range of the network. This type of routing has been demonstrated in many projects involving delay tolerant networking [71, 130]. Consideration of work that describes the tradeoffs of delay tolerant networks in energy, latency, and storage is helpful for implementation of the architecture [147].

4.1.2 Data Muling.

Many projects demonstrate that data mules, mobile surrogates such as smartphone gateways that can transport data between two hosts that would otherwise be unable to communicate with one another, can provide connectivity for sensor networks [35, 88]. Additionally, data muling over Bluetooth on human-carried mobile phones has been shown to provide a reliable network, even for remote sensor deployments [128].

4.1.3 Existing Services for IoT Devices.

Over the past couple of years, a number of companies have announced services promoting the connection of smart products. Thread, for instance, is described as a home-based mesh network capable of connecting hundreds of products within a house and enabling online control via a border router connection to Wi-Fi [152]. Helium is a platform developed for metropolitan-sized networks of low-powered connected devices using a modified 802.15.4 protocol and IPv6 addressing, but optimized for very low data transfer [77]. Apple's Homekit is a framework available since the release of iOS 8 that enables communication and control of connected products in the house which meet Apple's technical specification [18]. The AllSeen Alliance is a group of consumer brands promoting mainstream adoption of an interoperable and universal software framework for the Internet of Things based on the AllJoyn open source project [7]. Similarly, the Connectivity Standards Alliance is working on Matter, another interoperability protocol for devices connected through Wi-Fi or Thread [41].

4.1.4 BLE Proximity Services

In 2013, Apple introduced iBeacon, a low-powered and low-cost BLE-based proximity solution that specifies the public transmission of unique application-specific identifying information in a BLE peripheral's advertisements for which iPhones specifically scan as a background operation [20]. Detection of an application's known peripheral identifier on an iPhone can prompt various actions based on the peripheral's signal strength, enabling services like location-based advertisements and rough indoor navigation.

In 2014, Google introduced UriBeacon, which goes in a slightly different direction by, instead of prompting actions in an app, opening a URL specified by the nearby beacon within the mobile phone's browser [70]. Similarly, in the implemented *Gateway* protocol, a peripheral specifies a URL, and if the peripheral advertises a request for *Summon* mode, *Gateway* can open the URL—not, however, as just a site in a browser, but as a native-like interactive application.

4.1.5 Interfacing Using Web Technologies

Apache Cordova [11] and its popular distribution, Adobe PhoneGap [5], are mobile development frameworks that allow software programmers to develop mobile device applications using JavaScript, HTML, and CSS, instead of relying on platform APIs like ObjectiveC/Swift for iOS or Android-flavored Java. This is enabled by wrapping the code in an embedded web view with Javascript bindings for native functions on the respective device. While the software is developed with web technologies, they are not meant to be opened in the web browser, as they lose all or most functionality. Instead, they are typically package as native apps for distribution on each platform. This is a popular choice among developers who intend on porting their application to multiple platforms, or who are just more well-versed in web development.

Inspired by this approach, the proposed *Summon* service allows peripherals to open what are essentially unpackaged Cordova apps hosted online. Once pulled up on the smartphone, the apps are provided with appropriate bindings to act like native apps, and can interact with the device that "summoned" the application without the hassle of having to actually install an app. To enable, this, a *Summon* mode flag is introduced in the *Gateway* API for peripherals. While this chapter presents an early implementation of the service, the next chapter features a more significant focus on and expansion of this concept.

4.2 Applications

To motivate the need for a well-defined, cross-platform architecture for connecting low-power devices and sensors to the Internet, we explore several applications that are enabled or improved by the proposed gateway architecture.

4.2.1 Ambient Data Collection

Sensors installed in buildings, homes, cities, remote environments, and other locations can provide invaluable streams of data for monitoring, control, analysis, and prediction applications. Retrieving data from each device, however, is often challenging due to sensor power constraints, poor wireless connectivity, or expensive data links. One solution that has been extensively studied is to mesh-network sensors to allow data packets to hop through the network, but this often fails in areas with poor RF characteristics, and the demands of packet forwarding take a substantial toll on sensor lifetime.

In contrast, the proposed BLE gateway architecture would leverage the smartphones that people already carry to collect data from installed sensors. As an example, consider scientists seeking to measure temperature and relative humidity in a forest by deploying sensors. Rather than requiring a cellular data plan for each sensor or the scientists to visit each node periodically, one could imagine a system where hikers traveling on well-defined trails can provide connectivity for these sensors. As a hiker walks by a sensor, the sensor will attempt to use the hiker's mobile phone as a gateway. Because the sensors conform to a common architecture, a hiker would not need to download any software to connect to the sensors. The phone, which may be disconnected from a data network, could hold the data for some time before forwarding it. Hikers may be interested in being a courier for the data because of its scientific nature [10], or because the scientists will compensate them [93].

This method of data retrieval can extend to other applications as well. Sensors installed in buildings, particularly older buildings with challenging RF characteristics, could use the daily occupants of that building to relay their data. In this case, the occupants may be incentivized by obtaining controls for temperature and lighting on their smartphones in exchange for forwarding sensor data.

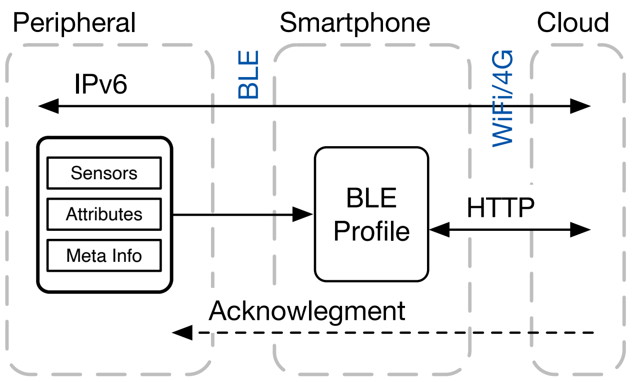


Figure 4.3: Proposed architecture. The gateway architecture introduced in Chapter 3 is applied and extended to mobile infrastructure. In it, Internet connectivity for BLE devices could be facilitated: (1) via IPv6, using the smartphone as a temporary IPv6 router and treating the peripheral as an IP-connected end host, and (2) via proxy, using the smartphone to forward the peripheral's BLE profile to the cloud.

4.2.2 Cross Platform Connectivity

Some devices are limited by the model of smartphone to which they are capable of connecting. For example, many devices are only compatible with iOS. This closed, siloed approach is detrimental to the growth and usefulness of this class of device.

With an open gateway architecture, any device could potentially ask any smartphone it encounters to agree to act as a gateway. The phone could then provide a connection for low-bandwidth Internet applications running on the device. Certain applications which are highly user-specific, such as notifications on the smartphone, may still require a specific smartphone or app running on the phone. The device can, however, link to its own dynamically loadable web-based interface, which, when detected and opened on the phone, may enable some aspects of device-user interaction, regardless of the phone platform or the presence of an associated native app.

4.2.3 Masking Smartphone Failures

Requiring a BLE peripheral or wearable device to link to exactly one smartphone inserts an unnecessary failure point for these devices. If the paired smartphone is not present or is discharged, the otherwise functional tethered device loses its ability to send or receive data. An open gateway model would allow devices to use any nearby smartphones to forward or receive data. In certain situations, such as when using a fitness monitor at the gym or after a smartphone's battery has depleted, it would be preferable not to lose functionality because a specific phone is unavailable, as many do today.

4.3 Gateway Overview & Design

To provide Internet connectivity for resource-constrained devices, we propose a smart-phone-centric approach. Smartphones can act as useful gateways due to their near-constant Internet connection, mobility, and ubiquity, but doing so means they also dictate what wireless protocol compatible IoT devices must use. Although Wi-Fi is ubiquitous throughout much of the world, and is presently implemented in many IoT devices, its large power requirements make it unsuitable for low-power applications. While some low-power links, like IEEE 802.15.4, provide features that would be useful in this regime, their lack of smartphone support make them unattractive. Bluetooth Low Energy (BLE), on the other hand, is a more promising protocol for connecting IoT devices via mobile infrastructure. Its widespread deployment in smartphones and suitably low-power draw make it an attractive solution. So we design the gateway architecture using BLE as the underlying technology.

4.3.1 Network Scheme

BLE is a link-based, point-to-point protocol between two devices, one in peripheral mode and the other in central mode. In the proposed architecture, the smartphone remains in central mode while all IoT devices behave as peripherals. Peripheral nodes transmit periodic beacons, termed advertisement packets, to notify nearby central nodes of their presence. Once a central device hears an advertisement, it can establish a connection between the two devices to transfer information. This connection process is standardized by the BLE specification. How and which information is transferred between the device and smartphone is specific to each application, however. To allow the phone to behave as a generic gateway, we consider the architecture from Chapter 3, which suggests two general and reusable data transport mechanisms that many applications could use. An overview of the architecture is shown in Figure 4.3.

IPv6 Routing. The first data transport mechanism between BLE peripherals and smartphones is a raw IPv6 packet transfer over BLE. This would allow each IoT device to behave as any other IP end host and to take advantage of the flexibility of working at the network layer. The phone acts as an IPv6 router between its Internet connection and the peripheral. The mechanisms for building this IP network on a BLE link are currently being formalized by the IETF and Bluetooth SIG [31, 123, 124], and early versions of IPv6 stacks for BLE peripherals and phones shows promise for future work [82, 131, 145, 162].

The primary challenge to using this data transport is the complexity of communicating at the IP layer. All resource-constrained peripherals should not be expected to support a full IP stack, on top of BLE. Further, this class of sensor can benefit from offloading work to a more capable device. In addition, network IPv6 support for smartphones is growing slowly, but remains sparse [64]. While the flexibility of providing an IP layer is extremely beneficial for supporting a wide variety of applications, we look to the second data transport option which offers less flexibility but is better optimized for immediate use with the BLE specification and contemporary IoT device applications.

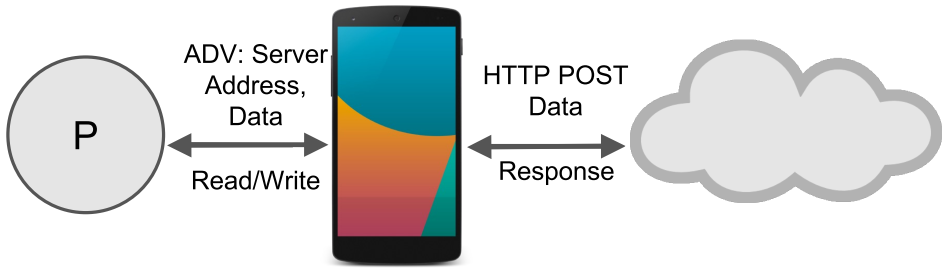


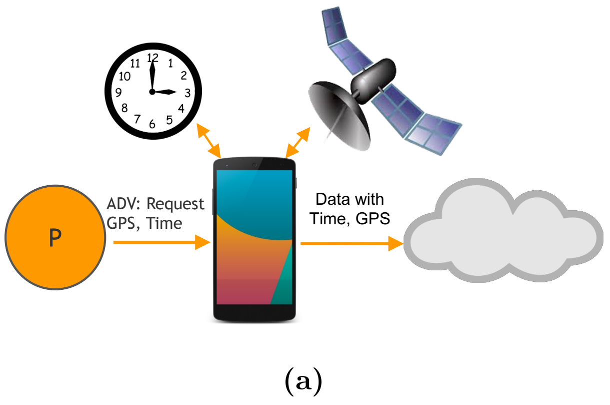
Figure 4.4: BLE Profile Proxy In default operation, the smartphone facilitates connection and profile-based communication between the BLE peripheral and the cloud endpoint specified in its advertisement.

BLE Profile Proxy. The second data transport mechanism operates by using the smartphone gateway as a proxy for the information contained in the BLE data structures on the peripheral. At a high level, the gateway relays the services, characteristics, and attributes shared with it from the BLE peripheral to a remote server. This more naturally aligns with existing BLE devices, as the data organization between the peripheral and central node in existing, application-specific BLE interactions does not fundamentally change.

To support this proxy architecture, IoT peripherals must extend the data they send to the phone with meta information that dictates how the phone should proxy the BLE profile data. This configuration meta information can be contained in the peripheral's broadcasted advertisements, to which the gateways will have access without requiring an explicit connection with the device. This meta information can include parameters specifying destination, reliability, and user incentives for sending the data, as shown in Figure 4.4. In addition, in this model, there is a unique opportunity for the peripheral to request the smartphone to supplement the data with its own inputs, like GPS data, time, or measurements from its on-board sensors, as in Figure 4.5a.

4.3.2 Application-Specific Apps

The proposed architecture separates core gateway components from siloed peripheral-specific apps, as shown in Figure 4.2. However, the architecture is not intended to replace all peripheral-specific apps on a smartphone. Some apps utilize or display data that is collected by the peripheral. These apps can be designed primarily to display information from the backend cloud service, and should, instead of implementing a custom siloed gateway for the peripheral, allow all forwarding data requirements to be handled by the gateway service on any nearby smartphone. But, there are services that may be built on top of the gateway architecture that enable simpler access to device-specific interactive applications, as in Figure 4.5b. Implementation of such a service, *Summon*, is described later.



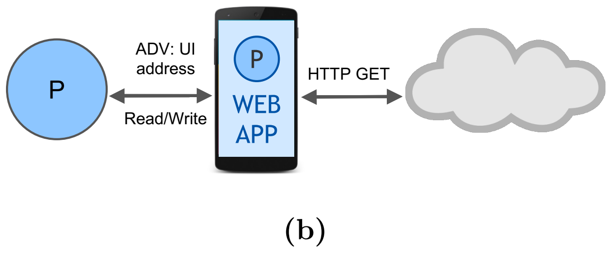


Figure 4.5: Extended services for BLE profile proxy. If the peripheral requests data augmentation services (a), the phone can provide additional data (e.g. location, time) in the peripheral's communication with the cloud. If the advertisement points to a UI location (b), the specified URL can be opened on the phone to allow the user to interact with the device.

4.3.3 Service Parameters

To use the service, BLE peripherals must specify certain configuration parameters. These parameters, as specified in Table 4.1, are communicated in the peripheral's broadcasted advertisements (shown in Figure 4.6), to which the gateways have access without requiring connection with the peripheral.

Destination and Reliability. As part of the parameters advertised, the peripheral must, first, indicate the target destination to which to send and/or receive data. The destination specified should be a shortened URL (e.g. a bit.ly or goo.gl address) of up to 14 characters in length. The parameters should also specify the level of reliability of service it expects, or effectively how UDP-like or TCP-like the gateway connection should be.

Modes: Default and *Summon*. The *Summon* parameter flag indicates in which mode the peripheral wishes the gateway to operate: default mode or *Summon* mode. In the default mode, communication with servers at the the specified target URL is facilitated by sending data over HTTP POSTs and POST responses. In *Summon* mode, the target URL is treated as the address for a user interface. No data is transmitted until the user chooses to open the application, which itself can control communication.

Smartphone Services. Because the gateway is a smartphone device, peripherals may wish to take advantage of some of the unique data inputs the phone can provide. For example, information about the location of the peripheral or the current global time may be difficult for the device to ascertain, but is straightforward for a smartphone, which can append the information to the data being sent. For this purpose, the addition of readings of time, GPS, acceleration, ambient light, pressure, and magnetic field can be requested by peripherals in default mode. In *Summon* mode, the called application can access these features on its own.

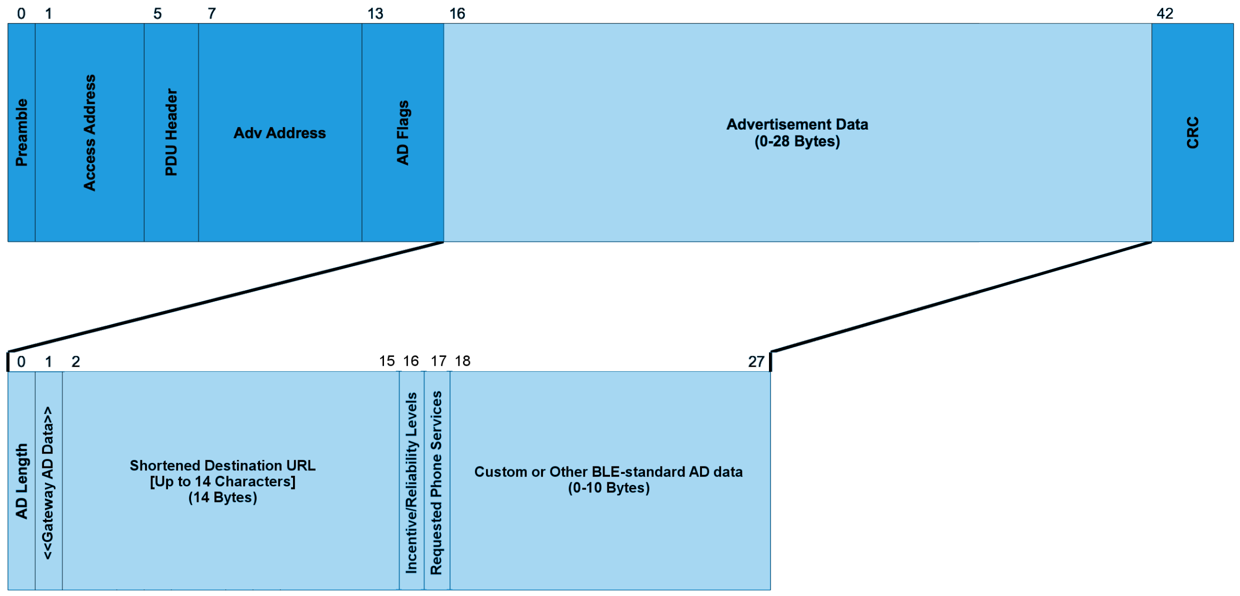


Figure 4.6: Advertisement Packet Format. The layout of a *Gateway*-compatible BLE advertisement packet, broadcast from a participating peripheral, by byte.

User Incentivization. The incentive level parameter allows the peripheral to specify what incentives it is willing to provide to the gateway owner to allow the smartphone to facilitate the connection in the background. In default mode, the current simulated scheme is a system where the level indicates participation in a particular payment program, which is validated upon retrieval of advertisements. In *Summon* mode, the incentive parameter is ignored, as no data is used until the user opens the corresponding application. The assumed incentive of opening the application is the exchange of potential services in return for data, much in the same way Google Maps provides map services in exchange for traffic data [24].

4.3.4 Gateway Administration

Gateway owners should be able to configure how and to what extent their smartphone is utilized as a gateway. The gateway configuration settings allow owners to cap the data rate and set preferences for the service parameters, like what modes of operation are supported, which smartphone services can be used for data augmentation, and what incentives are acceptable for data transport. Additionally, a gateway should maintain a whitelist and blacklist to enable fine-grained access control.

Parameter	**Value**	**Bits**
Shortened Destination URL	Up to 14 chars	112
Incentive Level	0-15	4
Reliability Level	0-15	4
Requested Phone Services		
Time Access	0,1	1
GPS Access	0,1	1
Accelerometer Access	0,1	1
Ambient Light Sensor Access (0/1)	0,1	1
Barometer Access (0/1)	0,1	1
Magnetic Field Sensor Request (0/1)	0,1	1
Requested Mode		
Summon (URL is an address for a UI)	0,1	1
IP [Reserved]	0,1	1
Optional		
Custom App Data	0-10 bytes	0-80

Table 4.1: Advertisement parameters. The set and structure of the parameters to be advertised by the peripheral. Any BLE peripheral wishing to use the Gateway service, need only encode these parameters in the advertisement service data. If *Summon* mode is set, instead of sending an HTTP post of the advertisement to the specified URL, the gateway can open the URL as a fully-functioning HTML/Javascript-based user-facing app.

4.3.5 Operation

In the *Gateway* service, the smartphone gateway continually scans for peripherals in the background. Upon discovering a peripheral and receiving an advertisement, the service takes steps depicted in the operational flow diagram shown in Figure 4.7. From, the advertisement, it can determine the destination URL and the mode of operation.

If *Summon* mode is requested by the peripheral and is allowed by the administration preferences, placed in a device list and the smartphone owner is notified about the nearby peripheral has an interactive application associated with. The owner can view the device list to see the peripheral and URL associated with it. If the owner chooses, he can select the peripheral to open the application at that URL.

If *Summon* mode is not selected, the gateway parses and verifies allowance of the remaining parameters in the administration settings. Once verified, it attaches any additional data from the requested smartphone services, and sends data to the destination URL, to which the destination server can respond with a request for BLE connection and specify actions.

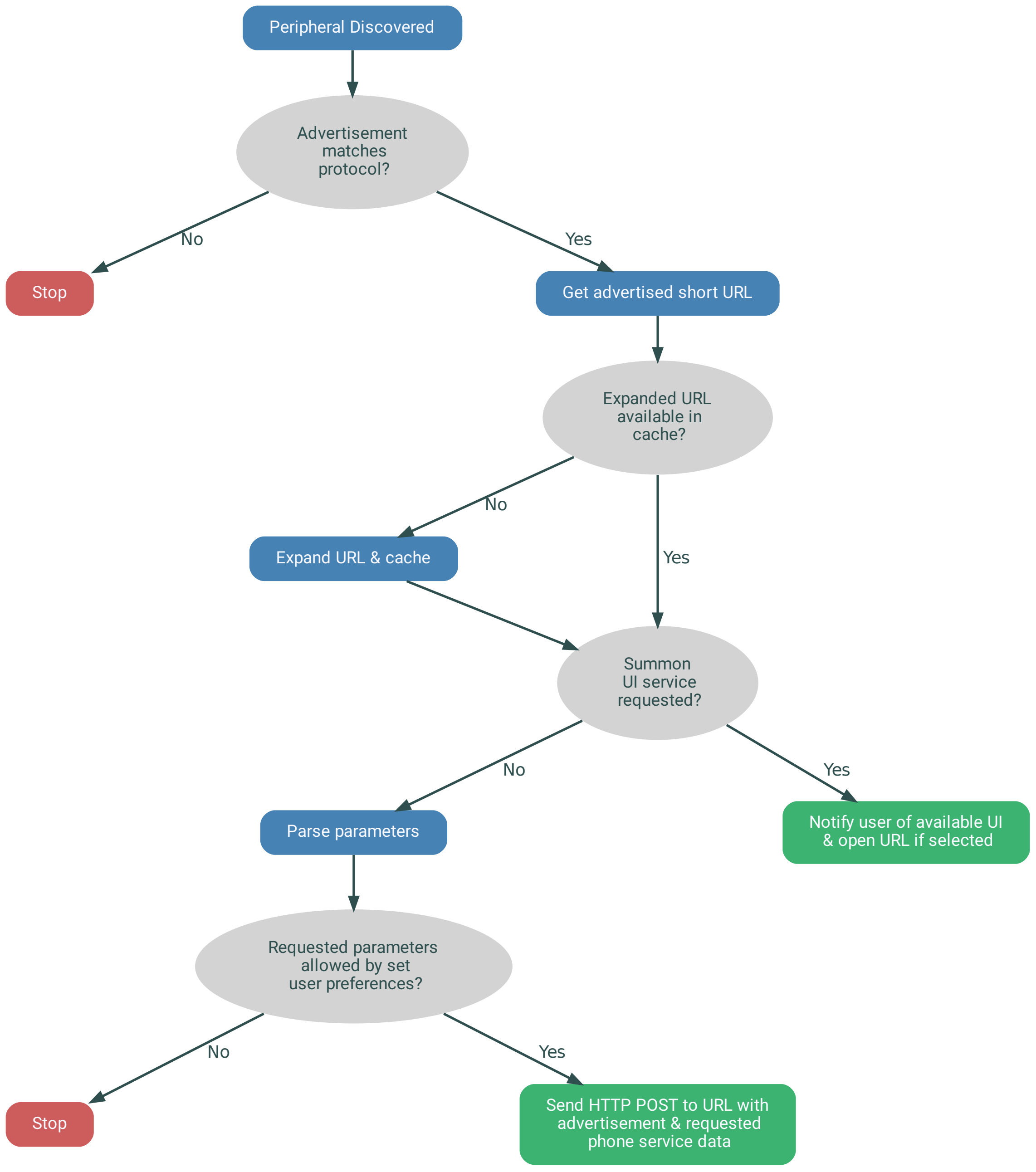
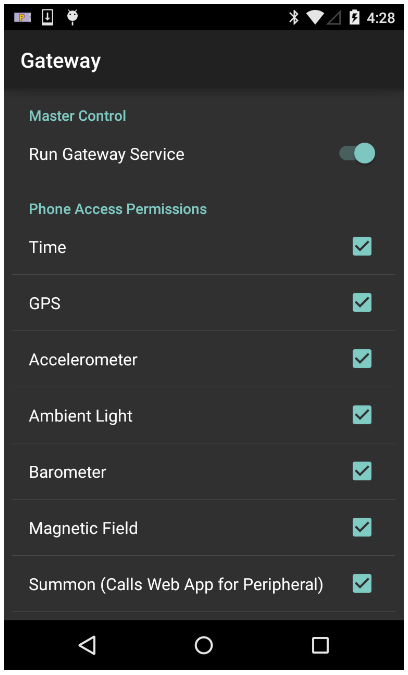

Peripheral Discovered
Advertisement matches protocol?
No
Yes
Stop
Get advertised short URL
Expanded URL available in cache?
No
Yes
Expand URL & cache
Summon UI service requested?
No
Yes
Parse parameters
Notify user of available UI & open URL if selected
Requested parameters allowed by set user preferences?
No
Yes
Stop
Send HTTP POST to URL with advertisement & requested phone service data

Figure 4.7: Operational flow of *Gateway* upon receiving an advertisement. The smartphone gateway scans for peripherals, and upon receiving a compatible advertisement, determines the specified URL and the mode of operation. If *Summon* mode is requested, the user is notified and can choose to open the URL-specified web app. Otherwise, the gateway parses the remaining parameters (see Table 4.1), and, if permitted by user preferences, attaches and sends data to the destination URL, to which the destination server can respond with a request for BLE connection.



Gateway

Master Control

Run Gateway Service

Phone Access Permissions

Time ✓

GPS ✓

Accelerometer ✓

Ambient Light ✓

Barometer ✓

Magnetic Field ✓

Summon (Calls Web App for Peripheral) ✓

Figure 4.8: Settings screen for *Gateway* Android app. The Gateway app lets smartphone owners set fine-grained access control preferences for each of the parameters based on their own comfort level. This way, the service can run in the background, while maintaining the smartphone owner's peace of mind.

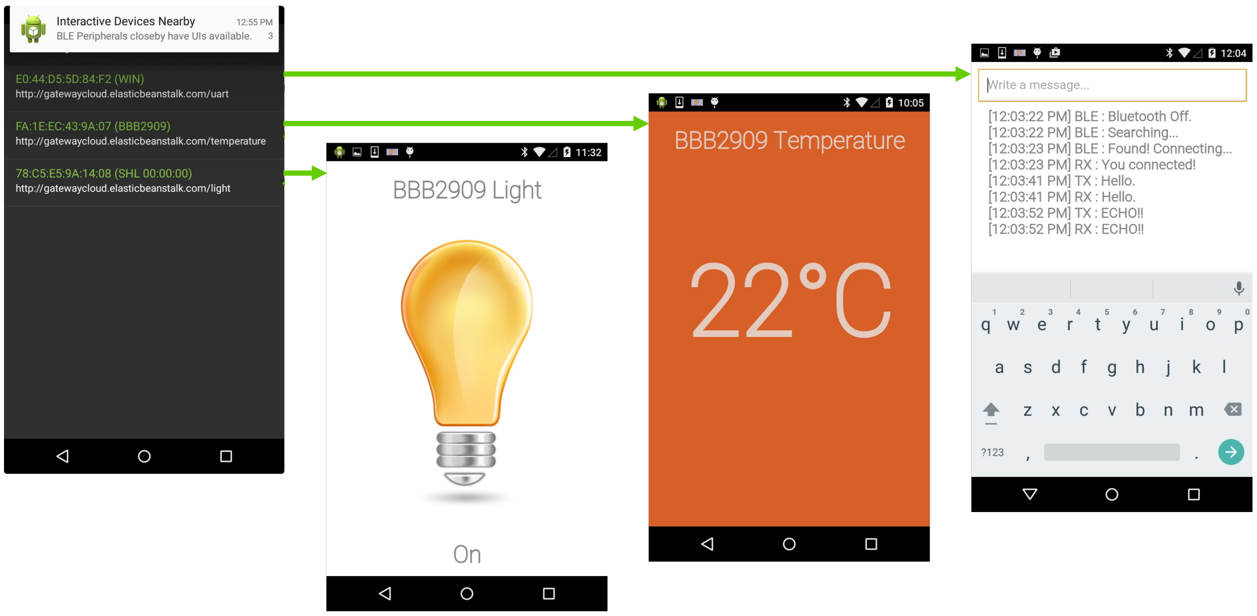
4.4 Mobile Gateway Implementation

The implementation consists of (1) an Android *Gateway* application with the *Summon* service, (2) a number of peripherals set up to use the *Gateway* API, (3) a web server to serve as the target destination for the peripherals' data and instruct the Gateway to perform connections and actions on its behalf, and (4) a few Cordova-based applications that allow users to interact with the BLE peripheral. All of the implementation for the project is available on Github [98].

4.4.1 *Gateway* App

Gateway is implemented as an application layer service that runs on Android 4.3 and up. It fulfills the role of the BLE Profile Proxy Gateway in the open gateway network architecture.

Operation. The application runs in the background and scans for peripherals that advertise according to the parameter specification shown in Table 4.1. Once it receives the advertisement, it follows the operational flow shown in Figure 4.7 and discussed in Section 4.3.5.


Interactive Devices Nearby 12:55 PM
BLE Peripherals closeby have UIs available. 3

E0:44:D5:5D:84:F2 (WIN)
http://gatewaycloud.elasticbeanstalk.com/uart

FA:1E:EC:43:9A:07 (BBB2909)
http://gatewaycloud.elasticbeanstalk.com/temperature

78:C5:E5:9A:14:08 (SHL 00:00:00)
http://gatewaycloud.elasticbeanstalk.com/light

```
[12:03:22 PM] BLE : Bluetooth Off.
[12:03:22 PM] BLE : Searching...
[12:03:23 PM] BLE : Found! Connecting...
[12:03:23 PM] RX : You connected!
[12:03:41 PM] TX : Hello.
[12:03:41 PM] RX : Hello.
[12:03:52 PM] TX : ECHO!!
[12:03:52 PM] RX : ECHO!!
```

Figure 4.9: Device manager screen and device-based web apps. If a peripheral advertises being in *Summon* mode, the user is notified that the peripheral has a UI, and it is listed in the device manager. When the user selects a device, Gateway "summons" the Cordova[11]-based HTML/Javascript app at the specified URL. Since it is opened within the Gateway framework (as opposed to a browser), the page will be able to act like a native app installed on the phone.

Administration. The application has a settings screen, shown in Figure 4.8, that allows the owner to enable access for particular smartphone services, specify the allowed incentives, When an advertisement comes in, the application parses out the payload data and checks the values against against the access preferences, to determine whether or not to facilitate the data transport. In addition, the owner can specifically set what services the *Summon* applications may use.

Default Mode. The application handles requests for both default mode and *Summon* mode at the same time. In default mode, the application sends an HTTP POST of the advertisement data to the specified destination URL. A POST response can be sent back to the gateway with instruction to initiate connection and perform read, write, and notification requests over Bluetooth. After actions are performed, the gateway sends the data back to the URL via HTTP POST, and may receive another response with more actions. This process can continue iteratively as long as the connection remains established.

***Summon* Mode.** In *Summon* mode, *Gateway* adds the advertised peripheral and its destination URL to a device manager list, and notifies the smartphone user of the presence of a peripheral with an interactive application. As shown in Figure 4.9, the smartphone user can open the device manager screen to view a list of the nearby peripherals and corresponding applications. If the user selects one to open, the URL is opened within the *Gateway* context.

Because Apache Cordova [11] JavaScript bindings are built-in to *Gateway*, the HTML page of any application developed using the Cordova or PhoneGap [5] framework can open like a native application on the smartphone. Included is support for Bluetooth Low Energy [39], allowing the application to access the device that "summoned" it.

Platforms. This implementation has been tested on a Blu Dash JR [27], an HTC One [80], two Nexus 4s[65], and two Nexus 9s [66].

4.4.2 Peripherals

A number of peripherals have been set up to use the *Gateway* API. A list of the devices used can be seen in Table 4.2. Each of the devices are programmed to demonstrate a different unique feature.

The Nexus 9 [66] runs an Android app can build and test any *Gateway*-compatible advertisement. This is particularly useful for testing all of the parameters and attachment of data from the set of smartphone services.

The Raspberry Pi [50], using a Kinivo BTD-400 Bluetooth dongle, runs a NodeJS app that increments a readable characteristic value every time the gateway connects to it.

The Tessel [150], using a BLE113A Bluetooth Smart module, runs a NodeJS application that sets up a temperature and humidity service. A connected gateway can read temperature and humidity characteristics, and write to a characteristic that specifies whether the temperature reading should be in Celsius or Fahrenheit.

The Opo [99], using a Nordic nRF8001 Bluetooth component, sets up a large-scale data offload service. A connected gateway, receiving notifications every time a characteristic changes, continuously re-reads a characteristic after every notification to collect a large stream of data sent in 20 byte chunks.

Two Squalls [76], using the Nordic nRF51822 Bluetooth component, and the Robosmart Light Bulb demonstrate the use of *Summon* mode of operation. Each one specifies a URL in their advertisement that contains a corresponding application that allows interaction with the device. One Squall has a UART over BLE application. Another has a temperature reading application. The light bulb has a light control application.

4.4.3 Web Server

A remote web server is implemented to receive, store, and respond to data from the peripherals operating in default mode. The server is able to instruct the gateway to form BLE connections and perform any actions, like read, write, or receive notifications on its behalf. For instance, when the initial HTTP POST message containing Tessel's advertisement is received at the destination web server, it sends a POST response with instruction to connect, read the temperature (initially in Celsius) and humidity characteristics, write to the temperature format characteristic to toggle to Fahrenheit, and re-read the temperature characteristic (now in Fahrenheit). The read data and confirmation of write is then sent back to the web server.

Device	Feature	Mode	Advertise	Read	Write	Notify	User Interface
Squall [76]	Temperature in Advertisement	*Summon*	Yes	No	No	No	Temperature Display
Squall [76]	UART Service with Notification	*Summon*	Yes	Yes	Yes	Yes	UART Console
Squall [76]	Static Test Data in Advertisement	Default	Yes	No	No	No	No
Tessel [150]	Temperature/Humidity Service	Default	Yes	Yes	Yes	No	No
Nexus 9 [66]	Any Parameter in Advertisement	Default	Yes	No	No	No	No
Opo [99]	Large Data Service with Notification	Default	Yes	Yes	Yes	Yes	No
Robosmart Bulb [148]	Light Control Service	*Summon*	Yes	Yes	Yes	No	Light Switch Control
Raspberry Pi [50]	Connect Counter Service	Default	Yes	Yes	No	No	No

Table 4.2: Peripherals using *Gateway*. The set of peripherals set up to use the *Gateway* API. All devices work successfully with the *Gateway*, and each demonstrate a unique feature. Three peripherals, in particular, demonstrate the proximity-based application "summoning". Reading, writing, and receiving notifications are actions that the gateway can perform on a peripheral after a connection to the peripheral has been established.

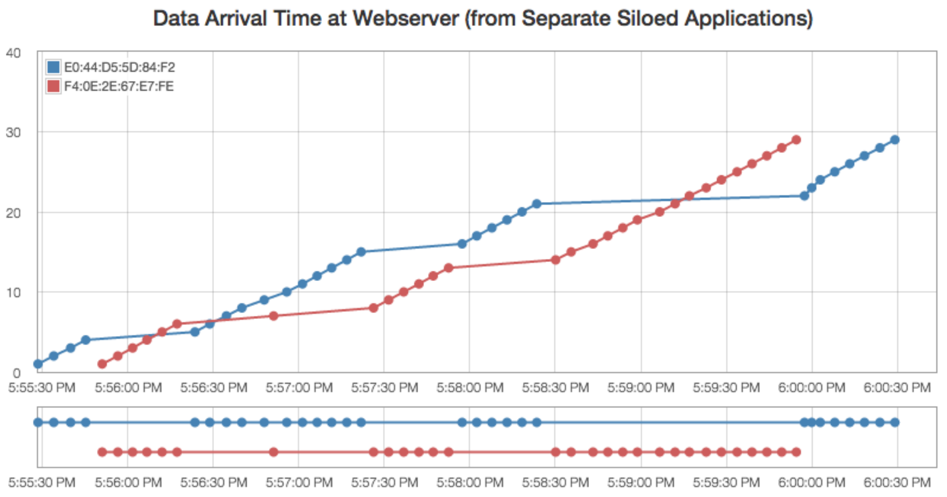
Also hosted on the server are the three HTML/JavaScript *Summon* applications. They make use of the Apache Cordova framework and are successfully able to interact with their corresponding devices when opened on a Gateway.

4.5 Mobile Gateway Analysis

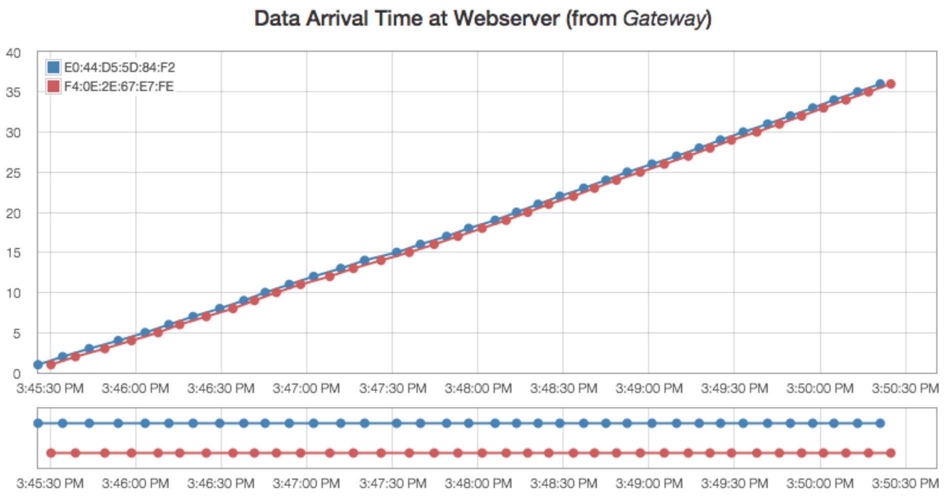
A set of evaluations were performed to verify and characterize the functionality, ability, and operation of *Gateway* and *Summon*.

4.5.1 *Gateway* Functionality and Performance

Every one of the peripherals listed in Table 4.2 have been able to successfully demonstrate their respective unique feature, as described in Section 5.3.4. Three peripherals, in particular, successfully demonstrate "summoning" of applications deployed online that interact with the corresponding devices. The web server implementation properly interacts with *Gateway* in response to proxied advertisements in order to establish connections with peripherals, and perform BLE operations. In terms of overall performance, the use of a central standard gateway service allows more evenly distributed and often higher throughput than when the smartphone attemps to context-switch between all of the siloed application-specific implementations of gateways. To illustrate this, Figure 4.10 compares the times at which data arrives at the destination web server in both a siloed application setup and a *Gateway* setup. This is a demonstration using only one smartphone gateway. The availability of a universal gateway on multiple smartphones can drastically increase accessibility to the Internet.



(a) Arrival pattern for peripheral data through siloed applications



(b) Arrival pattern for peripheral data through *Gateway*

Figure 4.10: Comparison of data arrival patterns. For this experiment, two peripherals are set up to be scanned, connected, and read by the smartphone, which in turn sends the data to the destination webserver. Each data arrival point represents the conclusion of a full iteration of the process. In the first scenario (a), the smartphone uses a siloed application setup in which two different smartphone applications implement their own version of a gateway by hardcoding the entire process to specifically cater to one of the peripherals. In the second scenario (b), the smartphone uses *Gateway*, which instead receives instruction from the webserver on which specfic actions to take and still manages to send data in reasonable time. Due to the siloed nature of the (a) scenario, the smartphone operating system is forced to context-switch between the applications in order to send data. Because (b) uses a core standard service, arrival is more evenly distributed.

	Energy	% Battery
LCD Screen	4 J	~0.0127%
BLE + CPU	616 J	~1.9578%
Wi-Fi	1400 J	~4.4495%
Total	**2020 J**	**~6.4200%**

Table 4.3: Daily energy usage for the *Gateway* service. Energy usage of the *Gateway* service is broken into LCD, BLE + CPU, and Wi-Fi categories. The service was run as a background process on a Nexus 5 with 3 peripherals spread 10 ft away. The battery percentage represents the portion consumed of the smartphone's 2300mAH battery. Part of the high Wi-Fi usage is attributed to extra data being sent to servers for debug-viewing purposes.

4.5.2 *Gateway* Power Usage

While power efficiency has not been a primary focus while implementing *Gateway*, it is generally beneficial to consider power usage of background services that continuously run on smartphones. To evaluate, approximate power measurements were taken using PowerTutor, an energy profiler tool made for Android. With measurements taken over 10-second intervals, the tool touts an accuracy within 0.8% on average with at most 2.5% error[173]. It was used to measure energy consumption of the background *Gateway* service over the course of 24 hours with 3 connecting peripherals spread 10 ft away. Table 4.3 shows the energy usage broken into CPU, Wi-Fi, and LCD categories. It shows that the application consumed a little over 2kJ—an average power draw of about 25mW. Bluetooth's usage is contained within the CPU category. It should be noted that part of the high Wi-Fi usage is attributed to extra data being sent to servers for debug-viewing purposes.

4.5.3 *Gateway* Data Usage

Approximate web data usage measurements were taken using Android's built in data usage monitoring tool. Measurements were taken on a Nexus 4 over the course of 2 weeks. During those two weeks, the phone was present around 1-3 *Gateway*-compatible peripherals at any given time. The smartphone used approximately 160 MB of data. 75% of this usage is attributed to the background service of *Gateway*. The other 25% is attributed to the opening of *Summon* applications approximately 3000 times.

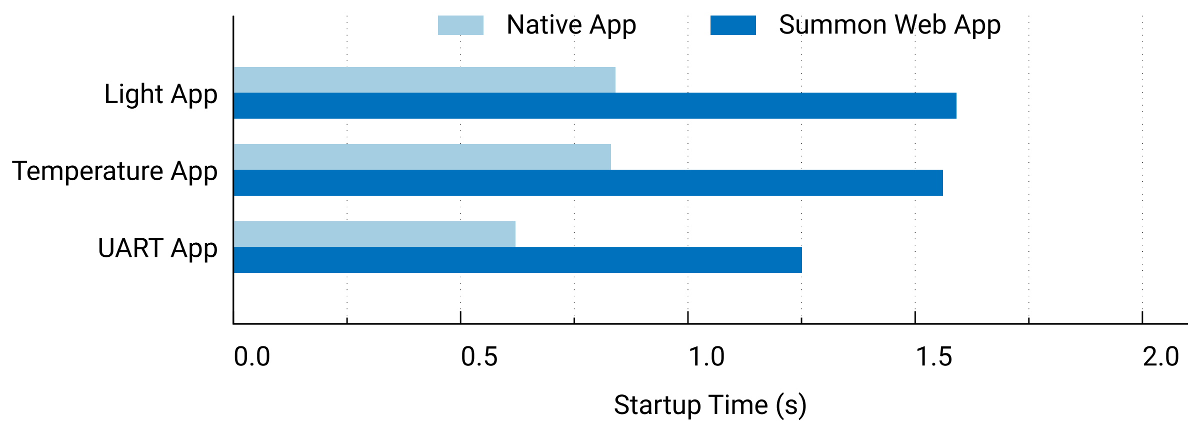


Figure 4.11: Native app vs *Summon* web app: startup times. Average startup times from initial launch to receiving Bluetooth data (over 10 runs each on a Nexus 4) of the HTML/JavaScript based apps, tested both natively (installed on the phone), and in the *Summon* service (retrieved from web server, uncached).

4.5.4 *Summon* Data Usage

The data usage measurements indicate that the 3 *Summon* web applications have an average download size of 13kB. These applications are minimal and fairly light weight. This represents a rough lower bound on usage size. For reference, the download of an average web page in a mobile browser, uses approximately 1156kB of data according HTTPArchive [81]. And with average native application sizes at over 6 MB [1], *Summon* seems to provide a reasonable alternative to the many one-time-use, event-specific, and location-specific applications on the smartphone app stores.

4.5.5 *Summon* Latency

Figure 4.11 shows a comparison of the Apache Cordova-based apps running natively and in the *Summon* service, called from a web server. The latency of opening a web app with *Summon* varies with the quality of Internet connection, but in most cases, took 1-2 seconds from initial launch to receiving Bluetooth data, when no data was previously cached. Timing was recorded using the logging console in AndroidStudio. No noticeable difference in the speed of performance during the actual runtime of a *Summon* application was observed in comparison to that of its native counterpart.

4.6 Discussion

In this section, we discuss further topics about the mobile gateway approach explored in this chapter.

4.6.1 Reducing Advertisement Overhead

Currently, as seen in Table 4.1, the advertisement parameters take up a minimum of 16 bytes of what is essentially a 26 byte payload. This is less overhead than encoding an iBeacon[20] or a UriBeacon[70]. Still, lower overhead in the precious advertising space makes it easier for peripherals to adopt the Gateway API while still being able to use space for other purposes. One way to improve could be to refactor the URL space, which is currently fixed at 14 bytes, to instead dynamically size to fit the URL length. In addition, a byte can be encoded to truncate the URL even further for the most popular short URL base domains (e.g. bit.ly, goo.gl, t.co). For instance, the URL “https://goo.gl/jRMxE0”, which currently occupies a space of 14 bytes, could be truncated down to 7 bytes: “jRMxE0” plus a byte code representing “https://goo.gl”.

4.6.2 Extending to Multiple Platforms

The implementation of the *Gateway* service has been created for Android smartphones. This has been primarily due to the flexibility the OS software provides with regard to Bluetooth processes. That being said, the gateway architecture is intended to be a universal standard. It is imperative that the architecture be portable to other platforms. The port to iOS is possible, and would require a few notable tweaks to the overall protocol, but transferring over the *Summon* service is most promising, due to the inherent platform-agnosticity of the web technologies upon which it is based. Supplementing the mobile gateway approach with the previous chapter’s static approach enables greater coverage of the architecture even in areas where smartphones are not as prevalent.

4.6.3 Security Considerations

The proposed network architecture relies on shared access using untrusted, crowd-sourced gateways. It is conceivable that a peripheral owner can localize a gateway owner by receiving data through that gateway from peripherals at known locations. Conversely, a peripheral moving through a collection of colluding gateways could be localized. Various works explore the security and accountability implications to access control over mobile devices, and solutions for policy and anonymization when interacting with devices and gateways of unknown origin, provenance, and intention [3, 9, 26, 44].

4.7 Summary

We propose a general-purpose IoT gateway on modern smartphones as a software service that provides universal and ubiquitous Internet access to BLE-connected IoT devices. This provides a scalable alternative to the narrow, application-specific gateway structure hampering the development and growth of IoT networks today. The described implementation, *Gateway*, successfully utilizes the smartphone as a BLE proxy for peripherals, relaying profile data from IoT devices to the cloud. It also introduces an early version of *Summon*, a convenient mechanism for presenting to a smartphone user, device- and location-specific user interface applications built and deployed using web technologies, instead of the current model of requiring installation of additional software. In following chapters, we expand on this to create a full-featured standalone interaction solution for IoT devices and ecosystems.

If successfully implemented on the global smartphone infrastructure, the proposed gateway architecture could help expedite the growth of a global, highly-connected, robust Internet of Things in a cost-effective and convenient manner. However, even if the vision of *any* IoT device connecting to *any* smartphone proves too radical a departure from the status quo, the current implementation shows that the basic ideas could still be deployed in more constrained administrative domains, like a home, office, or university campus. This approach provides most of the benefits sought while relaxing the more challenging aspects of security, privacy, and trust in the network, opening the door to a post-MANET for the post-mobile era.

Part III

Interaction

Chapter 5

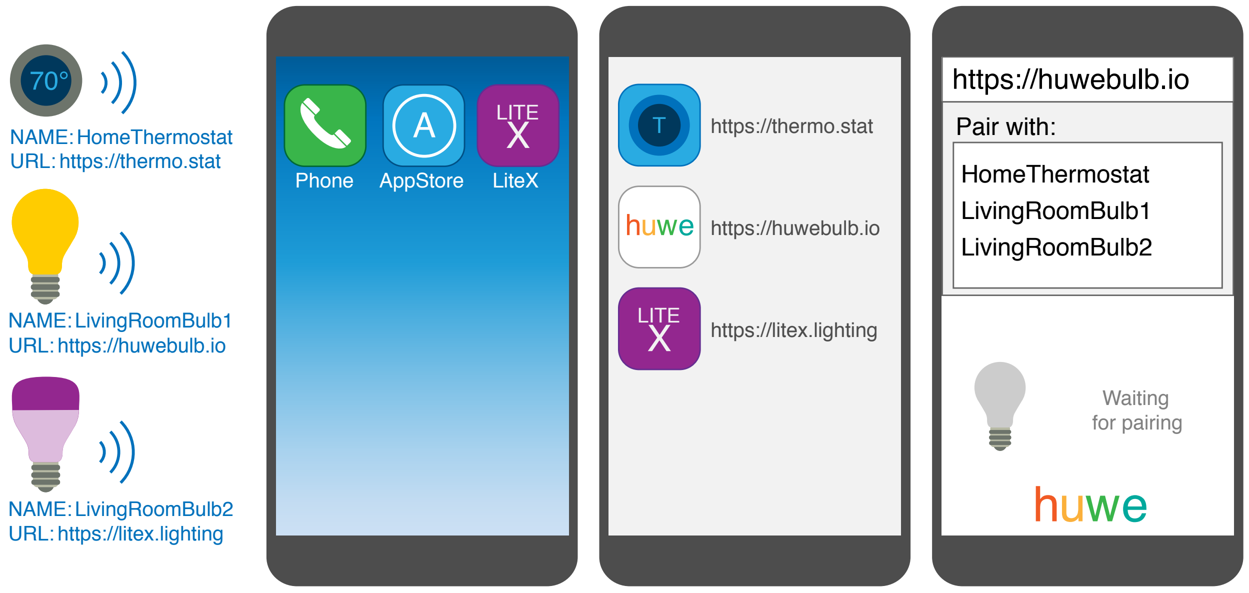
Browsing the Web of Things

The embedded sensors and devices that make up the Internet of Things (IoT) primarily forego physical on-device user interfaces (UIs), like buttons and displays, in favor of more fully-featured software UIs in the form of native apps or websites that run remotely on personal devices like mobile phones. Recently, a variety of tools, protocols, and ecosystems have materialized to help facilitate device interaction on mobile platforms. Unfortunately, due to the amount of burden these systems place on both users and devices, and the poor handling of tasks like device discovery, configuration, and local interaction, the IoT user experience remains awkward, disruptive, and often unintuitive. Even the most successful tools fail to mesh together in a logical way that can fully support a meaningful experience, while also scaling with the rising population of new devices in diverse contexts and constraints.

Currently, the popular form of mobile software interface for IoT devices is the native app. This, by nature, requires the involved process of app installation for every new device and assumes the user will have knowledge of the device prior to doing so. The user may not be aware of all nearby devices or have the appropriate apps required to interact with them, as shown in Figure 5.1a. Additionally, the developers who make these devices are tasked with creating and deploying a mobile app for multiple OSes, informing potential users of the existence of the device's corresponding app, and convincing them to download it. As a general purpose model for the increasing number of IoT devices, this does not scale well.

Furthermore, most mobile-based IoT ecosystems and tools require that both the device and smartphone maintain an Internet connection to operate and to facilitate device interaction. This expectation is not practical for many usage scenarios. Users should be able to browse nearby devices and immediately load dynamic, interactive, and context-specific UIs for devices that are not otherwise Internet-connected. Devices should be allowed to take advantage of the reduced power and complexity of interacting using local network protocols.

We can begin to bridge these divides by associating web content with these connectable devices themselves—effectively tying the *things* to the *Internet*. This concept has been referred to as the Web of Things (WoT) [73]. Recognition of the issues with native apps has led to developments in web standards to support various components of WoT. Notably, the Physical Web project enables discovery of URLs broadcast by nearby devices on a user's



(a) Native App **(b) Physical Web** **(c) Web Bluetooth**

Figure 5.1: Native App, Physical Web, and Web Bluetooth. The native app model (a) requires that a user have prior knowledge of a device and install an app to interact with it. The Physical Web model (b) allows discovery of URLs broadcast by nearby devices, but does not provide device details to the user. The Web Bluetooth model (c) requires that, once a user manually navigates to a website, the user select the device to pair with the website. Even if, perhaps, (b) and (c) are used together, the user would still need to identify the correct device without information on which device pointed to the site.

mobile phone [69], as depicted in Figure 5.1b. While useful, it hides device information when presented to user, which means no association can be made between the device and web content. Web Bluetooth is a drafted W3C standard JavaScript API that allows web pages to communicate directly with Bluetooth devices [161], as depicted in Figure 5.1c. But it requires that, once a user manually navigates to a website, the user select the device that a website should pair with.

By re-focusing on and intuitively extending web standards to better support discovery, connectivity, interactivity, and persistence, we design a browsing architecture that facilitates a broad set of both common and new paradigms for the Web of Things that provides a more seamless user experience and can more feasibly scale. With simple modifications to current web standards and browser-provided APIs, we allow IoT devices to point smartphones to rich app-like web interfaces, that enable greater interactivity than regular websites, fewer memory and latency impacts than native apps, and overall greater flexibility on the device and smartphone. Like a website, the contents of the interactive web interface can be downloaded from the Internet, but it can also immediately interact directly with the smart device over a local network protocol—most commonly over Bluetooth, as depicted in Figure 5.2.

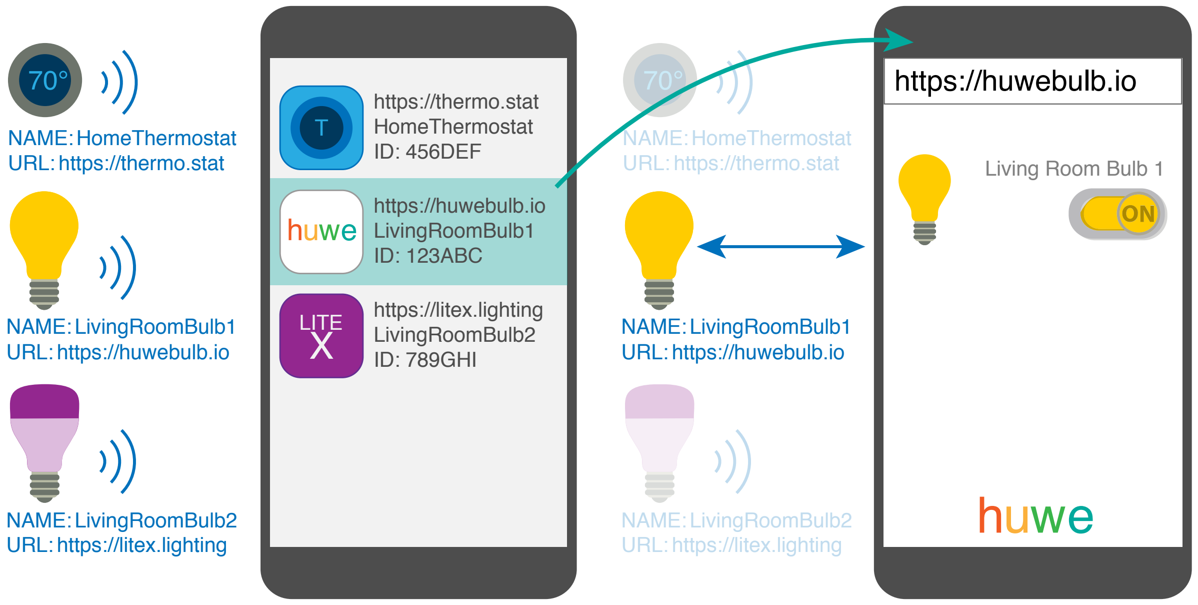


Figure 5.2: Proposed model of discovery, connection, and interaction. We suggest a scheme in which the device broadcasts a URL that is both a link to its UI and its declaration of origin. Users are shown a list of URLs along with details of corresponding devices associated with each. The user can select the link to open the corresponding website, which, with browser-extended APIs, can access devices that have declared it, enabling seamless interaction while providing transparency to users and preventing mismatch of device and UI.

Furthermore, we introduce an origin policy that treats devices as resources of their websites, which when enforced, enables seamlessness between discovery, connection, and interaction.

In this chapter, we sketch out the major design points of the browsing architecture, taking into consideration contributions from recent works that help address some of the highlighted challenges. We then describe and evaluate a new version of *Summon*—a full-featured browser implementation as a smartphone app—and its utilization for real devices by a variety of engineers and developers from the embedded systems community. We identify insights gained, challenges encountered, feedback received, and improvements made in the iterative design process and during the two-year deployment period.

5.1 Background & Related Work

As the Internet of Things has grown in popularity, a set of disparate practices, standards, and ecosystems have emerged to attempt to support it. Unfortunately, most are pieced together with only a limited set of devices in mind. However, new initiatives for web standards have sprung up to more broadly address various portions of this goal. We explore the contributions and vulnerabilities of these systems which have partially motivated how we design our architecture.

5.1.1 IoT Ecosystems and Initiatives

The number of ecosystems intended to interact with the Internet of Things has grown with the ubiquity of the devices themselves. Ecosystems are useful for providing groupings of interoperable devices, but the protocols and standards are typically only designed for or applicable to a narrow set of products (e.g. AndroidThings [60], Apple Homekit [17, 21], Samsung SmartThings [144], Nest [121, 122]). Many of the "standards" set for these ecosystems are not always extensible to network- or energy-constrained settings and are often only scoped to devices in a house or office. Additionally, they still usually require a native app for each device and, often, that the device have its own Internet connection either through its own Wi-Fi radio or a dedicated stationary gateway. Many also require the use of energy-intensive processes on the devices themselves. Ultimately, in our architecture, developers are afforded the flexibility to design for any such desired ecosystem, while also being provided opportunities for improvements in discovery, device setup, and persistent interaction.

Network communities are making efforts to advance the protocols for wireless communication for IoT applications (e.g. Bluetooth SIG [29], W3C [163], Thread Group [152], ZigBee [176]). Because we focus on mobile platforms, we delve mostly into Bluetooth and Wi-Fi. Notably, due to its low power usage, low cost, well-structured application-level protocol [28], and presence in smartphones, Bluetooth Low Energy (BLE) is an increasingly popular choice in IoT devices. In fact, it is experiencing the most rapid growth in adoption in the embedded device industry [155], and is especially useful for devices made for casual or public use. Ultimately, however, we believe our browsing architecture is extensible to other forms of local wireless communication that are integrated in mobile platforms or computers and have a broadcast protocol.

Many initiatives and architectures have been proposed to form a Web of Things by integrating embedded IoT devices into the current open infrastructure of the Internet [73, 74, 172] and extending features like search and web analytics [116, 154]. Prior work has explored spontaneous interaction with nearby devices [55, 164], use of smartphones as universal device remote controls [84], device abstraction for resource-sharing among several interfaces [107], cloud services for real-time visualization of device data [170], and device browsing with platform-agnostic interfaces [23, 53, 141, 142]. While these are parts of the solution, none quite provides a generalized framework to enable users to browse nearby devices and retrieve rich user interfaces that interact with devices directly.

5.1.2 Discovering Content in Physical Space

In the theme of "webifying" everything, Google launched the Physical Web project to enable discovery of web content related to smart devices and environments in one's proximity [89, 160]. In this model, the Eddystone beacon protocol is used to embed a URL in broadcasts of BLE devices that can prompt nearby smartphones to navigate to specific web pages [69].

This achieves the goal of providing smart devices with rich, scalable web *content*, however it does not necessarily provide rich, scalable web *interfaces* because there is currently no cross-platform API that enables web pages to interact directly with a non-Internet-connected smart device from the phone's browser. Additionally, in practice, the system obfuscates device information — opting to simply present physically relevant web content to users. Recent work has further described challenges with Physical Web [136, 143]. Google has notably removed Physical Web support from mobile platforms [3, 120].

5.1.3 Bluetooth from the Browser

Web Bluetooth is a newly-drafted W3C standard and JavaScript API that enables connection with BLE devices from websites [161]. It is currently implemented in Android, but not iOS. Just like with native apps, discovery of ambient devices is difficult with Web Bluetooth alone. The current implementation requires that the user have prior knowledge of the device and its associated website prior to manually navigating to that particular site. Once opened, the page requires the user to choose the device to connect to it from a, possibly filtered, list of nearby peripherals. The user is tasked with figuring out which device is the appropriate one.

This security model is meant to closely resemble the classic Bluetooth pairing flow. However in practice, this disrupts the flow of operation, places burden of accuracy and authentication on users, and exposes a number of risks from malicious, careless, or uninformed actors. Malicious webpages may spoof the webpages of real devices, tricking users into pairing with those devices. Alternatively careless webpages may present any number of inappropriate devices to the user. Malicious or careless devices can simply use the same name or service ID as a real device and easily trick users into pairing it with legitimate webpages. Malicious users may purposefully initiate an inappropriate pairing of device and webpage, or careless and confused users may do so accidentally. It is both too permissive for security and too restrictive for usability.

Discovery could potentially be improved by combining with Physical Web — devices broadcast a link to a page that uses Web Bluetooth. However, because Physical Web obfuscates device information, users do not actually know which device linked to the page. And even though the browser could technically possess that information, it still requires the user to choose the appropriate device to connect to the page, of which they are not necessarily informed. As a result, this model is also disruptive for usability and unsafe for security.

5.1.4 App-ifying the Web

Since the introduction of the smartphone as a platform, developers have primarily chosen to design their software as native apps. Compared to websites, they would typically load quicker, and could make use of the additional space that the frame of a browser window would normally use. However, new and upcoming web and mobile standards are enabling fairly extensive app-like capabilities for websites [54, 68]. By adding a manifest with metadata about the content and providing some extra JavaScript to specify a "Service Worker", websites can explicitly cache content on phones for quick loading and offline access in a browser-less view, send push notifications from the websites' servers, and, in network-constrained scenarios, can queue requests for content to be fired when appropriate network connections are established even if the site is not opened.

Initiatives for web standards like Physical Web, Web Bluetooth, and Service Workers are decent starts, but they are currently missing key components that would actually make web-based device interfaces viable and usable, and enable seamless discovery, connectivity, interactivity, and persistence.

5.2 Browsing Architecture & Design

In the design of the browsing architecture, we aim to extend web standards in a manner that emphasizes usability and extensibility. We focus specifically on enabling discovery, connectivity, and interaction for devices in both ephemeral and persistent contexts, while ensuring a seamless experience between each of these stages.

We have chosen to design primarily for devices using BLE. While the architecture does not necessarily require BLE, it is useful to work through and address its challenges and patterns for local discovery, interaction, and security that emerge in the context of IoT, as many subsets of the same design decisions apply to other network systems, including Wi-Fi.

5.2.1 Discovery

To prompt smartphone users to interact with a peripheral device, the device can use a simple broadcasting protocol. In its broadcast parameters, the peripheral need only indicate a target location from which to receive the interface. This location can be expressed as a URL to a web app.

Like Physical Web, the browser can receive broadcasts from BLE advertisements. The browser can be compatible with peripherals that broadcast using Bluetooth's URI protocol [28], or Eddystone-URL protocol [69]. To accommodate size constraints in BLE advertisements, a long URL can be specified using a shortened address (e.g. a bit.ly URL).

When the phone detects a device URL and displays the result to the user, it should be transparent about the associated device information. This keeps users informed of the ambient devices in their vicinity and builds a reasonable understanding of the devices an interface could potentially connect with. As localization techniques improve, the browser could even possibly use augmented reality to more tangibly tie results to the devices in physical space [168].

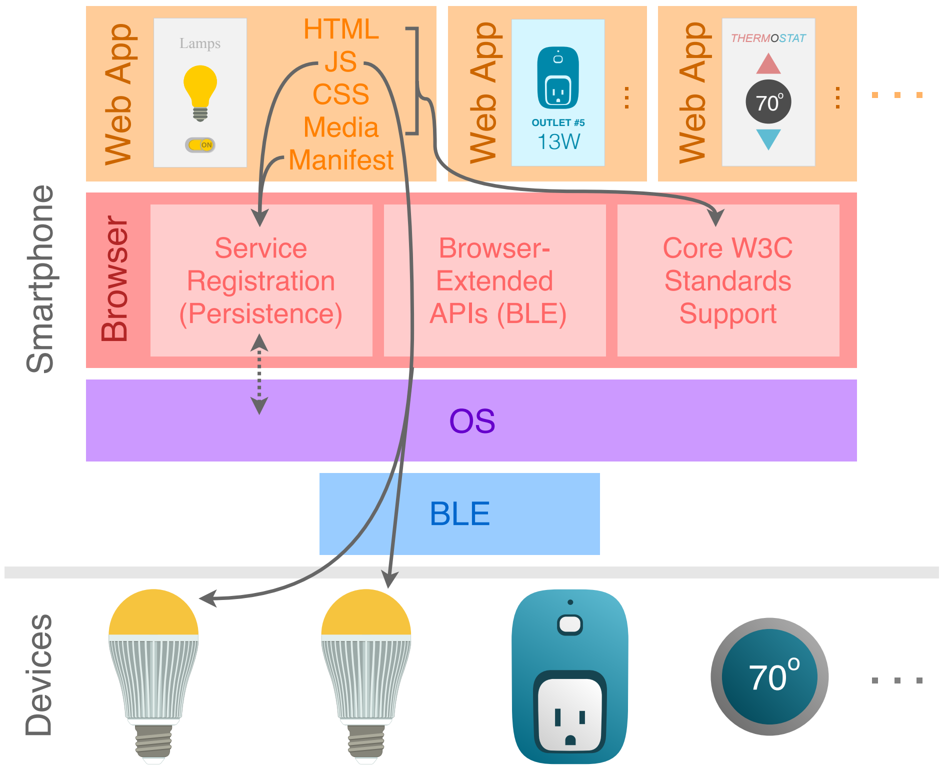


Figure 5.3: The Web App UI Model. Web apps may consist of typical web content: HTML, JavaScript, CSS, images, etc. An app can use a provided JS BLE API to interact directly with associated devices. By providing a manifest file for service registration, the web app can request special permissions and storage to enable persistent state for regularly-used devices.

5.2.2 Web Apps

Like ordinary websites, web apps should be developed using web standards (HTML, CSS, JavaScript, etc.), but they also need to be able to interact directly with devices, often through BLE. The browser can provide APIs to facilitate this. When a web app is retrieved, the browser can open the interface within its own context, providing the app with a set of standard library calls to native OS APIs. In this way, interactive web apps can interact directly with their associated devices over BLE, as depicted in Figure 5.3.

These interfaces may also wish to use other resources the smartphone can provide. For example, information about the location of the peripheral or the current global time may be difficult for the device to obtain, but is straightforward for a smartphone, which can expose the information for use by the user interface. For this purpose, access to items like time, GPS, acceleration, ambient light, pressure, and magnetic field can, with user permission, be provided to the web apps through browser-extended calls to the native API.

5.2.3 Device as a Web Resource (Origin Policy)

Web Bluetooth uses a user-select pairing model to associate a device with an interface. While done as an apparent security measure, this introduces more potential vulnerabilities to the device, user, and interface, while also presenting a disruptive user experience. To fix this, we can start treating "Web of Things" devices as actual things of the web—specifically, devices can be made resources of websites themselves.

This is possible if devices can declare the sites to which they belong or from which they can be accessed. Conveniently, in this architecture, devices are already broadcasting their web app's location in order to enable discovery. So the browser can at least take this as a declaration of origin. Since the browser would also be transparent about the devices associated with a web app listing, the user is informed of the devices it can access. When the web app is opened, the browser can enable BLE access exclusively for devices that have declared their association to the site or indicated some form of cross-origin access. This way, the need for the Web Bluetooth pairing model is eliminated. And developers, if they so choose, still have the ability to implement any additional authentication mechanism they normally would in the interface itself.

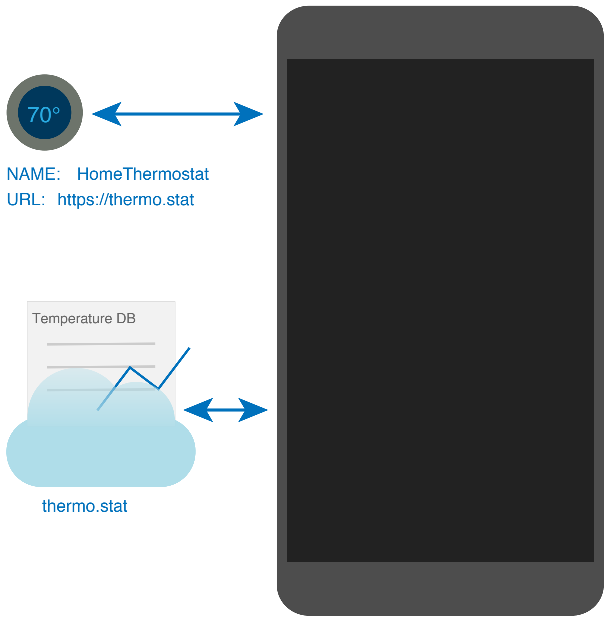
5.2.4 Persistence

While seamless discovery and interaction is nice, users will likely end up having regular interactions with a particular set of devices, like those in their home or office. In these cases, it is useful to have a method to save web apps (service registration) like caching, installing, or saving to home-screen, to be able to reload the UI quickly and work offline if necessary, as well as storing any local items like authentication data that might be used to connect with a device that the user owns. This is similar to the way Service Workers enable explicit caching of a website's content.

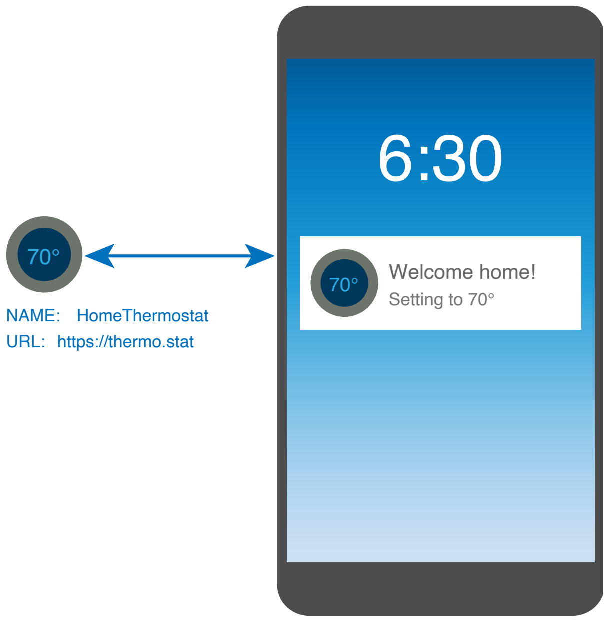
Furthermore, it would be useful for these apps to specify scripts to run in the background. For instance, a phone in the user's pocket could mule data for the device (with user permission), as in Figure 5.4a, or perform proximity based actions like turning on a light or setting the thermostat temperature to a preset setting when the user is near by, as in Figure 5.4b.

To make this work, the web app's script could specify a callback function to run when the browser, while scanning opportunistically in the background, detects the device. When this happens, the browser can pass a "device-detected" event to the web app's script and run the event's callback.

We have implemented examples of these scenarios in native Android apps, but the ability to run such background tasks for multiple web apps is currently limited by both Android and iOS, due to energy-saving optimizations. Service Workers, as implemented, do not yet support background operations. However Google and Apple's renewed commitment to progressive web app standards make such services seem feasible in the near future. We discuss this further in Section 5.5.2.



(a) Data Muling



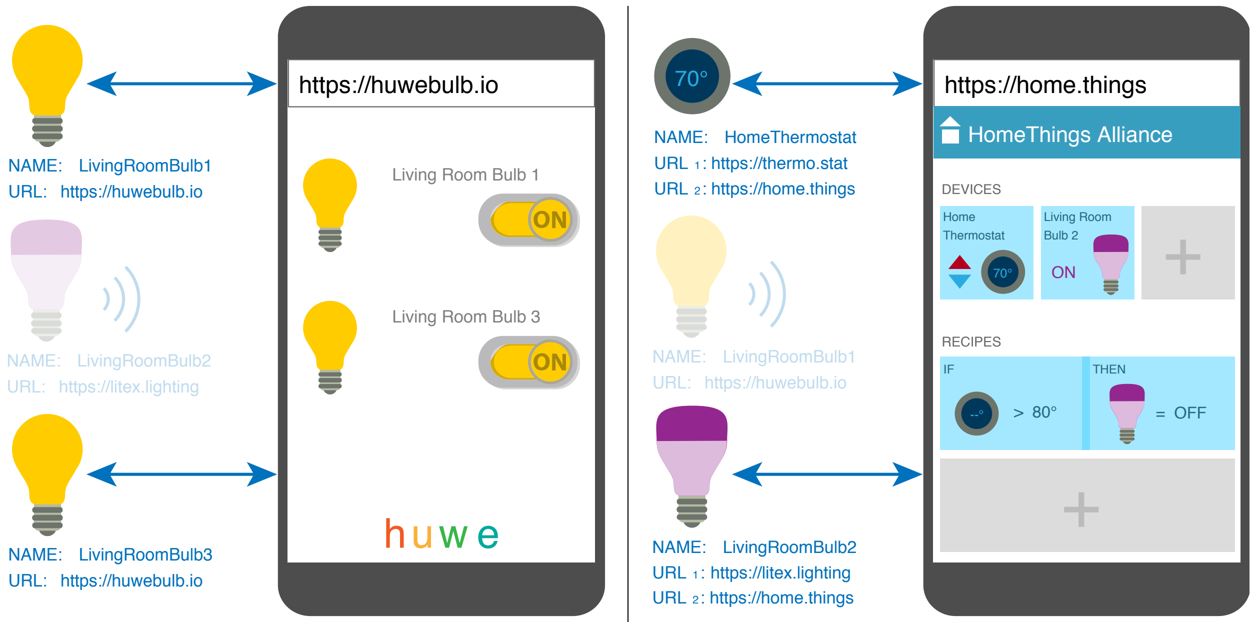
(b) Proximity-Based Action

Figure 5.4: Potential background services for persistent-use devices. If background mode is supported, an event could be passed to service-registered web app scripts when the phone detects the nearby device. In a data muling scenario (a), a phone in the user's pocket could mule data on behalf of the web app, with user permission. In a proximity-based action scenario (b), the web app could trigger some proximity based actions like turning on a light or setting the thermostat temperature to a preset setting when the user is nearby.

5.2.5 Aggregation

In some cases, simultaneous interactions with and between multiple devices may be desired from a single interface. We can, for instance, consider a scenario in which a person walks into a room with three power-metering devices and opens the browser on a smartphone. Instead of showing three instances of the power meter interface with each accessing only one corresponding device, the browser could show one common instance, which when opened can access any of the three. This concept can be extended to groups of multiple device classes—like the lights, speakers, and projector of a conference room being controllable from a single room-wide interface.

In order to enable such interactions while maintaining the standard of expected behavior between devices, user, and interface, we can provide web apps access to nearby peripherals that advertise its URL, instead of just to the single peripheral the user selects in the device list. An example of this is depicted in Figure 5.5a. We can also potentially extend to allow ecosystem web apps to claim devices from participant companies, either by having devices simultaneously broadcast a link to the ecosystem or by allowing the sites to specify cross-origin sharing whitelists for devices.



(a) Aggregation (b) Orchestration

Figure 5.5: Using a single interface with multiple devices. With aggregation (a), devices pointing to the same interface location could be accessible from a single instance of the interface, rather than opening a separate instance for each device. With orchestration (b), different classes of devices could be accessible from the same interface, which might be used to set up device-to-device interactions.

5.2.6 Orchestration

Different classes of devices could be accessible from the same interface, which could be used to set up device-to-device interactions. This could be done by providing extensions for "ecosystem" services like HomeKit [18] or IFTTT [85] for devices the user owns or has access to, as shown in Figure 5.5b.

In a different scenario, the browser can provide an interface for a stationary hardware gateway (or the gateway can provide its own interface) that allows users to setup connections between devices. The gateway can then handle operation of the services thereafter. Additionally, if a phone detects a device in its proximity that is out-of-range for a gateway, it could act as a bridge between the gateway and the device.

Alternatively, the browser could provide some interface that allows users to set rudimentary connections between devices. This can ultimately register a background service to facilitate device-to-device interaction through the phone.

In specific contexts, the browser could use location to enable customization for personal-, business-, or institution-based control and aesthetic. When a link for a verified device is detected in a participating organization's location, the user interface could be opened under the organizations own theme. This would minimize setup time for IoT device owners and enable quick deployability within controlled contexts. For instance, a hotel room could provide a single interface for all the devices in it with a theme for that specific hotel.

5.3 Browsing Implementation

As a proof-of-concept of key components of the proposed architecture, we have implemented: (1) a browser application for Android and iOS, (2) a broadcast specification for devices, (3) a cloud-hosted web service to scrape and obtain information about the peripheral's advertised interface location, and (4) an HTML/JavaScript API for web apps.

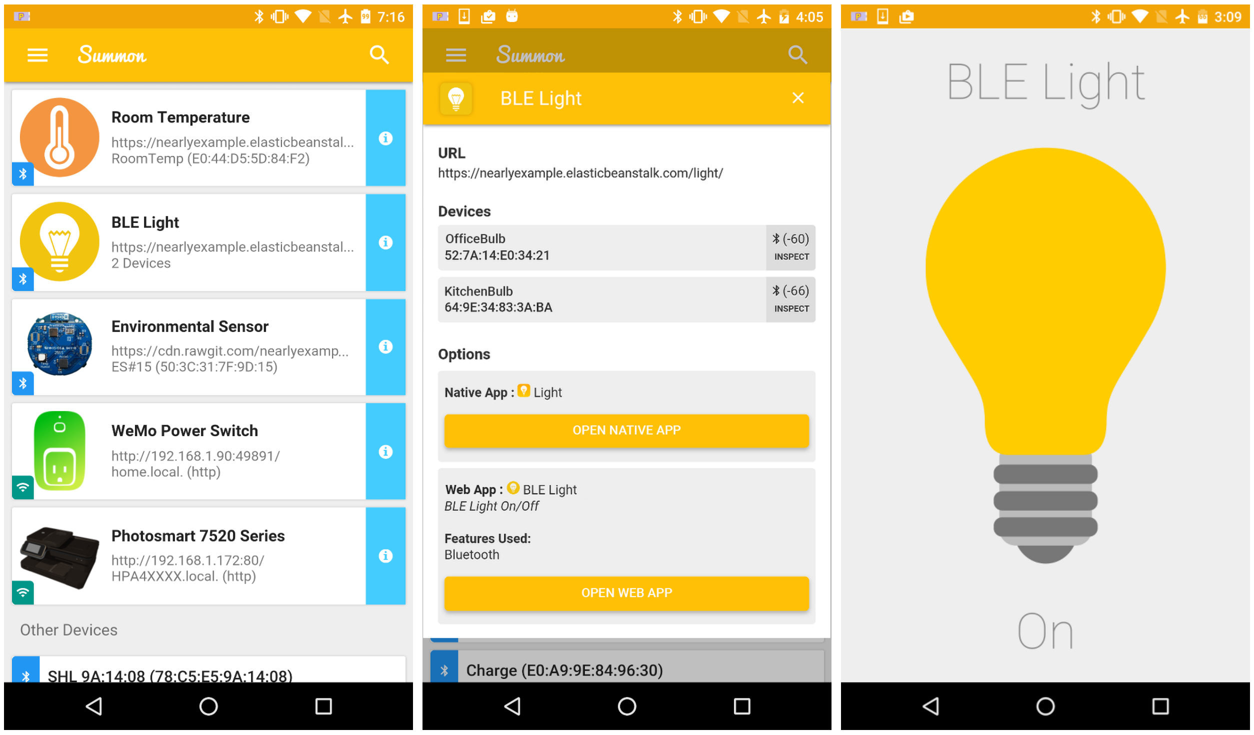
5.3.1 Browser App on Android and iOS

The browser is implemented as an application, called *Summon*, that runs on Android and iOS [100, 101]. It fulfills the role of the user-facing platform for discovering devices and viewing interfaces. The browser adds the advertised peripheral and its destination URL to a device manager list, and notifies the user of the presence of a peripheral with a linked interface. The smartphone user can open the device manager screen to view a list of the nearby peripherals and corresponding interfaces, as shown in Figure 5.6. When the user selects an item, an interactive web app is typically retrieved from the broadcast URL or from local cache, and is opened within a browser-controlled context. Alternatively, a native app or regular website may be opened.

For web apps, *Summon* provides a controlled context and native API bindings that are derived from core Apache Cordova frameworks [11]. Systems like Apache Cordova and PhoneGap [5] allow developers to write native applications for mobile phones in HTML that can access the phone's native API using JavaScript. The browser app provides a special Cordova-based context in which to open web content. In this context, the web content can access and utilize a set of JavaScript libraries that the browser provides as native smartphone APIs. In implementation, web apps are effectively websites that make use of these APIs. In particular, the provided BLE API allows web apps to interact directly with devices. Web apps can also request permission to use APIs that access smartphone sensor data, storage, and information.

Notably, *Summon* extends its own APIs, as newer web standards—particularly Web Bluetooth and Service Worker APIs—had not yet matured at the time of initial implementation. A Web Bluetooth-compatible shim-layer was added at a later stage.

Users are able to configure how and to what extent their smartphone is utilized as a user interface platform. They can enable or disable caching, choose which radios the browser can use to discover devices (BLE / Wi-Fi via mDNS), and allow or reject permissions associated with each web app. Additionally, users can filter by device or UI name, as well sort UI listings by device discovery time, device signal strength, and UI popularity.



Summon

Room Temperature
https://nearlyexample.elasticbeanstal...
RoomTemp (E0:44:D5:5D:84:F2)

BLE Light
https://nearlyexample.elasticbeanstal...
2 Devices

Environmental Sensor
https://cdn.rawgit.com/nearlyexamp...
ES#15 (50:3C:31:7F:9D:15)

WeMo Power Switch
http://192.168.1.90:49891/
home.local. (http)

Photosmart 7520 Series
http://192.168.1.172:80/
HPA4XXXX.local. (http)

Other Devices

SHL 9A:14:08 (78:C5:E5:9A:14:08)

Summon

BLE Light

URL
https://nearlyexample.elasticbeanstalk.com/light/

Devices

OfficeBulb 52:7A:14:E0:34:21	(-60) INSPECT
KitchenBulb 64:9E:34:83:3A:BA	(-66) INSPECT

Options

Native App : Light
OPEN NATIVE APP

Web App : BLE Light
BLE Light On/Off

Features Used:
Bluetooth
OPEN WEB APP

Charge (E0:A9:9E:84:96:30)

BLE Light

On

(a) (b) (c)

Figure 5.6: ***Summon*— Browser implementation for mobile phones.** If the browser detects a device that links to an interface, it is listed in the device browser (a). If multiple devices link to the same interface, they are grouped under one list item. When the user selects an item, its web app (c) is opened. Since the interface is opened within a controlled framework (instead of a regular browser), it can use provided JavaScript bindings to behave like a native app and interact with associated devices. If a device's native app is detected on the phone, it is opened instead. The UI options, linked devices, and list of features used by the web app are visible in detail view (b).

5.3.2 Destination Resolution

When the browser application receives a broadcast URL, it determines if the URL is a website or a fully-featured web app that requests use of the browser's extended APIs. It also checks to see if a native application for the device exists and is already installed. If the device does not broadcast a URL or the smartphone is not connected to the Internet, the browser checks its cache of web apps to check if any are already associated with the device's address or advertisement profile. This way, when a user makes a selection, the browser can take the appropriate action, whether that be opening a native app, or retrieving a web app from the Internet or local cache.

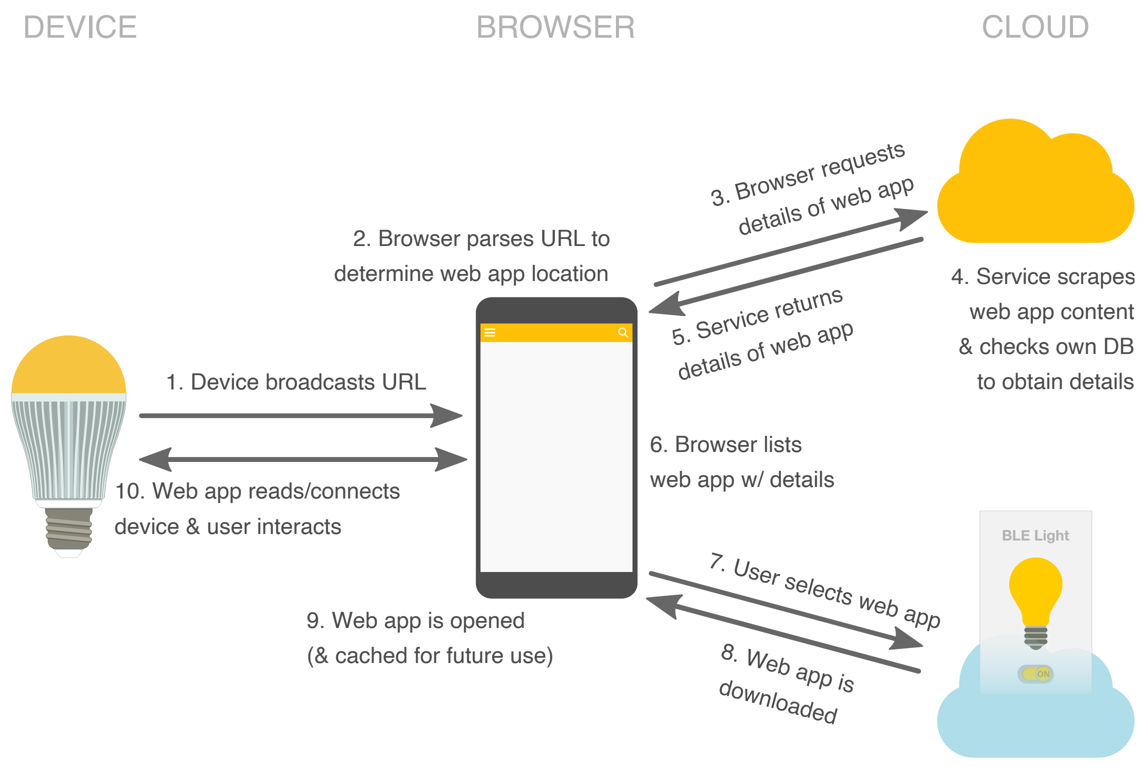
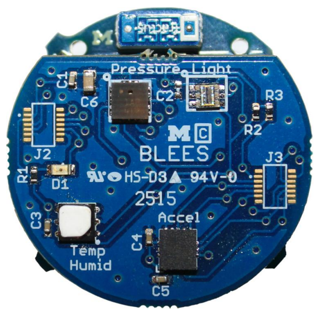
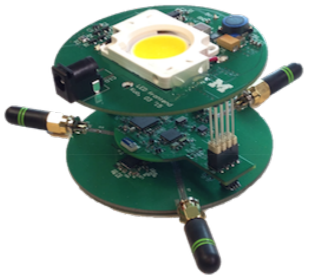
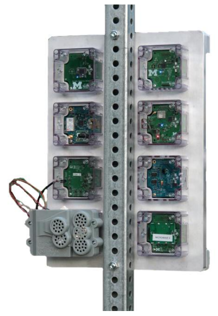

DEVICE
BROWSER
CLOUD
1. Device broadcasts URL
2. Browser parses URL to determine web app location
3. Browser requests details of web app
4. Service scrapes web app content & checks own DB to obtain details
5. Service returns details of web app
6. Browser lists web app w/ details
7. User selects web app
8. Web app is downloaded
9. Web app is opened (& cached for future use)
10. Web app reads/connects device & user interacts
BLE Light
ON

Figure 5.7: Overview of the implemented flow of device discovery and presentation between the device, the mobile browser, and the cloud. With help from an implemented destination-resolution cloud service, the browser can obtain detailed information on the interface at the device's advertised URL, and present it to the user.

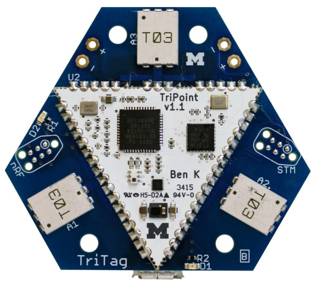
To assist the browser in obtaining information on the interfaces corresponding to URLs advertised by devices, we have set up a link resolution web service. The service sits in the cloud and responds to requests from the browser application. Figure 5.7 depicts the flow of this process. When the browser discovers a device, it sends a request containing the broadcast URL to the service. The service first resolves short URLs or any URL redirects in order to determine the full-form address of the actual web app location. It then scrapes the linked content, and returns useful data like the website or web app's title, description, potential native applications to check for, icon image location, browser-extended APIs used, and requested smartphone permissions. The service also stores this information in a database to provide a rapid response when the same URL is requested again from any phone. Additionally, it monitors popularity and usage of each web app, which can be used to sort web apps by relevance in the browser. While not fundamentally vital to the design of the architecture, the service provides useful information that will better inform the user about the relevant web apps prior to potentially opening one of them, while reducing latency in response time.


(a) Environment Sensor (BLEES [97])


(b) Smart LED Light (Torch [95])


(c) Smart City Platform (Signpost [4])

(d) CO Breathalyzer (Monoxalyze [2])


(e) Localization Tag (Polypoint [91])

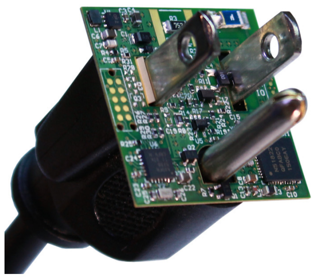
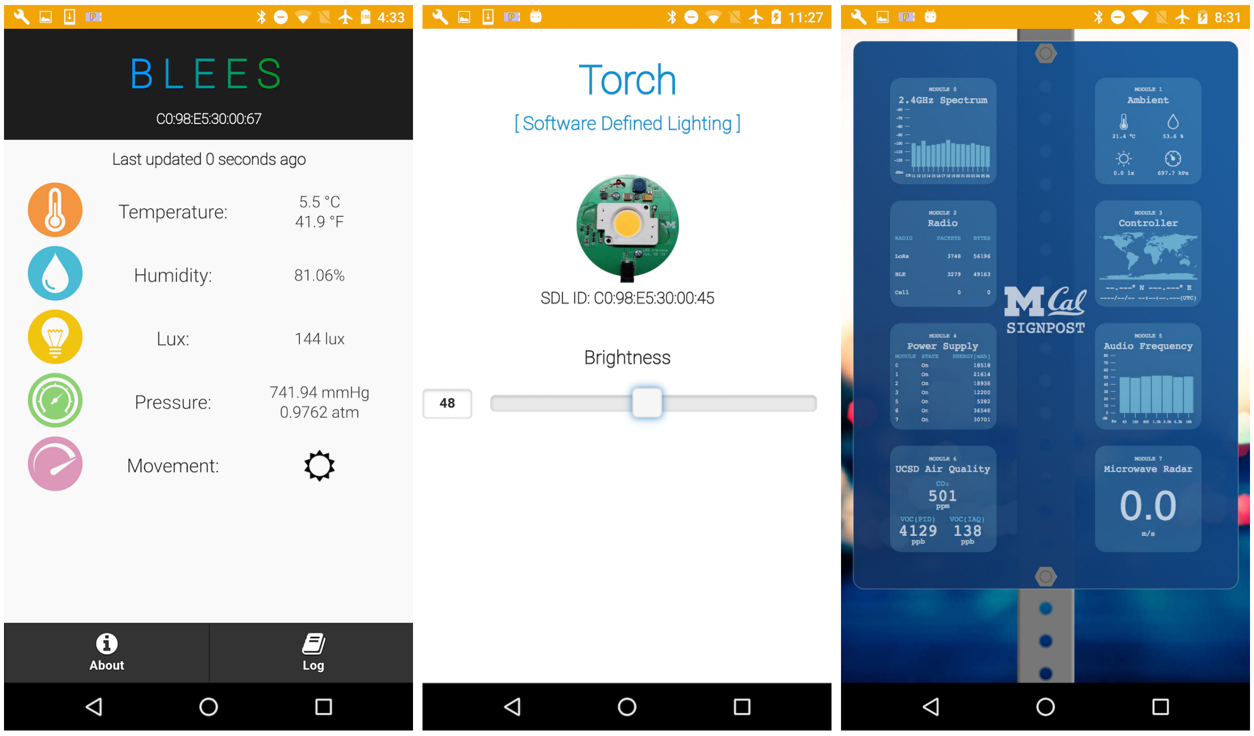

(f) Power Meter (PowerBlade [43])

Figure 5.8: A selection of real "browsable" devices

5.3.3 Caching

To further enable device interactions without requiring connection to the Internet, the implemented browser can also cache web apps. By default, the browser caches all of the web app's details when it is originally listed in the browser and caches the web app's resources when it is first opened. This is essentially *Summon*'s version of service registration. Unless specified otherwise in the HTTP header of a web app, the browser caches all web content for an indefinite period of time until the user manually clears it or until app cache capacity is reached, at which point the least-recently used resources are replaced. If a known device is detected again while the phone does not have Internet connection, the smartphone can retrieve the web app details and the web app itself from memory. When connection is available, the cached UI can be loaded if header-request validates that online resources have not changed. This improves load time and data usage. The cached UI details also allow for quicker response when scanning for devices when the phone is online and quicker loading of the actual UI when it is used often.


BLEES
C0:98:E5:30:00:67
Last updated 0 seconds ago
Temperature: 5.5 °C 41.9 °F
Humidity: 81.06%
Lux: 144 lux
Pressure: 741.94 mmHg 0.9762 atm
Movement:

Torch
[Software Defined Lighting]
SDL ID: C0:98:E5:30:00:45
Brightness
48

(a) (b) (c)

Figure 5.9: Subset of web apps for real devices. All interact directly with corresponding devices (Figures 5.8a to 5.8c) directly over BLE using a browser-defined JavaScript API.

5.3.4 Peripheral Devices

During the study, a number of peripherals have been configured to be discoverable by the browser app, including embedded devices like power meters, software-defined lighting controllers, indoor localization systems, environmental sensors, smoking cessation meters, and BLE tags. Some of these systems are shown in Figure 5.8. Examples of both BLE- and Wi-Fi-based peripherals have also been successfully set up on Android smartphones and tablets, Nordic nRF51/nRF52 and Espressif ESP32-based embedded devices, Raspberry Pis, and Linux and Mac computers.

The devices specify a URL in a BLE broadcast linking to corresponding user-facing content that can interact directly with the device without the peripheral requiring Internet access. While *Summon* is primarily used with BLE devices, the browser also detects and lists URLs that are broadcast from devices on the local Wi-Fi network via mDNS, a zero-configuration network service discovery protocol. Conveniently, some network-connected devices, like printers, already advertise web interfaces using this protocol, and are readily visible in the browser.

5.3.5 Web Apps

Concurrently, a number of web apps have been built specifically for use with our *Summon* browser application using standard web tools. Device advertisements link to the corresponding web app, which may be fetched from an online location or from the phone itself (if the interface is cached). Once loaded, the interface can enable interaction with the device and use native smartphone features via browser-extended JavaScript APIs. Notably, web apps have been made for each of the embedded devices listed earlier, some of which are shown in Figure 5.9. Figure 5.10 contains code for a simple web app, which requests and uses the *Summon* Bluetooth API to enable interaction with its corresponding light bulb device.

During a two year period, approximately 20 embedded developers created web apps for use with corresponding embedded devices and the *Summon* browser received about 1000 downloads on iOS and Android. The set of embedded developers were primarily from the academic community and included undergraduates, graduates, and faculty from multiple institutions. There was also participation from a couple of interested hobbyists. Developers chose to use the browser for their device interaction needs based on discussions and demonstrations that made apparent the relative ease of the development and deployment process for device user interfaces. No formal recruitment campaign was involved. Developers were provided online documentation, code samples, and informal tutorials to help create their web-applications and configure devices [102]. While most of the developers had little or no web and mobile app development experience, they were able to create fully functioning and moderately aesthetic interfaces that successfully enable user interaction with the devices.

5.4 Browsing Analysis

For the purpose of evaluation, we test the browser with a range of devices. Along with observing the general functionality between the browser, user, and devices, we quantify some tradeoffs between web apps and native apps. We explore the mechanics of device discovery and impacts on presentation.

5.4.1 Paradigms of Real Applications

For a high-level qualitative assessment, we have made the implemented browser app, APIs, and template code for web apps available to embedded developers to create prototype interfaces for their devices. By using standard web tools (HTML, JavaScript, CSS, etc) to create web apps and making slight modifications to appropriately configure their respective devices, the developers have been able to create easily discoverable, powerfully interactive interfaces. A subset of the devices and web apps are shown in Figure 5.8 and Figure 5.9. The ability to directly control devices and offload real-time data have been notably appealing to developers, particularly for those of energy- and network-constrained devices. The following real examples describe interaction paradigms for web apps and devices supported in the browser implementation.

```html
<!DOCTYPE html>
<html>
  <head>
    <summon request="bluetooth"/>
    <title>BLE Light Bulb</title>
  </head>
  <body>
    <h1>BLE Light</h1>
    <img src="bulb.png"/>
    <button>Off</button>
    <script src="index.js"></script>
  </body>
</html>
```

(a) HTML - index.html

```javascript
var ble, state, device;
var serviceID = "ABCD";                                       // Service ID of bulb's light switch service
var stateID = "1234";                                         // Characteristic ID of switch's state data (on/off)
var button = document.querySelector("button");                // <button> element

// When UI finishes loading -> onReady
document.addEventListener("ready", onReady, false);

// UI Ready Event Callback
function onReady() {
  ble = summon.bluetooth;                                     // get Bluetooth API from Summon
  ble.connectDevice(onConnect);                               // scan for device; when connected -> onConnect
}

// BLE Device Connect Callback
function onConnect(device) {
  ble.read(device.id, serviceID, stateID, onRW);              // read state of light; when read -> onRW
  button.addEventListener("click", onClick, false);           // set click event: if button clicked -> onClick
}

// Button Click Callback
function onClick() {
  ble.write(device.id, serviceID, stateID, [!state], onRW); // set light on or off; when written -> onRW
}

// BLE Read/Write Callback
function onRW(data) {
  state = data[0];                                            // get bool state from returned data
  button.innerHTML = state ? "On" : "Off";                    // update button state
}
```

(b) JavaScript - index.js

Figure 5.10: HTML & JavaScript of an Interactive Web App. When opened in *Summon*, this simple UI connects to the BLE light bulb device that linked to it, reads state from the device, and writes to the device to toggle the light when a button is clicked.

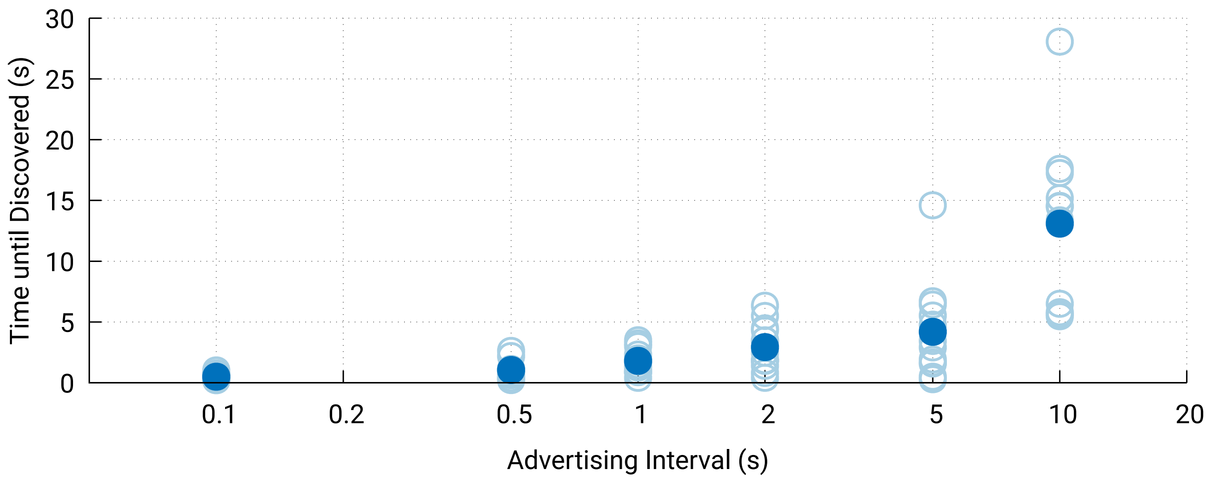


Figure 5.11: Device discovery latency for varying advertising intervals. The solid point is the mean of recorded latencies at each interval. Measurements taken on a first-generation Google Pixel.

Advertisement-only. The 1-inch round environmental sensor system shown in Figure 5.8a, continuously broadcasts BLE advertisements, sending out its shortened web app URL and sensor readings in alternate packets once a second. The browser detects the URL advertisement, obtains data for the linked URL from the browser's cloud service, and lists the web app in the device browser. When the user selects the listing, the web app, shown in Figure 5.9a, loads and immediately begins receiving and parsing the sensor reading advertisement packets to retrieve temperature, humidity, illuminance, air pressure, and motion data from the device. The display is updated with the corresponding readings in real time. If a user enters a room with one of these environmental sensors on the wall and opens the web app, updated data is typically displayed every 2 to 6 seconds (accounting for dropped packets), while taking approximately a quarter of a second to parse and format the data when each packet is received. Running on a coin cell battery, the environmental sensor device has a lifetime of approximately 6 months while continuously broadcasting and taking sensor readings.

Connection. The software-defined lighting system shown in Figure 5.8b, continuously advertises its URL. After the browser discovers the device and its corresponding web app, shown in Figure 5.9b, is opened, it is able to automatically connect with the device over BLE. Connection typically occurs within one second of the web app opening. The web app obtains the light's current brightness level and displays it on a slider interface. Whenever the user uses the slider interface to change the brightness level, the new value is immediately written to the device, and the light's brightness is changed accordingly. Since the device is in connection mode with the phone, there is low transmission latency and light state is visibly updated in approximately half a second.

Multiple Devices. The system shown in Figure 5.8c is a solar-powered city-scale sensing platform that is mounted on a signpost. It contains 6 sensor modules, a control module, and a power module, each of which send its own data via BLE. Because each module advertises the same URL, the browser aggregates them all under one listing in the device browser. The user can view all associated devices accessible by the web app in the listing's detail view. When the web app is opened, as in Figure 5.9c, it can scan for, obtain data from, and display appropriate interfaces for each associated module.

5.4.2 Device Discovery

The browser's ability and performance in discovering BLE peripherals depends on the advertisement rate, physical proximity, and transmit signal strength of the device. Figure 5.11 depicts how much of an effect the advertising rate has on the speed of discoverability, for intervals ranging 10ms to 10s. By default, the browser app displays results in the order in which devices are discovered. Devices with higher advertisement rates, as well as closer proximity and higher transmit signal strength, will have noticeably higher chances of early detection.

However, the only metric the phone can provide when it discovers a device is the received signal strength (RSSI). RSSI of BLE devices vary significantly over time due to transmission parameters, environmental factors, and interference. Ordering by device RSSI can still be useful, but doing so in real-time yields an unstable presentation to the user. User feedback has indicated that even slight changes to the ordering of the list of devices in real-time can cause significant user error when attempting to select a list item. To accommodate users who desire it, a sorting scheme using a sliding average of RSSI is presented as an option in the browser app.

5.4.3 Web App Size

While browsing, the user would likely be accessing and downloading web apps relatively often. This leads to concerns about Internet data usage, especially when the smartphone uses cellular network data. Most web apps developed during the deployment period have been on the order of 10 kB - 100 kB each. While the average web site is over 1 MB [81], the browser's environment seems to be more conducive of higher quantities of interfaces for short connections and speedy interactions. However, even when web apps contain rich elements typically seen on the average flashier websites, the usage impact is typically much less than native apps. A 2012 study revealed that the average mobile phone application was, at that time, 23 MB for iOS and 6 MB for Android, increasing 16% and 10%, respectively, over a 6 month period [1]. To meet growing demands in the past couple years, both Android and iOS app stores increased their app size limits to 4 GB (with a 100 MB network delivery limit) respectively. Without overhead of re-imported native libraries, web apps can also be cached trivially for repeated or offline use.

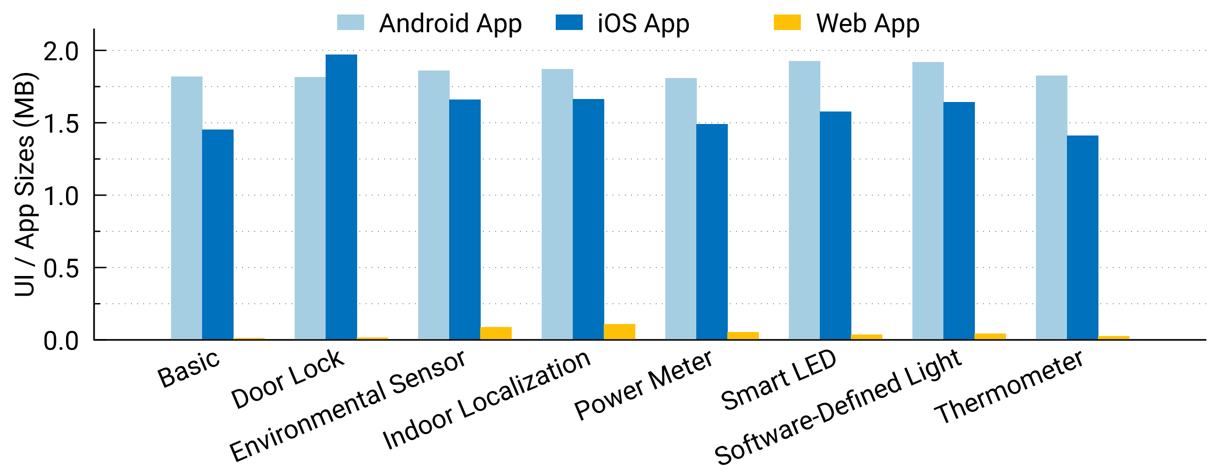


Figure 5.12: Size of example web apps vs native apps. Web app size accounts for bare web resources—HTML, JS, CSS, images, etc. Browser-provided JavaScript APIs allow web apps to use native smartphone features like BLE at run time. Because native apps repackage large commonly-used libraries into their binaries, they are significantly larger than web UIs.

Apps for casual interaction can be quite lightweight, but each must often re-import its own instance of commonly-used libraries, which ultimately bloats the size of the app binary. With the support of web-based interfaces, the browser app reduces size requirements. Figure 5.12 shows that the size of web-based interfaces are low ($\tilde{k}$Bs) when compared to the size of native apps ($\tilde{M}$Bs). In an experiment measuring the data usage of repeatedly opening a small web app with and without cache, the first opening consumed 92 kB of data both times. Without cache, the full 92 kB were used in each subsequent opening. With cache, 4 kB were used for each subsequent request to detect if any changes were made to the web app. While web apps sizes are already low, caching further diminishes impact on network data costs, especially when compared to those of installing full apps. This experiment illustrates that caching keeps the data usage cost of repeated web app downloads at a nearly negligible amount, all without the permanent memory requirements of an installed native application. While, still, native apps can provide very rich user interfaces and could actually reduce data costs when devices have high repeated use, we foresee a great increase in ambient interaction use cases in the near future of the Internet of Things, and find that this browser model will offer lower memory and data costs for those cases.

5.4.4 User Action

As an examination of seamlessness and ease of use in our approach, we consider a set of common tasks that the browser would be expected to handle. We perform a rudimentary analysis by calculating the number of gestures required to accomplish each task, and compare with Physical Web, Web Bluetooth, and native apps. This is depicted in Table 5.1.

Steps for...	Physical Web	Web Bluetooth	Phys. Web + Web BT	Native App	*Summon* (Our Browser)
Ambient device/UI discovery	- Open list screen - 1 (obfuscates device info) ≈ 1 gestures	N/A (no ambient discovery) $\approx \infty$ gestures	- Open list screen - 1 (obfuscates device info) ≈ 1 gestures	N/A (no ambient discovery) $\approx \infty$ gestures	- Open list screen - 1 ≈ 1 gestures
First-time setup of a device/UI requiring an associated account	N/A (no device interaction) $\approx \infty$ gestures	- Open browser - 1 - Type URL - *u* - Press go/enter - 1 - Type username - *n* - Type password - *p* - Press signup/login - 1 - Perform trigger action- 1 - Pick device in prompt - 1 - Press “pair” - 1 - Confirm/add device - 1 $\approx 7+u+n+p$ gestures	- Open list screen - 1 - Press listing - 1 - Type username - *n* - Type password - *p* - Press signup/login - 1 - Perform trigger action- 1 - Pick device in prompt - 1 - Press “pair” - 1 - Confirm/add device - 1 $\approx 7+n+p$ gestures	- Open app store - 1 - Enter search query - *q* - Press search/enter - 1 - Find app, tap install - 1 - Open app - 1 - Type username - *n* - Type password - *p* - Press signup/login - 1 - Confirm/add device - 1 $\approx 6+q+n+p$ gestures	- Open list screen - 1 - Press listing - 1 - Type username - *n* - Type password - *p* - Press signup/login - 1 - Confirm/add device - 1 $\approx 4+n+p$ gestures
General interaction	N/A (no device interaction) $\approx \infty$ gestures	- Open browser - 1 - Type URL - u - Press go/enter - 1 - Perform trigger action- 1 - Pick device in prompt - 1 - Press “pair” - 1 $\approx 5+u$ gestures	- Open list screen - 1 - Press listing - 1 - Perform trigger action- 1 - Pick device in prompt - 1 - Press “pair” - 1 ≈ 5 gestures	- Open app - 1 (app autoconnects) ≈ 1 gestures	- Open list screen - 1 - Press listing - 1 (UI autoconnects) ≈ 2 gestures

Table 5.1: User gesture analysis. The table compares the steps of actions necessary to perform tasks of discovery, setup, and interaction with BLE devices using Physical Web, Web Bluetooth, native app and our browser. Our approach generally requires fewer user gestures to accomplish tasks than the alternative methods, while also enabling better ambient discovery of devices and their UIs. Variables u, n, p, and q represent gesture counts when typing URL, username, password, and search query respectively.

We first examine how ambient device and UI discovery is facilitated. In our model, simply opening the browser will reveal ambient devices and their corresponding interfaces. While users can open a similar list screen from a Physical Web notification, device information is obfuscated.

In the general interaction scenario, an interface for a known device would need to be opened and receive data from the device. When a Web Bluetooth-enabled web page is opened, it requires that the user perform a trigger action, like pressing a button on the page, before then prompting the user to select the appropriate device. This is in addition to requiring the user to manually navigate to the device's page. Using Physical Web to direct users to the Web Bluetooth UI helps to reduce some work for the user. However, our implementation would allow a UI opened from the browser list screen to immediately receive data from its associated device. If a native app is already installed, it likely requires the same or less user action.

Next, we explore the actions a user might need to take to open a new UI for the first time for a device that requires a user to set up an account and link the device to their account. In the native app model, a user would need to have knowledge of the device's app and install it from the app store prior to creating a user account, scanning for the device,

	Energy	% Battery
LCD Screen	24.00 J	~0.0563%
App Processes	7.18 J	~0.0168%
Wi-Fi	0.88 J	~0.0021%
Bluetooth Low Energy	0.72 J	~0.0017%
Total	**32.78 J**	**~0.0769%**

Table 5.2: Phone energy usage while discovering devices. The listed total is an average of 10 one-minute trials on a Motorola Nexus 6 with a 3200mAh battery while in a setting with 5 URL-beaconing peripherals (4 BLE, 1 mDNS) and 5 other broadcasting peripherals (2 BLE, 3 mDNS), and with screen at lowest brightness. Measurements are recorded using PowerTutor and Trepn Profiler.

and adding it to their account within the app. The Web Bluetooth interface would face the same hurdles as its general interaction case. Our browser implementation would allow users to jump right into account setup and device confirmation immediately after opening the interface from the list screen.

5.4.5 Energy Usage

Peripherals do not require their own connection to the Internet via an on-board Wi-Fi or GSM chip, or through an external hardware gateway. They can, instead, leverage the smartphone's network connectivity, while using the energy efficient, short-distance communications of BLE. As a result, our architecture can help eliminate high power costs generally associated with such communication systems on peripherals. In the architecture, BLE peripherals need only operate in the low power advertising mode. NRF51822, the most common BLE chip used on our developers' devices, averages <80 µW (dependent on transmit power and payload), when advertising once per second [125]. It should be noted that most BLE peripherals already advertise, so adherence to our protocol does not likely impact typical consumption significantly.

While power efficiency has not been a primary consideration in the browser implementation, it is useful to know a rough breakdown of energy dependence of individual components of the system, particularly with the communications systems and application processing. To obtain a rough breakdown, approximate power measurements are taken using the PowerTutor tool [173] for the Android platform. The tool offers an accuracy within 0.8% on average with at most 2.5% error. Additionally, Qualcomm's Trepn Power Profiler [135] is used to help determine energy usage distribution among hardware components in finer detail. Table 5.2 depicts the average usage breakdown over 10 one-minute trials on a Nexus 6.

In this evaluation, usage is broken into four categories: LCD, Wi-Fi, BLE, and app processing. As with many native apps, most of the energy is consumed by the LCD screen, taking up nearly 75% of overall consumption. The *Summon* browser only operates in the foreground, so the application must be open and on screen to be active. Application processing consumes approximately 20%, which is mostly spent creating and manipulating visual elements in the graphical interface of the browser. The remaining $\tilde{5}$% is consumed by the BLE and Wi-Fi radios, with a slightly higher percentage attributed to the latter. Cellular network connectivity is inactive on the tested mobile device, but generally consumes about as much as Wi-Fi when used instead. Unless the phone screen remains on throughout the day, the browser's consumption does not raise major concern. Additionally, this should have diminishing impact as phone batteries continue to improve.

5.5 Discussion

During this study, we have gained insights from developers and users about their experiences, and have encountered and contemplated various technical challenges. In this section, we more deeply discuss questions regarding these challenges, examine how our system has evolved over time, and explore methods to better improve.

5.5.1 Bluetooth and Denial of Service

BLE peripherals have two primary modes of operation: advertising and connected. Peripherals typically spend most time advertising to broadcast identifying information. However, for interaction, it is often required that the phone connect to the peripheral. During the deployment, developers noticed that many BLE software stacks do not allow simultaneous advertisements and connections. Moreover, most stacks only allow a peripheral to be in one connection at a time. This means connecting to a BLE device could create a denial of service for other phones. While a one-to-one connection is useful for personal devices like the smoking cessation breathalyzer, it can limit casual interactions with ambient devices designed to be accessible to multiple users.

Possible solutions to this problem include imposing connection time limits either on the peripheral device itself or in the our browser application. While these solutions would help with the problem, starvation is inherent to a peer-to-peer networking architecture like BLE. Another approach could be to have phones rebroadcast connected peripherals' information and virtualize their services to other nearby users. This scheme would also allow the phone to absorb the energy cost of broadcasting while in a connection, instead of the peripheral device, which is more likely to be energy-constrained. However, this solution weakens the notion of spacial locality upon which our system relies, and raises the security concerns of untrusted phones facilitating connections. In order to prevent denial of service, developers should more often utilize interaction models that cut out or significantly reduce time in connected mode.

5.5.2 Feasibility of Background Service

Due to how the mobile OSes (both iOS and Android) regulate native applications' background service tasks, particularly with respect to services running BLE, web apps currently must share from the quota of background service execution time that is allotted to the browser application (bursts of approximately 5s periodically at unspecified intervals on iOS [16]). Native apps are able to claim their own background service that would, hypothetically, be able to run as long and often as the entirety of the browser app's allotment for all web app background services combined. This means that a native app can perform tasks like communicating with a corresponding device in the background for longer than individual web app. For example, we have successfully implemented a native app version of a background data muling service similar to the one in Figure 5.4a, albeit on a less restrictive Android 5 device. With both Apple and Google invested in Progressive Web App standards, it is possible the respective mobile OSes will evolve to offer better support and web apps could eventually gain a larger allotment of such OS resources.

5.5.3 Adaptation of Origin Policy

While we have developed and implemented a stand-alone browser application that successfully accomplishes most of the design goals of the browsing architecture, traditional browsers may benefit from at least integrating the key notion of treating devices as web resources. This device origin policy would still be applicable and useful in the typical type-and-go web browsing model. Friction can be drastically reduced in web apps that make use of Web Bluetooth by allowing devices to claim their origin, restricting website's access to specified/approved origins, and providing transparency to users rather than making them choose devices blindly—perhaps in the form of a permissions request prompt with a clear listing of associated devices that is displayed once, akin to geolocation request prompts in the browser.

5.5.4 Extending the Architecture

While we focus primarily on a mobile implementation that enables discovery of user interfaces for BLE devices, the architecture has been applied to a number of different platforms, contexts, and networks. For instance, the browser has been implemented and is regularly used on MacOS—the same web-apps can be discovered, enabling interaction with devices around the desktop using BLE and Wi-Fi. In the Android implementation, an API was provided to support discovery and communication over NFC. For settings in which Internet-connectivity is limited, the browser had also supported a custom BLE service that allows downloading of user interfaces directly from the device. Chapter 6 also explores how augmented reality could be used to improve the browsing approach by making discovery a more tangible user experience.

5.6 Summary

We have introduced a mobile browsing architecture that extends web standards to provide a seamless and open approach to discovering, connecting, and interacting with nearby "things" for both ephemeral and persistent use cases. The approach enables direct local access to and smartphone-mediated interaction with proximal devices, while embracing modern web technologies and open standards, and taking advantage of native smartphone capabilities. Through the implementation and distribution of *Summon*, a smartphone browser application that employs this architecture, we have learned that the approach scales better than current models of mobile-based interactions, particularly with low-power embedded devices, and provides intuitive, natural functionality for both users and developers. Insights from the deployment of this architecture can be generally applied in the creation and improvement of "webification" tools and in the development of a new generation of interfaceable devices.

To further facilitate the "webification" of the Internet of Things and aid the development of future web standards, we aim to help expose which architectural design choices achieve the highest level of simplicity, usability, comprehensiveness, and comfort for both the user and developer, while also ensuring enhanced reliability and security. If deployed on the worldwide network of smartphones, our approach could finally provide a user-interfacing solution that adequately enables, supports, and handles an increasingly global, robust, and intimate Internet of Things.

Chapter 6

Browsing Things in Mixed Reality

The advent of the Internet of Things (IoT) has brought with it a slew of new technologies and devices that together transform ordinary places into smarter, more connected spaces and environments. As noted in previous chapters — to facilitate interaction with smart devices, the trend for developers has been to create soft graphical interfaces in the form of smartphone apps that can potentially enable more comprehensive or complex levels of control, while drastically reducing, simplifying, or removing physical interfaces. Voice-based virtual assistants, which are accessed through smartphones and stationary smart home speakers, provide an additional means of device control.

While these modalities can provide for useful and novel functionality, they form a physical and cognitive barrier between the user and the object with which he or she intends to interact. This denies the user a measure of intuitive connection and tangibility that direct manipulation of tactile on-device interfaces more often elicit. Furthermore, these modalities often complicate fine-grained, filtered, and grouped control of devices with structural semantics and complex abstractions, or lack functionality altogether [32]. Additionally, these systems are still challenging for visitors to discover and access, and makes performing basic ephemeral tasks, like turning on a light, nontrivial.

In the last chapter, we begin to rectify some of these lapses with *Summon*, using network-based discovery of devices and dynamic loading of device-linked interactive web user interfaces. To further eliminate barriers between the user and device, we can look to smartphone- and tablet-based augmented reality (AR), or mixed reality (XR). In such mobile-based AR, virtual content can be overlaid onto a video feed of the real world from the perspective of the smartphone's camera. Having experienced a significant growth in capability and reliability in recent years, mobile AR provides tools that can, perhaps, be used to provide a more intuitive, tangible, and creative approach to interaction.

In this chapter, we explore the use of mobile AR in both *discovery* of and *interaction* with IoT devices. The approach for *discovery* makes use of mobile AR to allow users to identify new devices and easily access regularly-used devices in the physical space of their environment. It could enable immediate interaction with quickly-obtainable user interfaces (UIs) from the web, and provide developers with a convenient platform to display, within

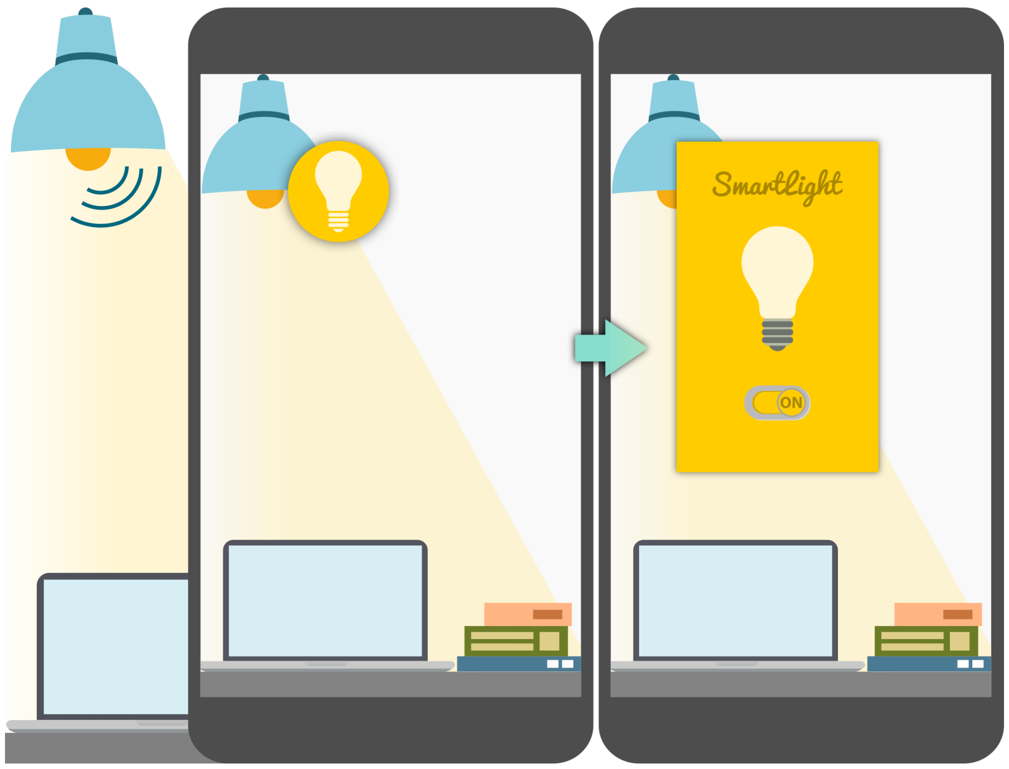


Figure 6.1: AR Browsing *discovery* model. In this model, users can open a "browser" on their smartphone or tablet, which uses the camera to identify devices and discover their associated web interfaces in physical space. When an interface is opened, it can use a JavaScript Bluetooth API or network protocol to interact with the device.

perceived physical space, custom interfaces for their devices that can be created using standard web tools. This mobile augmented reality browsing concept is depicted in Figure 6.1. In our study, we consider a few of the driving applications that could be supported, demonstrate with a set of early proof-of-concept implementations, and we explore some of the opportunities and challenges that arise in the development of such a system.

For more dynamic control in *interaction*, we also design a point-and-shoot model, in which users, guided by mobile AR, can find and select a single device or group of devices using their smartphone camera, and issue immediate action with custom interfaces. In this model, which is depicted in Figure 6.2, the phone's view acts as a framed physical search filter that directs the smartphone to the devices with which it should communicate. Using the entire screen as a selection mechanism relaxes the level of precision often required to interact with AR interfaces, while still providing a level of accuracy that is intuitive for use on mobile platforms. The targeting mechanism is partly inspired by frustum culling — the efficient practice in 3D graphics of only rendering objects that lie in front of the virtual camera rather than all objects in the environment. The components of the model are depicted in Figure 6.3.

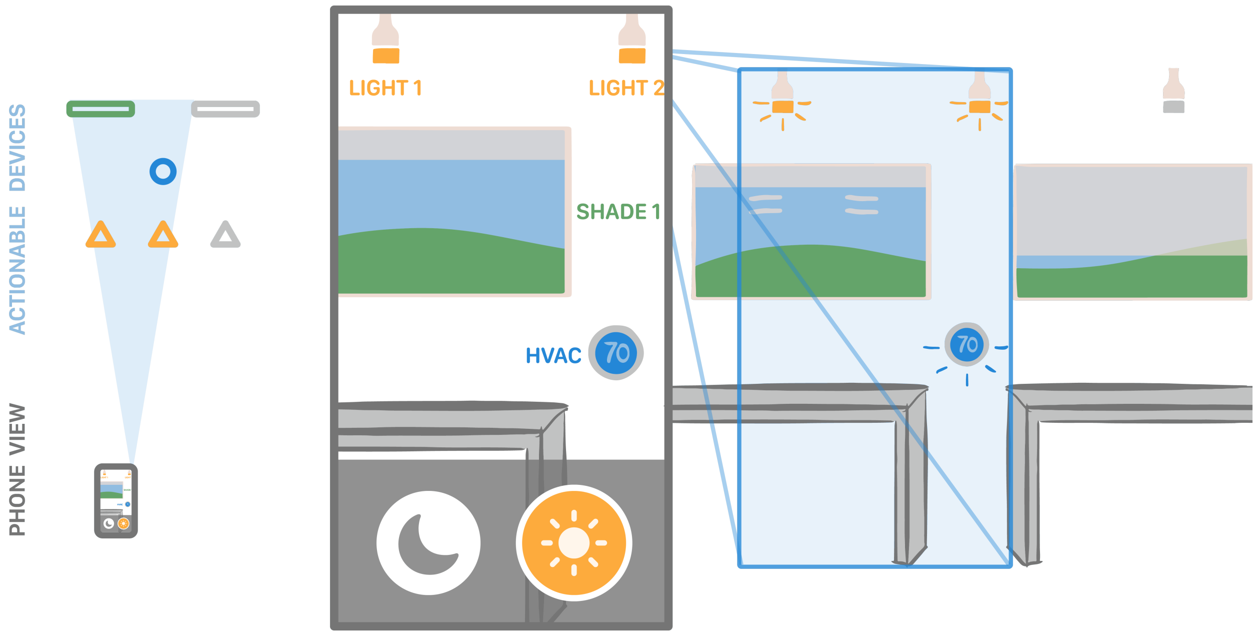


Figure 6.2: AR Point-and-Shoot *interaction* model. With point-and-shoot, an application can present a user interface on top of an augmented camera feed that maps virtual device objects to real smart devices in the space. The phone treats objects that intersect the frustum of the camera's view as actionable targets. Effectively, the phone‘s viewport acts as a filter to select one or more devices on which a desired action can be taken. In this example, the user sets devices in part of an office to day settings (lights on, shade up, A/C to 70° F) by selecting a button and facing the camera towards the items.

For the user, this removes the cognitive struggle of needing to know or recall the structural and semantic abstractions associated with a device in order to interact with it. The user should effectively be able walk into a room with their smartphone, see the "smart things" in the augmented space, use the camera to filter the scope of targeted devices, and interact using an appropriate interface. With this model, many actions can be performed on one or many smart devices with as little as a single on-sceen gesture and a motion to direct the camera. Additionally, use of the model inherently provides the user with a more useful and intuitive understanding of the devices in a space and their capabilities.

We present experiences and initial findings from building a prototype of the system and web applications that run on it that leverage, among other standards and technologies, an early version of the draft W3C WebXR API. The point-and-shoot work consists of: (a) a model for viewport-based filtering of smart space devices through which users may select groups of items on which to perform actions, (b) a prototype of a mobile platform for running AR-enabled web applications that can interact with smart spaces, (c) tools for developers to create web interfaces that can utilize the AR selection/filtering mechanism and access local network protocols (Bluetooth, mDNS, UDP) to communicate with devices, and (d) example applications to demonstrate functionality with various infrastructure strategies.

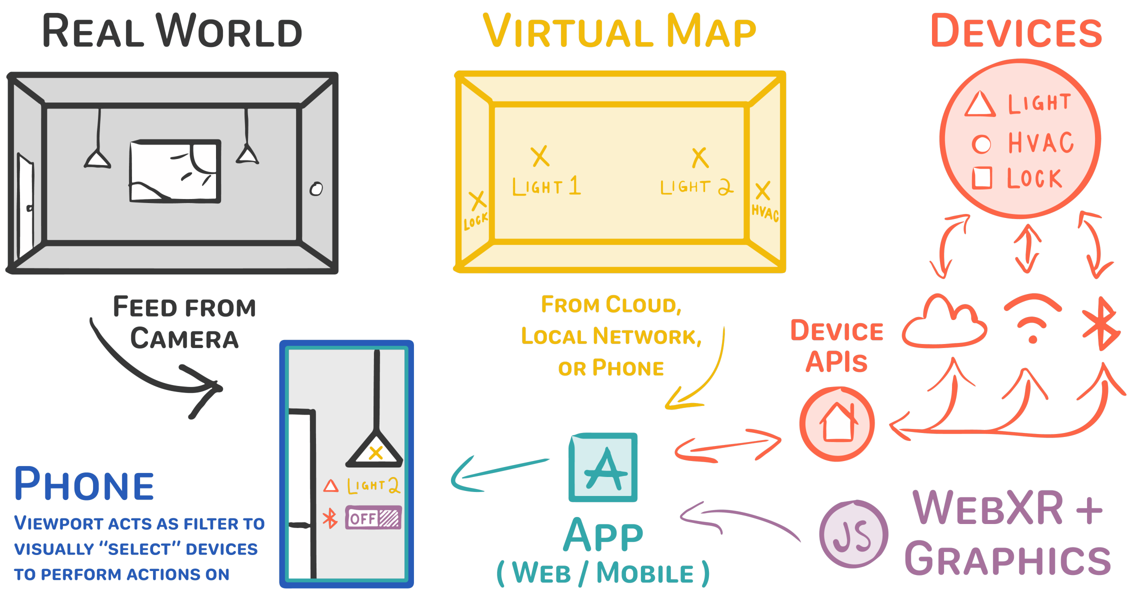


Figure 6.3: Components of point-and-shoot. The phone uses an overlaid virtual map on the realtime camera feed to both guide the user and keep track of virtual objects that map to real smart devices in the space. The phone‘s viewport acts as a filter to select groups of devices upon which users can perform context-specific actions in web-based interfaces.

6.1 Background & Related Work

We consider current standards for discovery and interaction in the Internet of Things, and technologies that enable the proposed mobile AR approaches.

6.1.1 Interaction Standards for IoT Devices

Device-specific apps are the current standard for interaction in the Internet of Things. They allow fine-grain setup and control of specific devices in an interface aesthetic of the developer's choosing. But this requires knowledge of the device and download of its app. It is also typically only accessible by the owners of the device, which discourages opportunities for ephemeral discovery and interaction.

6.1.2 Unified Control on Mobile Platforms

Unified control systems like Apple Home and Google Home begin to alleviate the problem by providing a single interface to control multiple devices in a house [13, 62]. As their names indicate, they are primarily intended for use in the home and limit scope to devices that implement specific proprietary protocols and are owned by the user. This still often requires download of the devices' apps for setup. It also imposes a standard aesthetic and provides a limited offering of control interfaces (e.g switches and sliders).

6.1.3 Device Discovery

Google's Physical Web allows discovery of nearby devices via Bluetooth Low Energy (BLE) [69]. The device broadcasts a URL which a phone can detect and open in a browser. The URL can point to a web page that acts as an interface for the device. If the device is connected to the Internet or the page uses a JavaScript Bluetooth API, users can interact with the device in real time. This enables immediate interaction in an interface of the developer's choosing. Unfortunately as the number of devices scales up, the listing model becomes overwhelming and it is often difficult for users to map the appropriate interface to device. This may be, in part, why Google has removed Physical Web support on Android and iOS. However, the concept of associating nearby physical devices with web content via a broadcast service may be useful in the context of a mobile AR browser that more tangibly connects interfaces to devices.

6.1.4 Early Inspirations for AR-Like Modalities

In 1987, a Bell Labs study on human-system communication discovered that users struggle to identify canonical names and keyword commands for systems using voice or type interfaces [52], indicating that visual articulation might be more effective. This problem continues today in IoT, where users must identify available devices, brands, and associated apps prior to any form of interaction, even with voice-control agents. A 2000 Microsoft study on emerging interfaces for smart devices examined user experiences of touch and speech interfaces [33]. While the study did not discuss AR as a modality, it suggested that the functionality of systems could improve when interfaces are reinforced with an input that provides location-awareness (e.g. gaze or gesture) to help disambiguate a target device.

6.1.5 Point-and-Shoot Photography

The interaction model discussed in this chapter draws inspiration from the spirit, principles, and impact of point-and-shoot cameras. They played a significant role in the rise of vernacular photography, democratizing capture of people, moments, and scenes to a greater non-professional population [38, 134]. In similar fashion, the point-and-shoot model could put greater accessibility and control of sophisticated devices and systems safely and meaningfully in the hands of average individuals who may not be as technically inclined, using a platform that has become the modern-day point-and-shoot camera [158].

6.1.6 AR on Mobile Platforms

In recent years, mobile platforms have featured increasingly advanced support for augmented reality (AR) with dedicated software frameworks and more capable hardware. Apple and Google have introduced ARKit and ARCore, software development kits that allow quicker and easier creation of AR-based applications [15, 61]. Some of the latest smartphone models improve AR functionality with infrared and LIDAR-based depth-sensing [15]. With native support for AR and more capable hardware on smartphones and tablets, mobile augmented reality has become a practical reality and exposes the technology to a large user-base. Prior to this, third-party mobile AR frameworks existed, but much work in AR tended to focus on head-worn display devices rather than mobile. Still, surveys of existing consumer AR technologies from the last decade indicate that while industry interest in AR is high, it lacks compelling practical applications [108, 156].

6.1.7 Mobile AR Systems in Practice

Mobile AR systems designed to enable interaction with objects in an environment usually only work with a specific set of devices, and limit the type of controls devices can have and how they are presented to the user. Some of the most popular mobile AR applications work only in highly specific environments like an outfitted office, classroom, or exhibit [25, 94, 114]. Many of these employ computer vision with limited training sets, which is made easier with built-in machine learning frameworks on mobile platforms, like Google's ML Kit and Apple's Core ML [13, 62]. HP Reveal is an example that uses image recognition to identify objects "anywhere", but requires the user to manually narrow the search-space by subscribing to a "channel" of objects [127]. It is primarily used to demonstrate object identification, rather than interaction.

6.1.8 Target Identifiers

Standard visual identifiers like QR codes or AprilTags are often used for general use cases. The mobile AR browser could potentially make use of such identifiers, but they may be obtrusive or distracting in the long run. They still generally require users to be somewhat informed about the environment around them and limit discovery when markers are not visible on screen. Early tablet-based AR control systems for appliances in homes with preset interfaces and tags have been studied, but not widely adopted [106]. Some work has been done to create AR systems with more subtle or visually appealing markers [79, 133]. Recent work in visible light communication (VLC) enable data dissemination through modulation of LEDs, which can be captured on a smartphone camera, while remaining mostly imperceptible to the human eye [96, 137]. This can potentially act as a visual "landmark" for devices in mobile AR.

6.1.9 Localization

Recent work has demonstrated integration of novel indoor localization techniques in AR. ALPS makes use of ultrasound beacons that broadcast audio that can be picked up on smartphone microphones and used to accurately determine location and orientation of the user [105]. Similarly, use of ultra-wideband (UWB) can accurately locate tags that can be attached to phones or other objects [138]. The location information obtained from these techniques can be used to help render an accurate and persistent AR environment for all users and between multiple uses [104, 117].

6.1.10 Smart Space Interaction

Previous work has considered various novel methods of device and smart space interaction. NaviCam and HomeWindow are early approaches for augmented interaction with real-world environments using portable devices [103, 139]. HomeBLOX is a graph-based user interface for IoT authoring [140]. Reality Editor and CAPturAR extend the approach to mobile and head-worn AR [78, 159]. Snap-To-It allows users to open a device interface by taking a picture of the object [51]. Summon allows discovery of web-based interfaces for multiple devices in a space in list and in-situ AR views [168]. Point-and-shoot builds on these works, providing selective device interaction in AR, through web interfaces.

6.2 AR Browsing & Discovery

To begin developing an AR-based browsing architecture, we consider design decisions about platform, target identification, user interfaces, and scope. To help aid the understanding of requirements and limitations, a demo prototype, shown in Figure 6.4, and a general-purpose implementation, shown in Figure 6.5, have been created. This demonstrates one mechanism for AR-based discovery and interaction in a known environment, and helps inform some of the design decisions that would need to be considered.

6.2.1 Platform

The most straightforward implementation of the mobile AR browser requires an Internet-connected mobile phone or tablet that has sufficient hardware and processing support for augmented reality. Most modern flagship smartphones are capable of this, at least with third-party software. Since 2018, iOS and Android platforms have had official support for mobile AR with dedicated software development kits (SDKs) from Apple and Google [13, 62]. Making this a mobile-based system, rather than one based on a head-worn device or a webcam, makes it more accessible to a larger population of current users and the touch interface can make it intuitive to use. Our prototype and general-purpose browser implementations are built on iOS using Apple's AR SDK, ARKit.

6.2.2 Targets

One of the more challenging parts of this system is defining how smart devices are identified as "targets"—locations which, when viewed on the phone's camera, prompt the phone to overlay associated content (e.g. images, videos, or, in our case, a linked fully-functioning user interface for the device).

One of the common ways to do this is with image recognition. If the phone's AR browser is trained to recognize the image of the device, it can overlay the interface relatively easily. The prototype is an implementation of such a system, as depicted in Figure 6.4. In this example, the target is identified visually using a trained image of the lamp. The challenge with this is one of scalability. It is unlikely that the browser can be trained to recognize every device in every angle and lighting scenario.

Alternatively, the geolocation (GPS coordinates) of devices can be obtained during the setup of the device. This, however, would likely only apply to stationary devices and would not account for mobile or wearable devices. Additionally, geolocation on phones is still quite coarse-grained (m-level accuracy) and development of a global map of devices may prove to be an intractable problem.

More recent phones are capable of creating their own local mapping of targets in their environment. It is possible for local mappings to be shared from phone to phone, but the variance in data and lack of a common origin point of reference may render the data useless when trying to load it on a different phone or even reload it on the same phone at a different time. A possible method to improve accuracy with local mappings might be to have users initially place their devices on a designated origin when they enter an IoT-outfitted space. This could re-calibrate the device's orientation to allow the phone to more accurately place known targets in a space, and to enable persistence of location between uses. One could also envision using a mount outfitted with NFC (near-field communication) or Bluetooth as the designated origin, which could provide a URL to load a space's map on to the phone. This way, any smartphone or tablet entering the space can retrieve the appropriate local mapping instead of needing to rely on global geolocation.

The devices themselves can broadcast their presence to the phone over the local Wi-Fi network or Bluetooth Low Energy. Like in the browsing architecture discussed in Chapter 5, the devices can provide a URL in their broadcast advertisements that would both help to identify the device and point to an address where the device's user interface is served. This can also help to simplify how devices are found. While fine-grained location (cm-level accuracy) cannot be obtained from the detection of Wi-Fi or Bluetooth, it indicates the relative proximity of the device, which can 1) inform the user of its presence and 2) provide a more simplified, scoped search-space for the phone to identify targets.

For example, a light bulb can advertise `https://lightswit.ch/control`, the address of its user interface, over Bluetooth Low Energy. Once detected on a user's phone, the device can be listed on-screen as a nearby "thing". At the same time, the phone can also refine its search-space of targets to objects that look like light bulbs associated with the `lightswit.ch` URL or, perhaps, known geolocations of such devices. If the phone is unable to identify the device, but the user has knowledge of its location, he or she can potentially contribute a picture or an updated coordinate to help identify the device in the future.

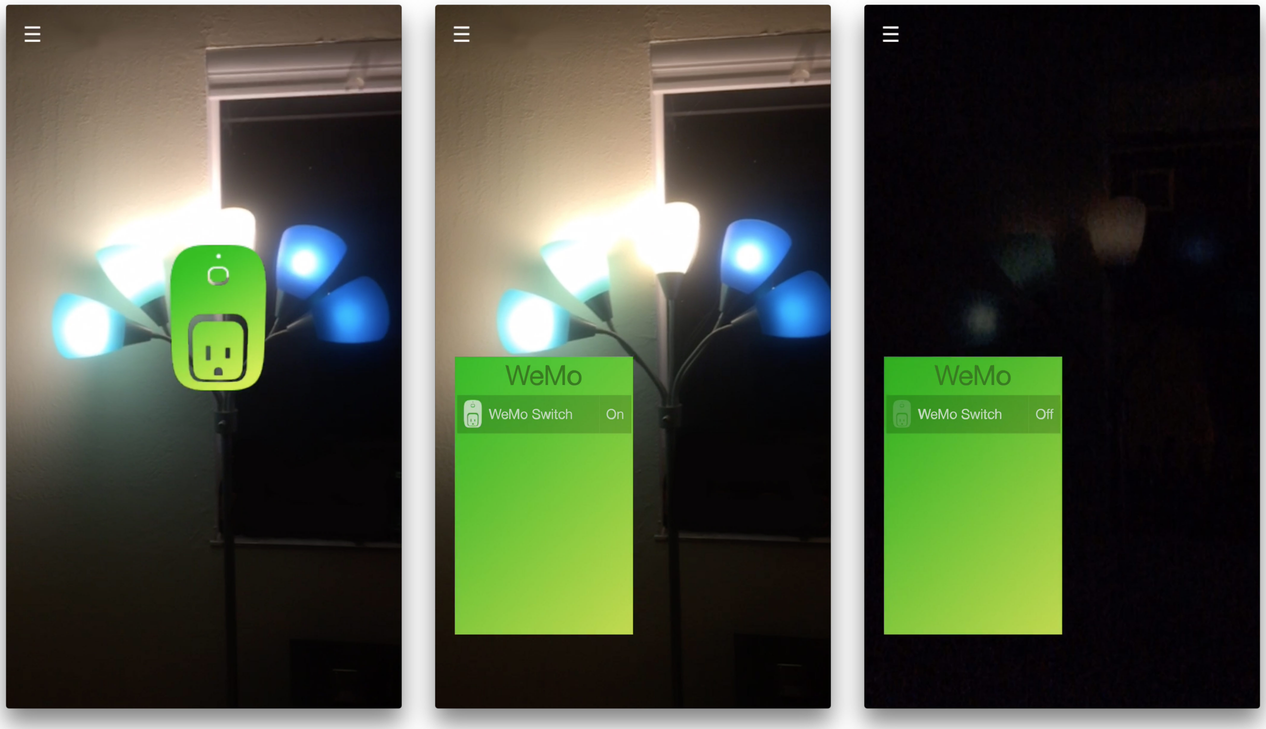
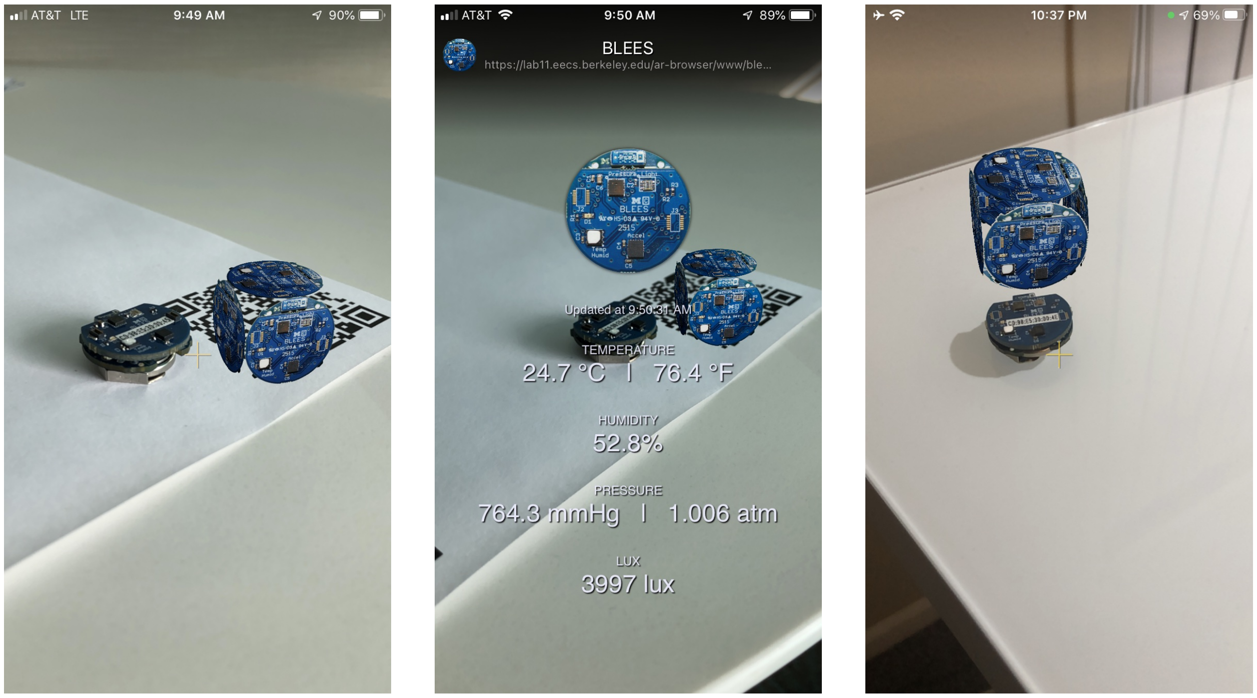


Figure 6.4: Screenshots of an early implementation of the mobile augmented reality browser. In this proof-of-concept implementation, the smartphone camera identifies the target and displays an indicator (the favicon for the target's URL) which the user may touch to open the device's linked web interface. Once opened, the interface, an HTML web page, uses JavaScript and a local network protocol to allow the user to interact with the Wi-Fi-connected device, and toggle the light on or off.

6.2.3 Target Proxies

The reality in many environments is that the device itself may not be a desirable target. For instance, a smart power meter outlet may not be visible when in use since the plug for the appliance it is metering may obscure it. Additionally, the data is directly related to the metered appliance, and the user may more naturally associate the interface for the meter with the appliance rather than the meter itself. The user may prefer that the interface for the meter be placed closer to the actual appliance. Figure 6.4 is also an example of this. The lamp was a chosen as the target location for the interface rather than the smart switch device at the outlet, as it more closely relates to the object of the intended activity of interaction. Consideration for these situations provide greater reasoning for allowing users to define/modify the placement of interfaces for the devices they own, or utilizing standard visual identifiers like QR codes that can be placed at the desired location, as is done general-purpose browser implementation shown in Figure 6.5.



BLEES
https://lab11.eecs.berkeley.edu/ar-browser/www/ble...

Updated at 9:50:31 AM

TEMPERATURE
24.7 °C | 76.4 °F

HUMIDITY
52.8%

PRESSURE
764.3 mmHg | 1.006 atm

LUX
3997 lux

(a) **(b)** **(c)**

Figure 6.5: Screenshots of AR Browser implementation. For the general-purpose AR Browser implementation, targets/interfaces are typically identified via QR codes (a). The app is also trained to detect visual cues from an image set of known device types (c), such as the BLEES environmental sensor [97]. It can visually identify the target and place the marker for the associated web app URL. When selected, the web app can be opened and display data from the device (b), received via BLE or other local network protocols.

6.2.4 Interfaces

Once a target has been identified, the browser should be able navigate to and display a linked user interface for the device that can display data from and interact with the device. To facilitate this in a general and scalable manner, standard web tools can be used to create the interface content. This means that the UIs are, in effect, web pages created with HTML, JavaScript, and CSS. This would allow developers to maintain their own creative liberties with the interaction model and interface aesthetic. The interface can make use of a browser-provided JavaScript Bluetooth API, a local networking protocol, or commands issued via cloud to enable communication between the browser and the device. More dynamic AR-based web interfaces are explored with the point-and-shoot approach described in Section 6.4.

In Figure 6.4, the opened interface is a web page served on the local Wi-Fi network. When the user toggles the switch , it issues a server-side command to the network-connected switch device through a UPnP protocol. In Figure 6.5, the opened interface is a remote web page which uses the JS BLE API to access and display data broadcast from the sensor.

6.2.5 Scope

Though it potentially limits a range of possibilities with public and ephemeral discovery, the browsing model can work as a local system that is focused on operating with devices in a user's home or work environment. Users can easily feed the system images of the devices in the actual environment they reside, providing targets that can be recognized more reliably. Alternatively, the user can define or correct an object's location using a drag and drop interface which will then be stored as coordinates in the phone's local mapping of the environment.

However, a system that is capable of working at a larger scale is more desirable. Identifying things as targets by image recognition is made more feasible when using the devices' Bluetooth or Wi-Fi broadcasts to refine the search-space. Use of standardized visual identifiers such as QR codes or AprilTags may improve performance, especially when devices are not stationary. Locations can be defined/corrected by the crowd using a drag-and-drop style interface to obtain new images of the target or modified geolocation.

Overall, defining and determining how the physical scope for interaction is interpreted will be important. Depending on the technologies used, a "nearby" device could be considered one within the device's Bluetooth range, on the device's Wi-Fi network, in identifiable visual range of the phone's camera, or within an arbitrary distance of the phone's geolocation.

6.3 AR Browsing Scenarios

In the mobile AR browsing model, a user would be able to point their phone at the device of interest, and immediately discover the interface for it. The interface would link to a web page which can make use of a cloud service, a local network protocol, or a browser-provided JavaScript Bluetooth API to interact with the device. We explore a few applications for which an MAR-based browsing model might be useful.

6.3.1 Smart Home Devices

The smart home has been the primary focus for the consumer IoT market. Home apps, like Apple Home and Google Home, provide a single console interface for all the smart devices in a house. When a moderate number of compatible devices are present, it is often difficult to remember which interface corresponds with which device, even when each one is appropriately labeled on the console, and used regularly. Additionally, to be compatible with such consoles, devices must utilize specific proprietary protocols, and are limited in the type of interface they can display on the console. For this reason, many IoT manufacturers struggle to or opt not to integrate with these consoles. Regardless of compatibility, each device still typically requires installation of its own mobile app to function.

With the mobile AR browser, finding the interface for a device would be similar to finding one in the physical world. Instead of the Home apps' strategy of relying on the user's recall of a particular device, the browser would utilize the user's recognition by allowing the user to find the appropriate interface by pointing to the device of interest, much like inspecting a lamp to find a switch.

6.3.2 Device Setup

To ease the process of connecting devices to the Internet in a home, the mobile AR browser could potentially enable a quicker and more convenient method for device setup and configuration. Instead of requiring the user to download an app and to go through a setup process of entering ID numbers or accessing settings to initiate a connection, the device's interface can be opened in the browser and it can use visual verification methods and a direct connection via Bluetooth to sync with a particular device. Additionally, users can place target locations for device interfaces in the augmented reality space during this process.

6.3.3 Ephemeral Devices

Breaking out of the confined scope of the smart home, an mobile AR browser could potentially be utilized to enable new opportunities for quick, ephemeral interaction with new and easily-discoverable interfaces in the global Internet of Things. Interactive devices and interfaces could be deployed in public (e.g. stores, sidewalks, conference rooms, exhibits). Users would explore these spaces by effectively using the browser as an AR "window" into the public web. Support for device discovery and ephemeral interaction of this kind would be a major departure from the app-per-device and single-scoping models that are prevalent in IoT. Enabling this, however, will require important consideration of the methods for retrieval of data about devices in public and both the virtual and physical association of the appropriate interfaces to those systems.

6.4 AR Point-and-Shoot Interaction

To flexibly facilitate implementation of a range of point-and-shoot applications, we create a platform that incorporates support for emerging web technologies — particularly, WebXR, a W3C draft API for developing AR and VR experiences on the web, while leveraging native platform AR SDKs and hardware [157]. This allows developers to easily and quickly create AR point-and-shoot applications in HTML and JavaScript, and provides opportunities for potential cross-platform support. Additionally, we extend native networking capabilities and shims for device APIs as Javascript libraries.

6.4.1 Point-and-Shoot Platform

In order to run WebXR-based point-and-shoot applications, we implement a new version of the mobile AR browser as an iOS app. Central to the platform is a specialized webview that runs native elements derived from Mozilla's WebXR Viewer [118, 165] in the background to enable WebXR support for HTML webpages and interface with iOS's native augmented reality SDK, ARKit [15]. As in the previous browser implementations, the platform extends the webview with interfaces for local network protocols (mDNS, UDP/datagram sockets, Bluetooth) to allow point-and-shoot applications to communicate with local devices. While remote webpages can be loaded on the platform, we have baked all of our implemented applications into the iOS app for quick access. If desired, any web app can be packaged with the platform's libraries to operate as a standalone native iOS app.

6.4.2 Applications

With the platform support in place, point-and-shoot applications can make use of Mozilla's WebXR JavaScript API [72, 119]. This, in a few lines of code, can transform a blank HTML webpage into a camera-fed canvas for augmented reality with hooks to key features of the native AR SDK.

Our implemented applications also make use of ThreeJS, a web-based 3D graphics library [34]. Using it, we create a core set of tools that generates the virtual object representation of devices and facilitates the tracking functionality that understands when devices are within the viewing frustum of the smartphone camera.

To communicate with local devices, the applications can use special JavaScript APIs provided by the point-and-shoot platform. These APIs provide access to the platform's extended local network protocols. We have further implemented a set of shims to interface with the existing network communication protocols for a set of popular brands of devices.

6.4.3 Mapping

To simplify and aid in configuration of the virtual maps of outfitted spaces, we provide tools for applications to incorporate a quick-setup procedure that dynamically populates a selection list of discovered relevant devices. This allows users to manually tap to place a corresponding virtual object near the physical location of the target device and interact with it using the application's interface in the matter of seconds. Depending on the application and target devices, the phone can use multiple discovery methods using the JavaScript network APIs extended through the platform. Most of the implemented applications discover devices by listening for zero-configuration services like Bonjour/mDNS, querying with device-specific local network protocols, or scanning for Bluetooth Low Energy broadcasts. The application can save the map and reload it as necessary, or even share to other devices. Some of the implemented applications discussed in Section 6.5 use this method for object placement. Additionally, for real-time automatic localization of devices in a smart space, we can consider techniques and technologies employed by various systems, such as visual markers like QR codes [78, 79, 168], light-based anchors [6, 96, 112, 146], ultrasound beacons [105], vibratory sensing [174], ultra-wideband tags [14, 22, 83], trained images [51, 127], and other computer-vision based mechanisms.

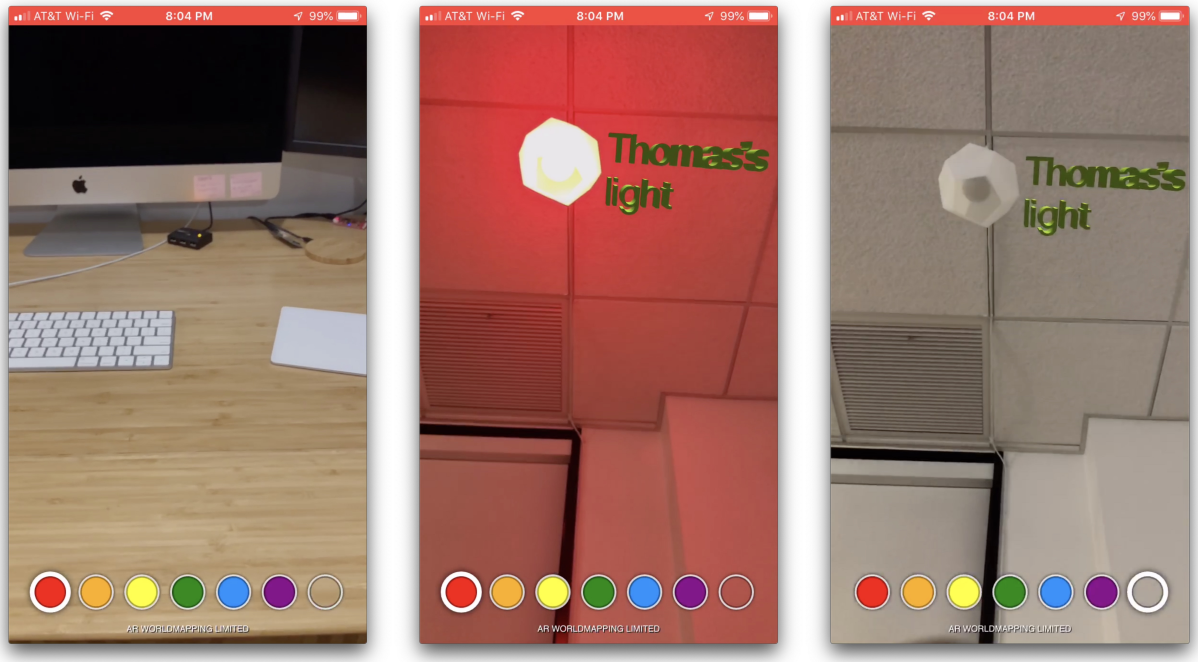


Figure 6.6: Application: Paint the Lights. An implemented web application using WebXR that allows users to control smart lights in a space by simply selecting a color button or the off button from on-screen controls, and turning the phone's camera towards the lights they would like to change.

6.5 AR Point-and-Shoot Applications

We explore and validate the capabilities of the point-and-shoot model with a series of web-based applications implemented using WebXR, ThreeJS, and extended network APIs.

6.5.1 Paint the Lights

"Paint the Lights" is a simple application that makes use of this model and demonstrates the ability to perform instantaneous action on devices in view. Augmented reality is used to label and mark the locations of smart RGB LED lights in the space. The screen presents options for colors, as well as an option for "off". If the user selects one of the options, she can point the phone's camera towards any of the lights, which are then changed instantly to the state indicated by the selection. The screenshots shown in Figure 6.6 depict the user selecting the color red and pointing the phone at the light above to change its color instantly.

This is achieved on the phone by keeping track of the virtual object representing the light. Any time the object enters the phone's viewport, the phone issues a command via REST API or through the local network to take the selected action on the light corresponding to the virtual object.

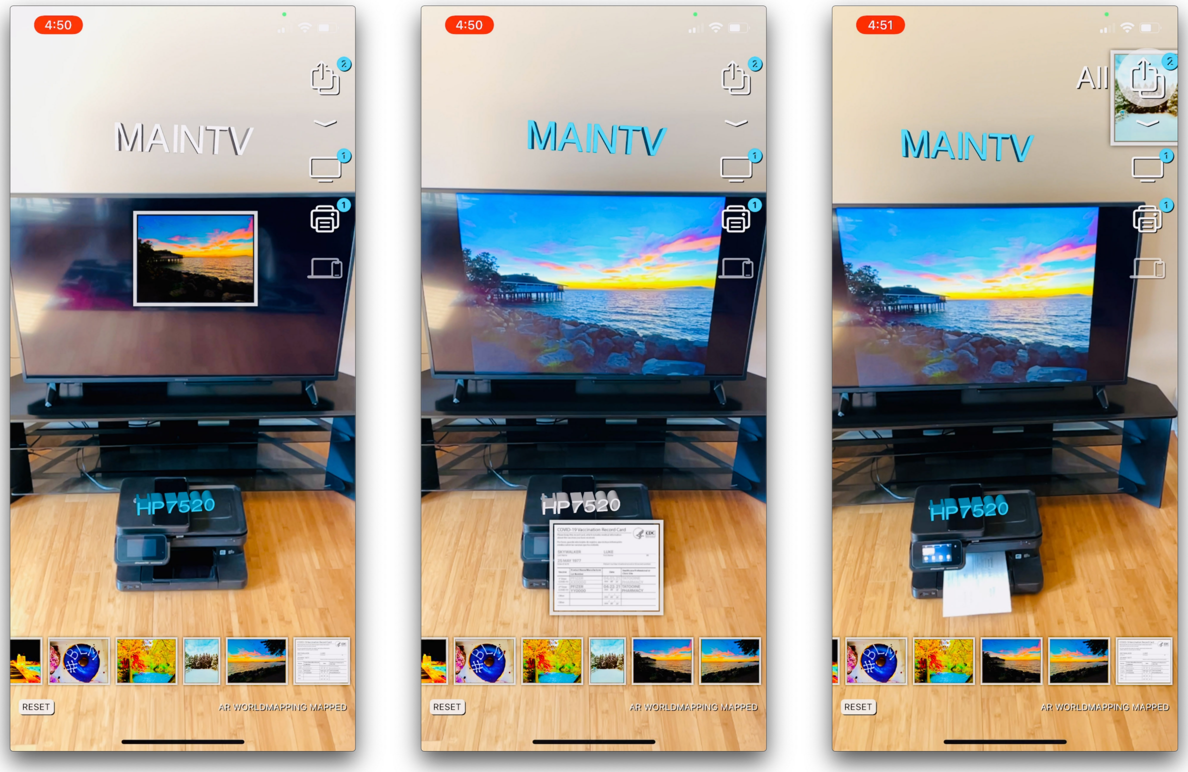


Figure 6.7: Application: Drag-and-Drop Share. A WebXR application that allows users to share images to network displays, printers, and computers, by dragging them to the device on the augmented camera feed. Users may also send media to all devices or all of a device type by dragging to corresponding on-screen action icons representing aggregates of all items in view.

6.5.2 Drag-and-Drop Share

A second application, "Drop Share", allows users to share media to identified devices like displays and printers with a drag-and-drop action. Like before, locations of compatible devices are marked on an AR overlay. The interface contains a photo picker bar at the bottom of the screen, as well as a set of action/summary icons for all devices and classes of device, each of which contain counts of the corresponding devices in view. The image can be dragged from the picker to a device's location on camera to share with it. For example, dragging a photo to a printer on the camera feed will initiate a job to print the photo on that device. An example of this is depicted in Figure 6.7. Alternatively, users can drag media to one of the action icons to share with all visible devices or class of devices in a single drop gesture. In the final panel of the figure, an image is dropped on the "All" icon, which will both display and print the image.

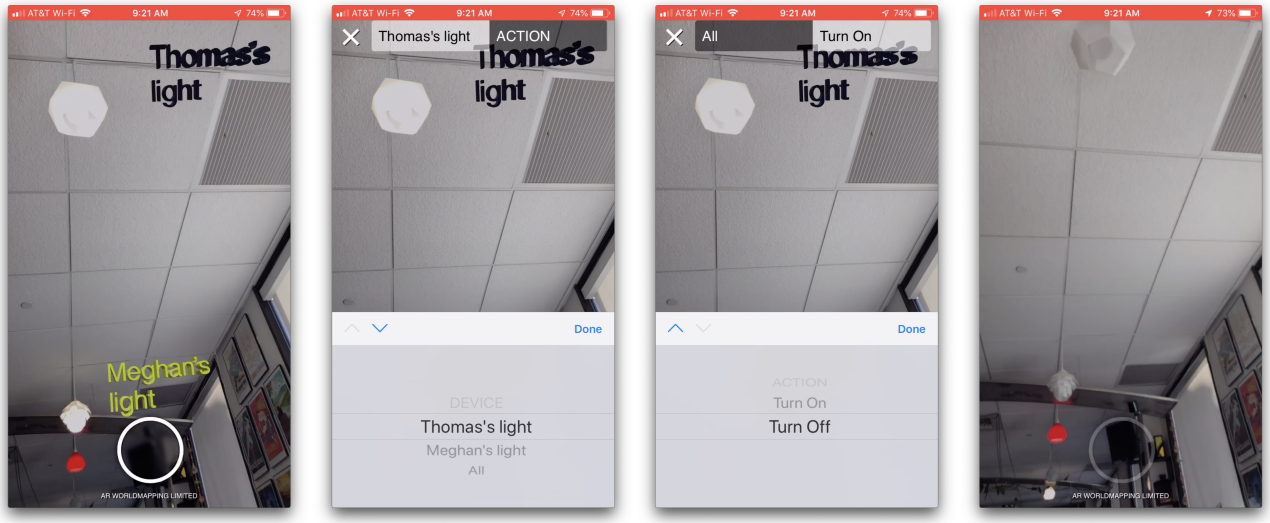


Figure 6.8: Application: Smart Space Snap. An implemented web application using WebXR that allows users to control devices in a space by taking a picture of a targeted selection of devices and choosing actions to perform on them from a set of options catered specifically to the selection.

When the user initiates a drop action, the app obtains the local address and callback function associated with the target device's virtual object — metadata that is assigned automatically in the mapping phase. Calling the function with the known address sends the dropped file to the device using the appropriate network protocol (e.g. Internet Printing Protocol [IPP] for printing, UDP/datagram APIs for casting). By keeping track of objects that enter or exit the viewport, the app allows the user to initiate aggregate action to a desired subset of devices at any moment.

6.5.3 Smart Space Snap

In an application we call "Smart Space Snap", augmented reality is again used to mark and label smart devices. A shutter button is present on screen to allow users to take a picture of the device or a set of devices with which they desire to interact. The button is only available to the user when a virtual object is present on screen. When the shutter button is pressed, the camera feed pauses on the last frame and the virtual mapping is frozen in place. In the example shown in Figure 6.8, the user sees multiple devices in the room, settles on two, takes a snap, chooses the devices to interact with (all two on screen), selects an action (turn off), and submits the command.

With this application, actions could be applied to devices of different classes. For example, a snap of a smart light and smart window shades might provide a "darken" action that would dim the light and close the blinds.

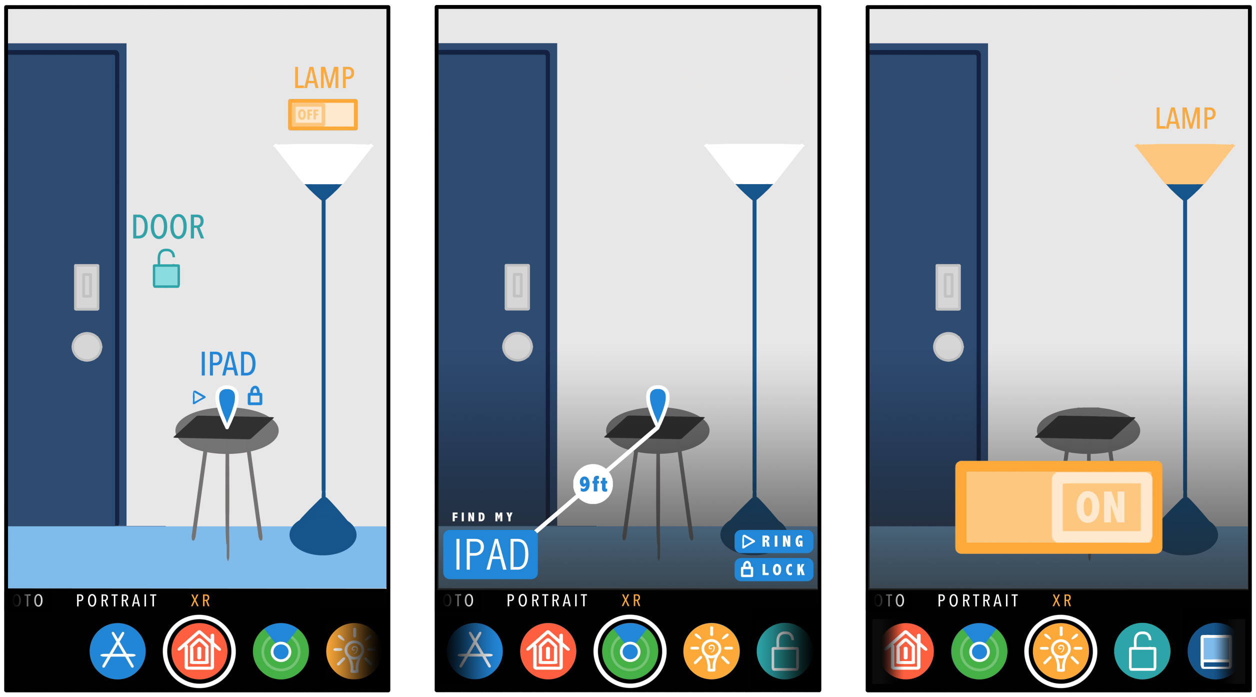


Figure 6.9: Concept: Integration with native camera app. In a hypothetical OS-level integration, point-and-shoot could be employed in a quickly-accessible mode in the smartphone camera app. The mode would present a set of applets that enable AR/XR overlays, as well as an app store to find relevant third-party applets. The user could, for instance, swipe to an applet that displays a virtual switch for all smart lights in view.

6.5.4 Quick-View Sensor Readings

In a "Quick-View" application, the phone displays real-time readings from plug-load power meters and environmental sensors (temperature, humidity, pressure, light) in a space. Readings are obtained directly from the devices via Bluetooth Low Energy broadcasts sent once every second. The metrics are displayed on the augmented feed at the corresponding sensor locations. On the bottom of the screen, the interface also displays average environmental sensor readings and aggregate power usage for all the sensors in view.

A version of this application uses QR codes for automatic localization, rather than requiring a mapping phase. The QR image contains encoded identification information to automatically link the appropriate Bluetooth data to the corresponding device. It also serves as a visually unique feature on which to place the virtual object. However, for the initial placement, the phone camera must be within roughly 1 meter of a 7.5 sq cm QR code for proper detection.

6.6 Alternative Platform Paradigms

As an alternative to our web-based implementation, the point-and-shoot model could potentially be integrated as a quickly-accessible feature on smartphone OSes. In Figure 6.9, we envision an XR mode in the phone's camera app (like Video, Photo, Pano, Portrait, etc). A selection of XR applets is presented at the bottom of the screen including app store and third-party options (similar to the app selection above the keyboard in iOS Messages app). Like the implementations described in Section 5.4.1, these applets could enable specialized XR "skins", "filters", or views of the environment that discover, identify, and interact with associated devices. This, for instance, may be the perfect place for iOS platforms to feature an AR-based "Find My" applet, where other devices with ultra-wideband technology can be located. The figure depicts a "Home" applet which overlays simplified controls for multiple device types, like those one might find in smartphone "Home" apps. The phone can detect nearby devices in order to sort applets by relevance to the space or suggest applets that users may not have. This mode could also feature applets for games and immersive experiences.

One such experience could provide further insight and encourage experimentation with the various components and spaces in a user's smart home (e.g. suggesting how one might be able to save energy by trying out different settings, providing personal recommendations of different "mood lighting" one may have not tried before, etc.) by displaying a simulated view of the changes in their home. This could make alternative scenarios visible and accessible in situ, and provide an option to make such changes immediately.

While this chapter focuses on mobile mixed reality, the browsing architecture and point-and-shoot model could be applied to AR-enabled headsets or other integrated vision-based platforms in future work. As such head- or eye-worn devices become efficient, compact, and portable enough for regular wear, we can envision enabling quick and intuitive control of devices in a variety of surrounding environments using eye tracking and head movement as potential targeting mechanisms.

6.7 Discussion

In this section, we discuss some research challenges and questions that should be considered when designing and further implementing the mobile AR browsing architecture.

6.7.1 Location Determination & Accuracy

A key consideration for the mobile AR browser is determining where interfaces are placed in augmented reality and ensuring that the placement corresponds with a user's intuitive understanding of how that interface relates to the physical environment. What are the methods to accomplish this? Can we do this without — or at least with minimal — user setup? Can we support both stationary and mobile objects of interest? How do we properly identify and distinguish interfaces for multiple targets in close proximity? How will different types and levels of lighting affect the accuracy of target identification and interface placement? We discussed some of these considerations in Sections 6.2.2 and 6.2.3. Some solutions for improvement might include updating location by crowd-sourcing user location (verification by proximity), storing device location meta-data in the cloud which can be downloaded based on user location, or "annotating" locations and devices of interest with QR codes that link to well-defined schema.

6.7.2 Communication Topology

Once a web-based user interface is opened, how do we enable direct interaction with the device? We suggest enabling support for local interaction using a JavaScript Bluetooth API for Bluetooth devices. Interfaces for Wi-Fi devices can implement local network protocols if the web content is served on the local network. Alternatively, as many mobile apps currently do, the interface can issue commands through a dedicated cloud service for devices with continuous Internet connectivity. This would effectively allow developers to create and deploy app-like UIs for their own devices.

6.7.3 Usability & User Experience

The intention of this browsing model is to begin taking steps to make software-based interaction with IoT devices feel as intuitive as physical interaction with our surrounding environment. Besides physical placement of interfaces, what are the design decisions that help interface feel more tangibly and physically tied to the devices they control? Can we make interaction via mobile as convenient as on-device interfaces? Can we ensure that this gives developers the power to create interfaces suitable for their product rather than constraining the product to a minimal set of services that are available? How do we design the browser to better promote ephemeral discovery and interaction in public?

Augmented reality, while intuitive in many ways, can still generally feel awkward to users. Will users adjust to the paradigm of placing a phone between them and the world around them? Interesting approaches have been taken in the past to help reduce awkwardness in MAR, like using camera attachments to facilitate a more comfortable screen angle [40] or incorporating novel methods of feedback [109].

While not entirely important at least with initial prototypes, consideration of occlusion and other visual features that help provide a sense of realism to the virtual objects in physical space will eventually be useful for improving the user experience.

6.7.4 Developer Experience

Ideally, the process of creating devices that are compatible with the browser, and developing interfaces that are discoverable and operable on the browser should be a relatively convenient process compared to the current siloed mobile app approach. What should the programming environment look like and what tools should be made available to developers?

Because the browser would essentially run on an extension of web standards, developers could still make use of a typical web programming environment to create the interfaces. As shown in implementation, browser-defined APIs are useful to facilitate systems like BLE and WebXR. They could also extend camera and visual verification protocols. Determining the exact set of functions required will be vital. For device development, standard advertisement services over BLE or Wi-Fi (e.g. mDNS) can be used to broadcast URL and other metadata to the phone. The developer can also set up a server to host a UI on the device itself.

6.7.5 Browsing Model

The UIs displayed on a user's smartphone can be physically tied to the devices they control or to a relevant point in physical space. But does this present a larger benefit than previous paradigms, like simple linking through Physical Web or QR codes alone? While Physical Web provides a convenient means for content discovery, it can either overwhelm the user with notifications for a long list of web pages or prevent relevant results from being presented in an attempt to filter that list. Additionally, it could be difficult for the user to determine which interface is associated with which device.

QR-linking does enable more physically-tied associations of content to target objects. However, at present, the common uses for QR seem to be very deliberate, single-time actions like linking to a download for an app or to a configuration web page within a browser. It also requires switching back and forth between camera and browser to open multiple interfaces. The mobile AR browser would effectively be combining camera, browser, and local network protocols to allow for a more seamless experience when inspecting and interacting with a space.

Once a model for linking interactive web interfaces to target devices is established, are there more interesting and novel features that could be supported through the mobile AR browser? Perhaps, for instance, a method could be established to allow users to easily perform trigger-action programming connecting multiple devices in their environment.

6.7.6 Data & Privacy

Because the system uses camera, location, and data about devices in local and private environments, there are risks associated with the size and sensitivity of the data. It is important to determine exactly what data is necessary to save locally or send online, and develop measures to ensure that no more than the necessary amount is retained, transmitted, or shared. Additionally, the browser model will likely require some defined notions of ownership and permission, and rules to enforce them. Safeguards will also need to be put in place to detect and prevent links to malicious web interfaces. It is essential that, at least, the standard practices of web security are preserved and supported through the browser.

6.7.7 Power Usage

Currently, use of augmented reality is a notable drain on phone batteries, which may discourage people from using the browser, especially in public, where phone charging is often limited. Phone companies seem to be allocating a considerable amount of effort into making it more efficient. Considering what practices make more efficient use of energy will be useful when implementing the design. For example, using flat imagery like favicons and standard rectangular web page interfaces in the mobile AR browser will generally require less energy than rendering 3D objects and controls, like the interfaces using point-and-shoot.

6.8 Summary

In this chapter, we have presented mobile AR-facilitated browsing and interaction models, in which users can discover, select, and perform action on one or many smart devices by controlling the view of their smartphone's camera. We implement these models in a series of applications and a set of underlying system prototypes that leverage mobile, web, and network technologies. This represents a step towards the more natural usage patterns of direct manipulation, but enhances it with interaction akin to point-and-shoot photography. To operate at scale, key decisions will need to be made on standards, particularly for target recognition and device discovery. Further studies will be required to better assess and improve the usability of a such a system, but utilization of this architecture may enable us to begin taking steps to break free of the walled-garden infrastructure that is stifling the Internet of Things, while also providing compelling use cases for the augmented reality technology that is becoming more prevalent on our mobile devices. This has the potential to be a key enabling methodology in both mixed reality and IoT applications, but poor accessibility, implementation, and user experience can render it a short-lived paradigm if it is overtaken by misguided practices.

Part IV

Reflections & References

Chapter 7

Conclusion

The Internet of Things (IoT) is a growing industry, with an ever-increasing number of smart devices being connected to the Internet everyday. However, the potential of that growth is hampered by the endemic utilization of siloed application-layer gateways as a stopgap for connectivity and interaction in IoT industry. The prevalence of this isolating practice is restricting scalability, affordability, and accessibility for a broader range of constrained IoT devices.

To address these limitations, this work has presented a set of application-agnostic approaches to connectivity and interaction for IoT devices. The proposed network architecture uses ubiquitous short-range gateways to openly facilitate data transport for constrained devices to the Internet, while the browsing architecture provides a scalable approach to discovering and interacting with nearby devices. These architectures offer a simple, yet dynamic framework for enabling Internet connectivity and user interaction for IoT devices, breaking down the barriers imposed by siloed gateway standards.

Furthermore, the proposed methodology for IoT connectivity and interaction opens up exciting possibilities for the future of the IoT industry. By providing a more versatile and accessible approach to IoT, this work paves the way for the creation of more innovative and intuitive IoT devices and interfaces. The design principles presented in this work offer a foundation for researchers and device developers to build upon, further advancing the development of the smart and connected world of the future.

The scope of this work can be broadened by applying the proposed approaches to alternative network systems and protocols. As different low-power wireless technologies find their way into smartphones, computers, microcontrollers, and other emerging platforms over time, these architectures could continue to provide a means to facilitate discovery, open connectivity, and interaction for devices. Implementing support for end-to-end IPv6 could allow devices to participate and communicate as full-fledged citizens of the Internet. Additionally, conducting comprehensive studies on user and developer experiences would offer valuable insights to further validate and improve the browsing mechanism for device discovery and interaction.

If major smartphone manufacturers were to embrace the mobile gateway and device browsing techniques as operating system-level services, it could unlock enormous potential in the coverage and availability of IoT connectivity and interaction capabilities. The proliferation of inexpensive general-purpose static gateway hardware can further empower unprecedented accessibility for IoT devices. Furthermore, the mixed reality browsing approach could provide a significantly enhanced user experience when integrated with new technologies that enable accurate fine-grained localization while maintaining low power operation, as well as upcoming AR headsets and platforms.

Overall, the proposed application-agnostic approaches to connectivity and interaction represent a significant step forward in realizing the full potential of the IoT industry. By providing a general-purpose alternative to the current siloed gateway standard, this work opens up new possibilities for the development of more advanced, dynamic, and affordable IoT devices — making the Internet of Things more accessible to everyone.

Bibliography

[1] ABI Research. *Average Size of Mobile Games for iOS Increased by a Whopping 42% between March and September*. Oct. 2012. URL: https://www.abiresearch.com/press/average-size-of-mobile-games-for-ios-increased-by-/.

[2] Joshua Adkins and Prabal Dutta. "Monoxalyze: Verifying Smoking Cessation with a Keychain-sized Carbon Monoxide Breathalyzer". In: *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*. SenSys '16. Stanford, CA, USA: ACM, 2016, pp. 190–201. ISBN: 978-1-4503-4263-6. DOI: 10.1145/2994551.2994571. URL: http://doi.acm.org/10.1145/2994551.2994571.

[3] Joshua Adkins, Branden Ghena, and Prabal Dutta. "Freeloader's Guide Through the Google Galaxy". In: *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. HotMobile '19. Santa Cruz, CA, USA: ACM, 2019, pp. 111–116. ISBN: 978-1-4503-6273-3. DOI: 10.1145/3301293.3302376. URL: http://doi.acm.org/10.1145/3301293.3302376.

[4] Joshua Adkins et al. "The Signpost Network: Demo Abstract". In: *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*. SenSys '16. Stanford, CA, USA: ACM, 2016, pp. 320–321. ISBN: 978-1-4503-4263-6. DOI: 10.1145/2994551.2996542. URL: http://doi.acm.org.proxy.lib.umich.edu/10.1145/2994551.2996542.

[5] Adobe Systems. *Adobe PhoneGap*. Apr. 2018. URL: http://phonegap.com/.

[6] Karan Ahuja et al. "LightAnchors: Appropriating Point Lights for Spatially-Anchored Augmented Reality Interfaces". In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. UIST '19. New Orleans, LA, USA: Association for Computing Machinery, 2019, pp. 189–196. ISBN: 9781450368162.

[7] AllSeen Alliance. *Open Source IoT to Advance the Internet of Everything*. URL: https://allseenalliance.org.

[8] Amazon. *All-new Echo (4th Gen) — With premium sound, smart home hub, and Alexa*. Oct. 2021. URL: https://www.amazon.com/All-New-Echo-4th-Gen/dp/B07XKF5RM3.

[9] Amazon. *Amazon Sidewalk Privacy and Security Whitepaper*. Tech. rep. Amazon, Sept. 2020. URL: https://m.media-amazon.com/images/G/01/sidewalk/final_privacy_security_whitepaper.pdf.

[10] David P. Anderson et al. "SETI@Home: An Experiment in Public-resource Computing". In: *Commun. ACM* 45.11 (Nov. 2002), pp. 56–61. ISSN: 0001-0782. DOI: 10.1145/581571.581573. URL: http://doi.acm.org/10.1145/581571.581573.

[11] Apache Software Foundation. *Apache Cordova*. Mar. 2019. URL: https://cordova.apache.org/.

[12] Apple. *Airtag - Apple*. Apr. 2021. URL: https://apple.com/airtag/.

[13] Apple. *Apple Developer Documentation*. Oct. 2018. URL: https://developer.apple.com/documentation/.

[14] Apple. *AR Quick Look - Apple Developer*. Nov. 2020. URL: https://developer.apple.com/augmented-reality/quick-look/.

[15] Apple. *ARKit — Apple Developer Documentation*. Feb. 2021. URL: https://developer.apple.com/documentation/arkit.

[16] Apple. *Extending Your App's Background Execution Time*. Sept. 2019. URL: https://developer.apple.com/documentation/uikit/app_and_environment/scenes/preparing_your_ui_to_run_in_the_background/extending_your_app_s_background_execution_time.

[17] Apple. *HomeKit - Apple Developer*. Sept. 2019. URL: https://developer.apple.com/homekit/.

[18] Apple. *HomeKit — Developing Apps and Accessories for the Home*. June 2021. URL: https://developer.apple.com/homekit/.

[19] Apple. *HomePod*. Oct. 2021. URL: https://www.apple.com/homepod/.

[20] Apple. *iBeacon — Apple Developer*. Apr. 2017. URL: http://developer.apple.com/ibeacon.

[21] Apple. *iOS - Home - Apple*. Sept. 2019. URL: https://www.apple.com/ios/home/.

[22] Apple. *Nearby Interaction — Apple Developer Documentation*. Jan. 2021. URL: https://developer.apple.com/documentation/nearbyinteraction.

[23] Sungho Bae et al. "Browsing Architecture with Presentation Metadata for the Internet of Things". In: *2011 IEEE 17th International Conference on Parallel and Distributed Systems*. Dec. 2011, pp. 721–728. DOI: 10.1109/ICPADS.2011.36.

[24] David Barth. *The bright side of sitting in traffic: Crowdsourcing road congestion data*. 2009. URL: http://googleblog.blogspot.com/2009/08/bright-side-of-sitting-in-traffic.html.

[25] P. Belhumeur et al. "Searching the World's Herbaria: A System for Visual Identification of Plant Species". In: 2008, pp. 116–129. DOI: 10.1007/978-3-540-88693-8_9. URL: http://dx.doi.org/10.1007/978-3-540-88693-8_9.

[26] Alex Bellon, Alex Yen, and Pat Pannuto. "TagAlong: Free, Wide-Area Data-Muling and Services". In: *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*. HotMobile '23. Newport Beach, California: Association for Computing Machinery, 2023, pp. 103–109. DOI: 10.1145/3572864.3580342. URL: https://doi.org/10.1145/3572864.3580342.

[27] BLU. *BLU Dash JR*. URL: http://www.bluproducts.com/index.php/dash-jr-4-0-k.

[28] Bluetooth Special Interest Group. *Bluetooth Core Specifications*. July 2017. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification.

[29] Bluetooth Special Interest Group. *Bluetooth SIG*. Sept. 2019. URL: https://www.bluetooth.com/.

[30] Bluetooth Special Interest Group. *Core Specification 4.0 — Bluetooth Technology Website*. June 2010. URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[31] Bluetooth Special Interest Group. *Internet Protocol Support Profile — Bluetooth Technology Website*. Dec. 2014. URL: https://www.bluetooth.com/specifications/specs/internet-protocol-support-profile-1-0/.

[32] Julia Brich et al. "Exploring End User Programming Needs in Home Automation". In: *ACM Transactions on Computer-Human Interaction* 24.2 (Apr. 2017). ISSN: 1073-0516. DOI: 10.1145/3057858. URL: https://doi.org/10.1145/3057858.

[33] B. Brumitt and J. Cadiz. "" Let There Be Light": Examining Interfaces for Homes of the Future." In: *INTERACT*. Vol. 1. 2001, pp. 375–382.

[34] Ricardo Cabello. *three.js: Javascript 3D Library*. Mar. 2021. URL: https://github.com/mrdoob/three.js/.

[35] Arnab Chakrabarti, Ashutosh Sabharwal, and Behnamm Aazhang. "Using Predictable Observer Mobility for Power Efficient Design of Sensor Networks". In: ISPN, 2003, pp. 129–145. URL: http://www.isi.edu/~johnh/PAPERS/Park11a.pdf.

[36] Bharat S. Chaudhari, Marco Zennaro, and Suresh Borkar. "LPWAN Technologies: Emerging Application Characteristics, Requirements, and Design Considerations". In: *Future Internet* 12.3 (2020). ISSN: 1999-5903. DOI: 10.3390/fi12030046. URL: https://www.mdpi.com/1999-5903/12/3/46.

[37] Cisco. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. Tech. rep. Cisco, Apr. 2011. URL: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[38] Paul Cobley and Nick Haeffner. "Digital cameras and domestic photography: communication, agency and structure". In: *Visual Communication* 8.2 (2009), pp. 123–146.

[39] Don Coleman. *Bluetooth Low Energy (BLE) Central Plugin for Apache Cordova*. Jan. 2017. URL: https://github.com/don/cordova-plugin-ble-central.

[40] A. Colley et al. “Changing the Camera-to-screen Angle to Improve AR Browser Usage”. In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI ’16. Florence, Italy: ACM, 2016, pp. 442–452. ISBN: 978-1-4503-4408-1. DOI: 10.1145/2935334.2935384. URL: http://doi.acm.org/10.1145/2935334.2935384.

[41] Connectivity Standards Alliance. *Matter is the Foundation for Connected Things*. June 2021. URL: https://buildwithmatter.com/.

[42] Seyed Mahdi Darroudi and Carles Gomez. “Bluetooth Low Energy Mesh Networks: A Survey”. In: *Sensors* 17.7 (2017). ISSN: 1424-8220. DOI: 10.3390/s17071467. URL: https://www.mdpi.com/1424-8220/17/7/1467.

[43] Samuel DeBruin et al. “PowerBlade: A Low-Profile, True-Power, Plug-Through Energy Meter”. In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’15. Seoul, South Korea: Association for Computing Machinery, 2015, pp. 17–29. ISBN: 9781450336314. DOI: 10.1145/2809695.2809716. URL: https://doi.org/10.1145/2809695.2809716.

[44] Tess Despres et al. “Where the Sidewalk Ends: Privacy of Opportunistic Backhaul”. In: *Proceedings of the 15th European Workshop on Systems Security*. EuroSec ’22. Rennes, France: Association for Computing Machinery, 2022, pp. 1–7. ISBN: 978-1-4503-9255-6. DOI: 10.1145/3517208.3523757. URL: https://doi.org/10.1145/3517208.3523757.

[45] Drok. *Portable USB Doctor Multifunction Digital Meter*. June 2019. URL: https://www.droking.com/usb-tester/Portable-USB-Doctor-USB-Voltmeter-Ammeter-Capacity-Meter-Energy-Meter-Temperature-Meter-Running-Time-Tester-6in1-Multifunction-Digital-Meter.

[46] Ericsson. *CEO to Shareholders: 50 Billion Connections in 2020*. Apr. 2010. URL: https://www.ericsson.com/en/press-releases/2010/4/ceo-to-shareholders-50-billion-connections-2020.

[47] Espressif Systems. *ESP32*. July 2020. URL: https://www.espressif.com/en/products/socs/esp32.

[48] Fitbit. *Fitbit*. Jan. 2017. URL: http://fitbit.com.

[49] Flic. *Flic — The Smart Button for Lights, Music, Smart Home and More*. Mar. 2021. URL: https://flic.io/.

[50] Rasberry Pi Foundation. *Raspberry Pi*. Jan. 2017. URL: https://www.raspberrypi.org/.

[51] Adrian A de Freitas et al. “Snap-to-it: A user-inspired platform for opportunistic device interactions”. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI ’16. New York, NY, USA: ACM, 2016, pp. 5909–5920.

[52] George W. Furnas et al. “The Vocabulary Problem in Human-system Communication”. In: *Commun. ACM* 30.11 (Nov. 1987), pp. 964–971. ISSN: 0001-0782. DOI: 10.1145/32206.32212. URL: http://doi.acm.org/10.1145/32206.32212.

[53] J. Antonio Garcia-Macias et al. “Browsing the Internet of Things with Sentient Visors”. In: *Computer* 44.5 (May 2011), pp. 46–52. ISSN: 0018-9162. DOI: http://doi.ieeecomputersociety.org/10.1109/MC.2011.128.

[54] Matt Gaunt. *Service Worker*. Sept. 2019. URL: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker.

[55] Hans Gellersen et al. “Supporting Device Discovery and Spontaneous Interaction with Spatial References”. In: *Personal Ubiquitous Comput.* 13.4 (May 2009), pp. 255–264. ISSN: 1617-4909. DOI: 10.1007/s00779-008-0206-3. URL: http://dx.doi.org/10.1007/s00779-008-0206-3.

[56] Mohammad Ghamari et al. “Detailed Examination of a Packet Collision Model for Bluetooth Low Energy Advertising Mode”. In: *IEEE Access* 6 (2018), pp. 46066–46073. DOI: 10.1109/ACCESS.2018.2866323.

[57] Branden Ghena. “Investigating Low Energy Wireless Networks for the Internet of Things”. PhD thesis. EECS Department, University of California, Berkeley, Dec. 2020. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-209.html.

[58] Branden Ghena, Jean-Luc Watson, and Prabal Dutta. “Embedded OSes Must Embrace Distributed Computing”. In: *Proceedings of the 1st International Workshop on Next-Generation Operating Systems for Cyber-Physical Systems*. NGOSCPS’19. Montreal, Canada: Association for Computing Machinery, Apr. 2019. URL: https://www.cse.wustl.edu/~cdgill/ngoscps2019/papers/NGOSCPS2019_Ghena_etal.pdf.

[59] Branden Ghena et al. “Challenge: Unlicensed LPWANs Are Not Yet the Path to Ubiquitous Connectivity”. In: *The 25th Annual International Conference on Mobile Computing and Networking*. MobiCom ’19. Los Cabos, Mexico: Association for Computing Machinery, 2019. ISBN: 9781450361699. DOI: 10.1145/3300061.3345444. URL: https://doi.org/10.1145/3300061.3345444.

[60] Google. *Android Things*. Aug. 2019. URL: https://developer.android.com/things.

[61] Google. *ARCore Overview — Google Developers*. Nov. 2020. URL: https://developers.google.com/ar/discover.

[62] Google. *Google Developers*. July 2018. URL: https://developers.google.com/products/.

[63] Google. *Google Nest Smart Speakers and Displays*. Sept. 2021. URL: https://store.google.com/us/magazine/compare_nest_speakers_displays.

[64] Google. *IPv6 - Google*. Oct. 2021. URL: https://www.google.com/intl/en/ipv6/statistics.html.

[65] Google. *Nexus 4*. Jan. 2016. URL: http://www.google.com/nexus/4/.

[66] Google. *Nexus 9*. Jan. 2016. URL: http://www.google.com/nexus/9/.

[67] Google. *OpenThread — An Open Foundation for the Connected Home*. June 2021. URL: https://openthread.io/.

[68] Google. *Progressive Web Apps*. Sept. 2019. URL: https://developers.google.com/web/progressive-web-apps/.

[69] Google. *The Physical Web*. June 2017. URL: https://google.github.io/physical-web/.

[70] Google. *UriBeacon — The Web Uri Open Beacon for the Internet of Things*. July 2015. URL: https://github.com/google/uribeacon.

[71] Matthias Grossglauser and M. Vetterli. "Locating nodes with EASE: Last encounter routing in ad hoc networks through mobility diffusion". In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 3. Mar. 2003, 1954–1964 vol.3. DOI: 10.1109/INFCOM.2003.1209217.

[72] Immersive Web W3C Working Group. *WebXR Polyfill*. May 2020. URL: https://github.com/immersive-web/webxr-polyfill.

[73] Dominique Guinard, Vlad Trifa, and Erik Wilde. "A resource oriented architecture for the Web of Things". In: *2010 Internet of Things (IOT)*. Nov. 2010, pp. 1–8. DOI: 10.1109/IOT.2010.5678452.

[74] Dominique Guinard, Vlad Mihai Trifa, and Erik Wilde. "Architecting a mashable open world wide web of things". In: *Technical report/Swiss Federal Institute of Technology Zurich, Department of Computer Science*. Vol. 663. ETH Zurich, Feb. 2010.

[75] Albert F. Harris et al. "Smart LaBLEs: Proximity, Autoconfiguration, and a Constant Supply of Gatorade(TM)". In: *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. New York, NY, USA: IEEE, 2016, pp. 142–154. DOI: 10.1109/SEC.2016.43.

[76] Helena Project. *Squall: Low cost 1 inch round BLE sensor tag*. Sept. 2017. URL: https://github.com/helena-project/squall.

[77] Helium Systems. *Helium*. Jan. 2016. URL: https://helium.co.

[78] Valentin Heun, James Hobin, and Pattie Maes. "Reality Editor: Programming Smarter Objects". In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland: Association for Computing Machinery, 2013, pp. 307–310. ISBN: 9781450322157. DOI: 10.1145/2494091.2494185. URL: https://doi.org/10.1145/2494091.2494185.

[79] Valentin Heun, Shunichi Kasahara, and Pattie Maes. "Smarter Objects: Using AR Technology to Program Physical Objects and Their Interactions". In: *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '13. Paris, France: Association for Computing Machinery, 2013, pp. 961–966. ISBN: 9781450319522. DOI: 10.1145/2468356.2468528. URL: https://doi.org/10.1145/2468356.2468528.

[80] HTC. *HTC One*. Jan. 2016. URL: http://www.htc.com/us/smartphones/htc-one-m8/.

[81] HTTPArchive Mobile. *Interesting Stats*. Feb. 2016. URL: httparchive.org/interesting.php.

[82] Jonathan W. Hui and David E. Culler. "IP is Dead, Long Live IP for Wireless Sensor Networks". In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. 2008, pp. 15–28.

[83] Ke Huo et al. "Scenariot: Spatially Mapping Smart Things Within Augmented Reality Scenes". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–13. ISBN: 9781450356206. URL: https://doi.org/10.1145/3173574.3173793.

[84] Liviu Iftode et al. "Smart Phone: an embedded system for universal interactions". In: *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*. May 2004, pp. 88–94. DOI: 10.1109/FTDCS.2004.1316598.

[85] IFTTT. *IFTTT — Do More With the Things You Love*. Oct. 2021. URL: https://ifttt.com/.

[86] InfluxData. *InfluxDB Time Series Platform*. Dec. 2020. URL: https://www.influxdata.com/products/influxdb/.

[87] IoT Analytics. *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time*. Nov. 2020. URL: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[88] Sushant Jain et al. "Exploiting Mobility for Energy Efficient Data Collection in Wireless Sensor Networks". In: *Mobile Network Applications* 11.3 (June 2006), pp. 327–339. ISSN: 1383-469X. DOI: 10.1007/s11036-006-5186-9. URL: https://doi.org/10.1007/s11036-006-5186-9.

[89] Scott Jenson et al. "Building an On-ramp for the Internet of Things". In: *Proceedings of the 2015 Workshop on IoT Challenges in Mobile and Industrial Systems*. IoT-Sys '15. Florence, Italy: ACM, May 2015, pp. 3–6. ISBN: 978-1-4503-3502-7. DOI: 10.1145/2753476.2753483. URL: http://doi.acm.org/10.1145/2753476.2753483.

[90] Wha Sook Jeon, Made Harta Dwijaksara, and Dong Geun Jeong. "Performance Analysis of Neighbor Discovery Process in Bluetooth Low-Energy Networks". In: *IEEE Transactions on Vehicular Technology* 66.2 (2017), pp. 1865–1871. DOI: 10.1109/TVT.2016.2558194.

[91] Benjamin Kempke, Pat Pannuto, and Prabal Dutta. "PolyPoint: Guiding Indoor Quadrotors with Ultra-Wideband Localization". In: *2015 ACM Workshop on Hot Topics in Wireless*. HotWireless '15. Paris, France, Sept. 2015.

[92] Hyun-Soo Kim, Jungyub Lee, and Ju Wook Jang. "BLEmesh: A Wireless Mesh Network Protocol for Bluetooth Low Energy Devices". In: *2015 3rd International Conference on Future Internet of Things and Cloud*. IEEE. New York, NY, USA: IEEE, 2015, pp. 558–563. DOI: 10.1109/FiCloud.2015.21.

[93] Aniket Kittur, Ed H. Chi, and Bongwon Suh. "Crowdsourcing User Studies with Mechanical Turk". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 453–456. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357127. URL: http://doi.acm.org/10.1145/1357054.1357127.

[94] Neeraj Kumar et al. "Leafsnap: A Computer Vision System for Automatic Plant Species Identification". In: *Computer Vision – ECCV 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 502–516. ISBN: 978-3-642-33709-3.

[95] Ye-Sheng Kuo, Pat Pannuto, and Prabal Dutta. "System Architecture Directions for a Software-Defined Lighting Infrastructure". In: *1st ACM Workshop on Visible Light Communication Systems*. VLCS '14. Maui, Hawaii, USA, Sept. 2014.

[96] Ye-Sheng Kuo et al. "Luxapose: Indoor Positioning with Mobile Phones and Visible Light". In: *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. MobiCom '14. Maui, Hawaii, USA: Association for Computing Machinery, 2014, pp. 447–458. ISBN: 9781450327831. DOI: 10.1145/2639108.2639109. URL: https://doi.org/10.1145/2639108.2639109.

[97] Lab11. *BLEES: Bluetooth Low Energy Environmental Sensors*. Feb. 2017. URL: https://github.com/lab11/blees.

[98] Lab11. *Internet of Things Gateway*. June 2015. URL: https://github.com/lab11/iot-gateway.

[99] Lab11. *Opo*. Aug. 2016. URL: https://github.com/lab11/opo.

[100] Lab11. *Summon [Lab11] - Apps on Google Play*. Oct. 2016. URL: https://play.google.com/store/apps/details?id=edu.umich.eecs.lab11.summon.

[101] Lab11. *Summon [Lab11] on the App Store*. Oct. 2016. URL: https://apps.apple.com/us/app/summon-lab11/id1051205682.

[102] Lab11. *Summon: Browser for the Local Web of Things*. May 2017. URL: https://github.com/lab11/summon.

[103] Paul Lapides, Ehud Sharlin, and Saul Greenberg. "HomeWindow: An Augmented Reality Domestic Monitor". In: *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*. HRI '09. La Jolla, California, USA: Association for Computing Machinery, 2009, pp. 323–324. ISBN: 9781605584041. DOI: 10.1145/1514095.1514197. URL: https://doi.org/10.1145/1514095.1514197.

[104] Patrick Lazik, Anthony Rowe, and Nicholas Wilkerson. *ALPS: The Acoustic Location Processing System*. 2018. URL: https://www.microsoft.com/en-us/research/uploads/prod/2017/12/Patrick_Lazik_2018.pdf.

[105] Patrick Lazik et al. "ALPS: A Bluetooth and Ultrasound Platform for Mapping and Localization". In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. SenSys '15. Seoul, South Korea: Association for Computing Machinery, 2015, pp. 73–84. ISBN: 9781450336314. DOI: 10.1145/2809695.2809727. URL: https://doi.org/10.1145/2809695.2809727.

[106] Jangho Lee et al. "A Unified Remote Console Based on Augmented Reality in a Home Network Environment". In: *2007 Digest of Technical Papers International Conference on Consumer Electronics*. New York, NY, USA: IEEE, 2007, pp. 1–2. DOI: 10.1109/ICCE.2007.341516.

[107] Amit A. Levy et al. "Beetle: Flexible Communication for Bluetooth Low Energy". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore: ACM, June 2016, pp. 111–122. ISBN: 978-1-4503-4269-8. DOI: 10.1145/2906388.2906414. URL: http://doi.acm.org/10.1145/2906388.2906414.

[108] Haibin Ling. "Augmented Reality in Reality". In: *IEEE MultiMedia* 24.3 (2017), pp. 10–15. ISSN: 1070-986X. DOI: 10.1109/MMUL.2017.3051517.

[109] Can Liu et al. "Evaluating the Benefits of Real-time Feedback in Mobile Augmented Reality with Hand-held Devices". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, pp. 2973–2976. ISBN: 978-1-4503-1015-4. DOI: 10.1145/2207676.2208706. URL: http://doi.acm.org/10.1145/2207676.2208706.

[110] Jia Liu et al. "Adaptive Device Discovery in Bluetooth Low Energy Networks". In: *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*. IEEE. New York, NY, USA: IEEE, 2013, pp. 1–5. DOI: 10.1109/VTCSpring.2013.6691855.

[111] Jia Liu et al. "Energy Analysis of Device Discovery for Bluetooth Low Energy". In: *2013 IEEE 78th Vehicular Technology Conference (VTC Fall)*. IEEE. New York, NY, USA: IEEE, 2013, pp. 1–5. DOI: 10.1109/VTCFall.2013.6692181.

[112] Edward Lu et al. "FLASH: Video-Embeddable AR Anchors for Live Events". In: *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. New York, NY, USA: IEEE, 2021, pp. 489–497. DOI: 10.1109/ISMAR52148.2021.00066.

[113] Alexander Maier, Andrew Sharp, and Yuriy Vagapov. "Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things". In: *2017 Internet Technologies and Applications (ITA)*. IEEE. New York, NY, USA: IEEE, 2017, pp. 143–148. DOI: 10.1109/ITECHA.2017.8101926. URL: https://doi.org/10.1109/ITECHA.2017.8101926.

[114] Diana Marques and Robert Costello. “Skin & bones: an artistic repair of a science exhibition by a mobile app”. In: *MIDAS. Museus e estudos interdisciplinares* 5 (2015).

[115] Steven McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-Level Packet Capture”. In: *Proceedings of the USENIX Winter 1993 Conference Proceedings*. USENIX’93. San Diego, California: USENIX Association, 1993, p. 2. DOI: 10.5555/1267303.1267305. URL: https://dl.acm.org/doi/10.5555/1267303.1267305.

[116] Mateusz Mikusz et al. “Repurposing Web Analytics to Support the IoT”. In: *Computer* 48.9 (Sept. 2015), pp. 42–49. DOI: 10.1109/MC.2015.260.

[117] John Miller et al. *Realty and Reality: Where Location Matters*. 2018. URL: https://www.microsoft.com/en-us/research/uploads/prod/2017/12/John_Miller_2018.pdf.

[118] Mozilla. *Project WebXR Viewer: An AR Project by Mozilla*. Sept. 2019. URL: https://github.com/mozilla-mobile/webxr-ios.

[119] Mozilla. *WebXR Viewer JS*. May 2020. URL: https://github.com/MozillaReality/webxr-ios-js.

[120] Ritesh M Nayak. *Discontinuing support for Android Nearby Notifications*. Oct. 2018. URL: https://android-developers.googleblog.com/2018/10/discontinuing-support-for-android.html.

[121] Nest Labs. *Weave*. Mar. 2018. URL: https://nest.com/weave/.

[122] Nest Labs. *Works with Nest*. Mar. 2018. URL: https://nest.com/works-with-nest/.

[123] Johanna Nieminen et al. *IPv6 over BLUETOOTH(R) Low Energy*. RFC 7668. Oct. 2015. DOI: 10.17487/RFC7668. URL: https://www.rfc-editor.org/info/rfc7668.

[124] Johanna Nieminen et al. *Transmission of IPv6 packets over Bluetooth low energy “draft-ietf-6lo-btle-10”*. Feb. 2015. URL: https://tools.ietf.org/html/draft-ietf-6lo-btle-10.

[125] Nordic Semiconductor. *nRF51822 - Multiprotocol Bluetooth low energy/2.4 GHz RF System on Chip*. 2014. URL: https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.1.pdf.

[126] Nordic Semiconductor. *nRF52840 - Bluetooth 5.2 SoC*. Aug. 2019. URL: https://www.nordicsemi.com/Products/nRF52840.

[127] Hewlett Packard. *HP Reveal*. Apr. 2020. URL: https://www.hpreveal.com/.

[128] Unkyu Park and John Heidemann. “Data Muling with Mobile Phones for Sensornets”. In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’11. Seattle, Washington: ACM, 2011, pp. 162–175. ISBN: 978-1-4503-0718-5. DOI: 10.1145/2070942.2070960. URL: http://doi.acm.org/10.1145/2070942.2070960.

[129] David Pérez-Diaz de Cerio et al. “Analytical and experimental performance evaluation of BLE neighbor discovery process including non-idealities of real chipsets”. In: *Sensors* 17.3 (2017), p. 499. DOI: 10.3390/s17030499.

[130] Charles E. Perkins and Elizabeth .M. Royer. “Ad-hoc on-demand distance vector routing”. In: *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA ’99. Second IEEE Workshop on.* Feb. 1999, pp. 90–100. DOI: 10.1109/MCSA.1999.749281.

[131] Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. “Mind the Gap: Multi-Hop IPv6 over BLE in the IoT”. In: *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies.* CoNEXT ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 382–396. ISBN: 9781450390989. DOI: 10.1145/3485983.3494847. URL: https://doi.org/10.1145/3485983.3494847.

[132] Philips. *Smart Lighting — Hue.* Oct. 2021. URL: https://www.philips-hue.com/.

[133] Juri Platonov et al. “A mobile markerless AR system for maintenance and repair”. In: *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality.* Oct. 2006, pp. 105–108. DOI: 10.1109/ISMAR.2006.297800.

[134] Annebella Pollen. “Objects of Denigration and Desire: Taking the Amateur Photographer Seriously”. In: *The Handbook of Photography Studies.* London, UK: Bloomsbury Academic, 2020.

[135] Qualcomm Technologies. *When Mobile Apps Use Too Much Power: A Developer Guide for Android App Performance.* Dec. 2013. URL: https://developer.qualcomm.com/download/trepn-whitepaper-power.pdf.

[136] Dave Raggett. “The Web of Things: Challenges and Opportunities”. In: *Computer* 48.5 (May 2015), pp. 26–32. ISSN: 0018-9162. DOI: 10.1109/MC.2015.149.

[137] Niranjini Rajagopal, Patrick Lazik, and Anthony Rowe. “Visual light landmarks for mobile devices”. In: *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks.* Apr. 2014, pp. 249–260. DOI: 10.1109/IPSN.2014.6846757.

[138] Niranjini Rajagopal et al. “Welcome to My World: Demystifying Multi-user AR with the Cloud: Demo Abstract”. In: *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks.* IPSN ’18. Porto, Portugal: IEEE Press, 2018, pp. 146–147. ISBN: 978-1-5386-5298-5. DOI: 10.1109/IPSN.2018.00036. URL: https://doi.org/10.1109/IPSN.2018.00036.

[139] Jun Rekimoto and Katashi Nagao. “The World through the Computer: Computer Augmented Interaction with Real World Environments”. In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology.* UIST ’95. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1995, pp. 29–36. ISBN: 089791709X. DOI: 10.1145/215585.215639. URL: https://doi.org/10.1145/215585.215639.

[140] Michael Rietzler et al. "HomeBLOX: Introducing Process-Driven Home Automation". In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland: Association for Computing Machinery, 2013, pp. 801–808. ISBN: 9781450322157. DOI: 10.1145/2494091.2497321. URL: https://doi.org/10.1145/2494091.2497321.

[141] Christof Roduner. "BIT–A Browser for the Internet of Things". In: *Proceedings of the CIoT Workshop 2010 at the Eighth International Conference onPervasive Computing (Pervasive 2010)*. May 2010, pp. 4–12.

[142] Enrico Rukzio, Sergej Wetzstein, and Albrecht Schmidt. "A Framework for Mobile Interactions with the Physical World". In: *Wireless Personal Multimedia Communication*. WPMC '05. Aalborg, Denmark, Sept. 2005.

[143] Michele Ruta et al. "From the Physical Web to the Physical Semantic Web: Knowledge Discovery in the Internet of Things". In: *The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016)*. Oct. 2016.

[144] Samsung. *Smart Home - Home Monitoring, Smart Things*. Sept. 2019. URL: https://www.samsung.com/us/smart-home/.

[145] Teemu Savolainen and Minjun Xi. "IPv6 over Bluetooth low-energy prototype". In: *Aalto University Workshop on Wireless Sensor Systems, Aalto, Finland*. 2012.

[146] Rahul Anand Sharma et al. "All that GLITTERs: Low-Power Spoof-Resilient Optical Markers for Augmented Reality". In: *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. New York, NY, USA: IEEE, 2020, pp. 289–300.

[147] Tara Small and Zygmunt J. Haas. "Resource and Performance Tradeoffs in Delay-tolerant Wireless Networks". In: *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*. WDTN '05. Philadelphia, Pennsylvania, USA: ACM, 2005, pp. 260–267. ISBN: 1-59593-026-4. DOI: 10.1145/1080139.1080144. URL: http://doi.acm.org/10.1145/1080139.1080144.

[148] SmartBotics. *RoboSmart Wireless LED Light Bulb*. URL: http://www.smartbotics.com/%5C#!products/c218d.

[149] Michael Spörk et al. "BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. SenSys '17. Delft, Netherlands: Association for Computing Machinery, 2017. ISBN: 9781450354592. DOI: 10.1145/3131672.3131687. URL: https://doi.org/10.1145/3131672.3131687.

[150] Technical Machine. *Tessel*. Jan. 2016. URL: http://tessel.io.

[151] The Power Consumption Database. *The Power Consumption Database*. Jan. 2022. URL: http://www.tpcdb.com/list.php?type=11.

[152] Thread Group. *Thread*. July 2021. URL: https://threadgroup.org/.

[153] Tile. *Tile*. Jan. 2017. URL: http://thetileapp.com.

[154] Nguyen Khoi Tran et al. "Searching the Web of Things: State of the Art, Challenges, and Solutions". In: *ACM Computing Surveys* 50.4 (Aug. 2017), 55:1–55:34. ISSN: 0360-0300. DOI: 10.1145/3092695. URL: http://doi.acm.org/10.1145/3092695.

[155] UBM Technology Group. "2017 Embedded Markets Study". In: *Embedded Systems Conference*. Embedded Systems Conference. Boston, MA, USA, May 2017.

[156] D. Van Krevelen and R. Poelman. "A Survey of Augmented Reality Technologies, Applications and Limitations". In: *International Journal of Virtual Reality* 9.2 (June 2010), pp. 1–20. ISSN: 1081-1451. URL: http://www.ijvr.org/issues/issue2-2010/paper1%5C%20.pdf.

[157] W3C. *WebXR Device API - W3C Working Draft*. July 2020. URL: https://w3.org/TR/webxr.

[158] Daisuke Wakabayashi. "The Point-and-Shoot Camera Faces Its Existential Moment: As More Users Opt for Smartphones, Companies Wonder What's Next". In: *The Wall Street Journal* 262.21 (July 2013).

[159] Tianyi Wang et al. "CAPturAR: An Augmented Reality Tool for Authoring Human-Involved Context-Aware Applications". In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 328–341. ISBN: 9781450375146. URL: https://doi.org/10.1145/3379337.3415815.

[160] Roy Want, Bill N. Schilit, and Scott Jenson. "Enabling the Internet of Things". In: *Computer* 48.1 (Jan. 2015), pp. 28–35. ISSN: 0018-9162. DOI: 10.1109/MC.2015.12.

[161] Web Bluetooth W3C Community Group. *Web Bluetooth Draft Community Report*. Aug. 2019. URL: https://webbluetoothcg.github.io/web-bluetooth/.

[162] Alexander Wieland. "IPv6 over Bluetooth Low Energy on Android OS". MA thesis. Institute of Technical Informatics, Graz University of Technology, Aug. 2020. URL: https://diglib.tugraz.at/download.php?id=60a4ea6a34af4&location=browse.

[163] World Wide Web Consortium. *World Wide Web Consortium (W3C)*. Sept. 2019. URL: https://www.w3.org/.

[164] Lina Yao et al. "Unveiling Correlations via Mining Human-Thing Interactions in the Web of Things". In: *ACM Transactions on Intelligent Systems and Technology* 8.5 (June 2017), 62:1–62:25. ISSN: 2157-6904. DOI: 10.1145/3035967. URL: http://doi.acm.org/10.1145/3035967.

[165] Thomas Zachariah. *Cordova WebXR Plugin*. Oct. 2019. URL: https://github.com/tzachari/cordova-plugin-webxr.

[166] Thomas Zachariah, Joshua Adkins, and Prabal Dutta. "Browsing the Web of Connectable Things". In: *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks*. EWSN '20. Lyon, France: Junction Publishing, 2020, pp. 49–60. ISBN: 9780994988645. DOI: 10.5555/3400306.3400313. URL: https://dl.acm.org/doi/10.5555/3400306.3400313.

[167] Thomas Zachariah, Meghan Clark, and Prabal Dutta. "Bluetooth Low Energy in the Wild Dataset". In: *Proceedings of the First Workshop on Data Acquisition To Analysis*. DATA '18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 27–28. ISBN: 9781450360494. DOI: 10.1145/3277868.3277882. URL: https://doi.org/10.1145/3277868.3277882.

[168] Thomas Zachariah and Prabal Dutta. "Browsing the Web of Things in Mobile Augmented Reality". In: *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. HotMobile '19. Santa Cruz, CA, USA: Association for Computing Machinery, 2019, pp. 129–134. ISBN: 9781450362733. DOI: 10.1145/3301293.3302359. URL: https://doi.org/10.1145/3301293.3302359.

[169] Thomas Zachariah, Neal Jackson, and Prabal Dutta. "The Internet of Things Still Has a Gateway Problem". In: *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*. HotMobile '22. Tempe, Arizona: Association for Computing Machinery, 2022, pp. 109–115. ISBN: 9781450392181. DOI: 10.1145/3508396.3512881. URL: https://doi.org/10.1145/3508396.3512881.

[170] Thomas Zachariah, Noah Klugman, and Prabal Dutta. "ThingSpeak in the Wild: Exploring 38K Visualizations of IoT Data". In: *Proceedings of the Fifth Workshop on Data Acquisition To Analysis*. DATA '22. Boston, Massachusetts: Association for Computing Machinery, 2023, pp. 1035–1040. ISBN: 9781450398862. DOI: 10.1145/3560905.3567766. URL: https://doi.org/10.1145/3560905.3567766.

[171] Thomas Zachariah et al. "ReliaBLE: Towards Reliable Communication via Bluetooth Low Energy Advertisement Networks". In: *Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks*. EWSN '22. Linz, Austria: Association for Computing Machinery, 2023, pp. 96–107.

[172] Thomas Zachariah et al. "The Internet of Things Has a Gateway Problem". In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. HotMobile '15. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2015, pp. 27–32. ISBN: 9781450333917. DOI: 10.1145/2699343.2699344. URL: https://doi.org/10.1145/2699343.2699344.

[173] Lide Zhang et al. "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones". In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES/ISSS '10. Scottsdale, Arizona, USA: ACM, 2010, pp. 105–114. ISBN: 978-1-60558-905-3. DOI: 10.1145/1878961.1878982. URL: http://doi.acm.org/10.1145/1878961.1878982.

[174] Yang Zhang, Gierad Laput, and Chris Harrison. “Vibrosight: Long-Range Vibrometry for Smart Environment Sensing”. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST ’18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 225–236. ISBN: 9781450359481. DOI: 10.1145/3242587.3242608. URL: https://doi.org/10.1145/3242587.3242608.

[175] Yuhong Zhong et al. “XRP: In-Kernel Storage Functions with eBPF”. In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, July 2022, pp. 375–393. ISBN: 978-1-939133-28-1. URL: https://www.usenix.org/conference/osdi22/presentation/zhong.

[176] Zigbee Alliance. *Zigbee Alliance*. May 2019. URL: http://www.zigbee.org/.