Browsing the Web of Connectable Things

Thomas Zachariah, Joshua Adkins, Prabal Dutta
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720
{tzachari,adkins,prabal}@berkeley.edu

Abstract

As the number of Internet of Things devices continue to grow, a pattern of off-loading user control and interaction to individuals' mobile devices has emerged. The user experience, however, is burdened by high-friction tasks, including device discovery, app installation, scanning, pairing, and configuration. Additionally, the tools and systems currently employed to facilitate interaction vary in degree of usefulness, provide inconsistent support, and fail to mesh together in a meaningful way. This user experience model scales poorly with the increasing population of "things", and significantly hinders casual interactions with ambient devices, as well as regular interactions with persistent devices. To break away from this restrictive paradigm, we propose an architecture that provides a seamless, scalable approach to discovering and interacting with nearby "things" in both short- and long-term contexts. The system takes advantage of multiple network patterns and modern web technologies to supply users with rich device interfaces that can interact directly over local networking protocols. A mobile app-based implementation of this system is tested with several embedded wireless devices. In our analysis, we find that our method scales better than current popular models and that it enables powerful and complex functionality while remaining natural, intuitive, and secure for users.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—Web-based Interaction

General Terms

Design, Management, Performance, Human Factors, Standardization

Keywords

Internet of Things, Mobile Phones, Device Discovery, Bluetooth Low Energy, Web Browsing, User Interfaces

1 Introduction

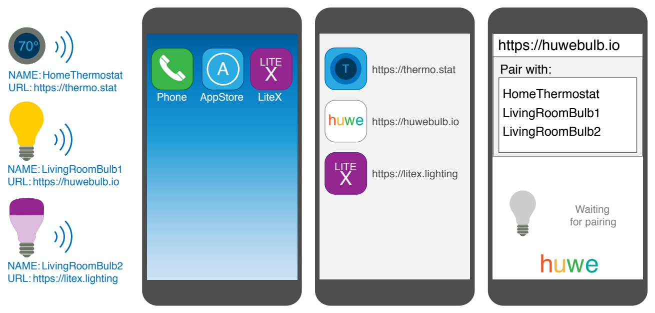
The embedded sensors and devices that make up the Internet of Things (IoT) primarily forego physical on-device user interfaces, like buttons and displays, in favor of more fully-featured software UIs in the form of native apps or websites that run remotely on personal devices like mobile phones. Recently, 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. 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 provide the user an interactive interface for the device. This expectation is not practical for many classes of smart device usage scenarios. Users should be able to browse nearby devices and immediately load dynamic, interactive, and context-specific user interfaces for devices that are not otherwise Internet-connected. Moreover, devices should be able 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) [21]. 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 mobile phone [18], as depicted in Figure 1b. While useful, it hides device infor-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2020
17–19 February, Lyon, France © 2020 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-4-5



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

Figure 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.

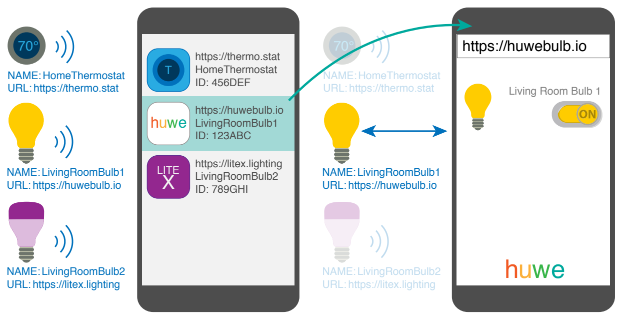


Figure 2: Proposed model of discovery, connection, and interaction. We suggest a scheme in which the device broadcasts a URL that is both a link to its UI and its declaration of origin. Users are shown a list of URLs along with details of corresponding devices associated with each. The user can select 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.

mation 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 [49], as depicted in Figure 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 2. Furthermore, we introduce an origin policy that treats devices as actual resources of their websites, which when enforced, enables seamlessness between discovery, connection, and interaction.

In this paper, 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 *Summon*—an implementation of the browser 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.

2 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.

2.1 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 [19], Apple Homekit [9, 8], Samsung SmartThings [44], Nest [36, 37]). 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 [12], W3C [50], Thread Group [45], ZigBee [55]). 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 [11], 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 [47], 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 [22, 21, 53] and extending features like search and web analytics [46, 34]. Prior work has explored spontaneous interaction with nearby devices [17, 51], use of smartphones as universal device remote controls [24], device abstraction for resource-sharing among several interfaces [33], discovery and interaction in mobile augmented reality [52], and device browsing with platform-agnostic interfaces [41, 42, 15, 10]. 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.

2.2 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 in one's proximity [26, 48]. In this model, the Eddystone beacon protocol is used to embed a URL in broadcasts of BLE devices that can prompt nearby smartphone to navigate to specific web pages [18].

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 [40, 43]. Google has notably removed Physical Web support from mobile platforms [4, 35].

2.3 Bluetooth from the Browser

Web Bluetooth is a newly-drafted W3C standard and JavaScript API that enables connection with BLE devices from websites [49]. 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

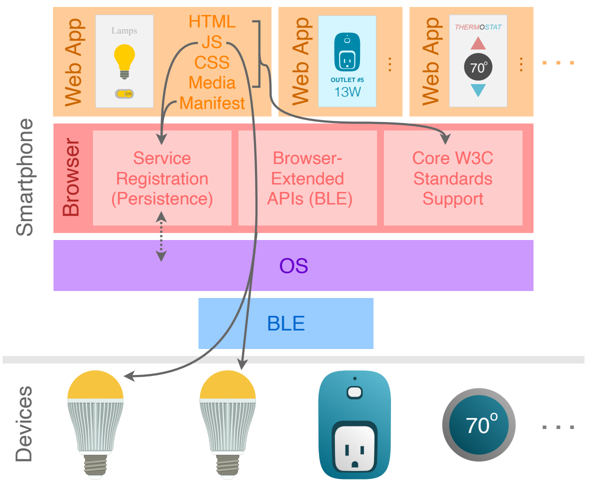


Figure 3: The Web App UI Model. Web apps may consist of typical web content: HTML, JavaScript, CSS, images, etc. An app can use a provided JS BLE API to interact directly with associated devices. By providing a manifest file for service registration, the web app can request special permissions and storage to enable persistent state for regularly-used devices.

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.

2.4 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, upcoming web and mobile standards are enabling fairly extensive app-like capabilities for websites [20, 16]. 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. While not yet implemented in iOS, Apple has stated their intent on supporting it in the near future [14].

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.

3 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.

3.1 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 [11], or Eddystone-URL protocol [18]. 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 [52].

3.2 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 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.

3.3 Device as 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.

3.4 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.

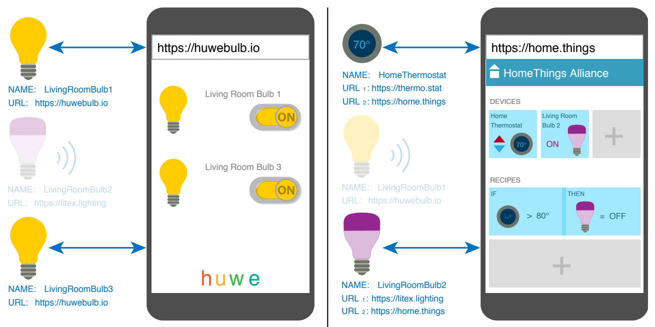
Furthermore, it would be useful for these apps to specify scripts to run in the background. For instance, a phone in the user's pocket could mule data for the device (with user permission), as in Figure 5a, or perform proximity based actions like turning on a light or setting the thermostat temperature to a preset setting when the user is near by, as in Figure 5b.

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 real background service. 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 6.2.

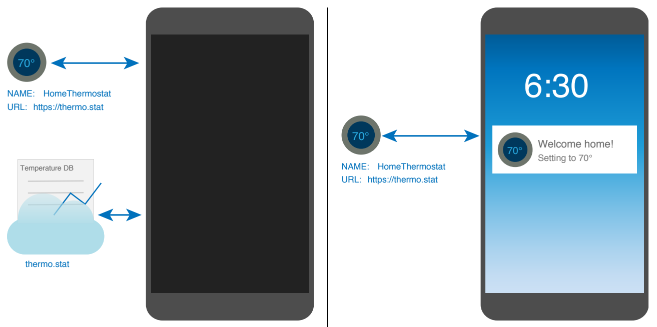
3.5 Aggregation

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.



(a) Aggregation (b) Orchestration

Figure 4: 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.



(a) Data Muling (b) Proximity-Based Action

Figure 5: 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.

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 4a. 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.

3.6 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 [8] or IFTTT [25] for devices the user owns or has access to, as shown in Figure 4b.

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.

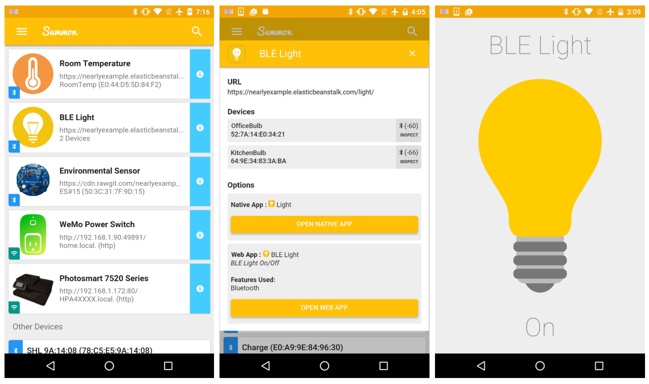
4 Implementation

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.

4.1 Browser App on Android and iOS

The browser is implemented as an application, called *Summon*, that runs on Android and iOS [30, 29]. 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 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 [6]. 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.


7:16

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)

4:05

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)

3:09

BLE Light

On

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

Figure 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).

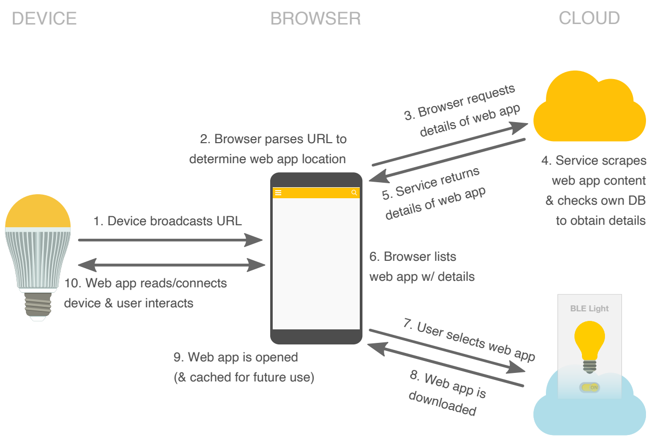

DEVICE
BROWSER
CLOUD
3. Browser requests details of web app
2. Browser parses URL to determine web app location
4. Service scrapes web app content & checks own DB to obtain details
5. Service returns details of web app
1. Device broadcasts URL
6. Browser lists web app w/ details
10. Web app reads/connects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

Figure 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.

Notably, *Summon* extends its own APIs, as 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.

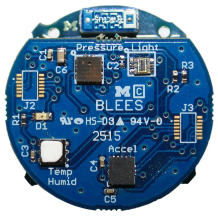
4.2 Destination Resolution

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.

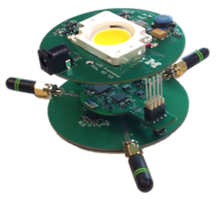
To assist the browser in obtaining information on the interfaces corresponding to URLs advertised by devices, we have set up a link resolution web service. The service sits in the cloud and responds to requests from the browser application. Figure 7 depicts the flow of this process. When the browser 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.

4.3 Caching

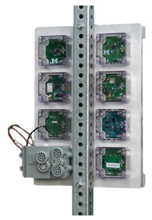
To further enable device interactions without requiring connection to the Internet, the implemented browser can also cache web apps. By default, the browser caches all of the web app's details when it is originally listed in the browser and caches the web app's resources when it is first opened. This 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.



(a) Environment Sensor (BLEES [31])

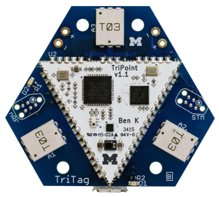


(b) Smart LED Light (Torch [28])

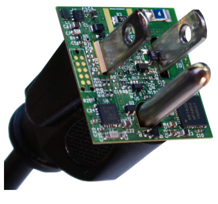


(c) Smart City Platform (Signpost [2])

(d) CO Breathalyzer (Monoxalyze [3])

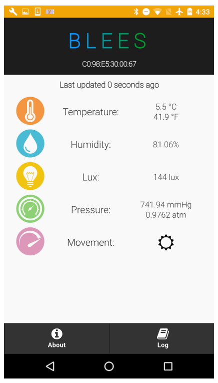


(e) Localization Tag (Polypoint [27])



(f) Power Meter (PowerBlade [13])

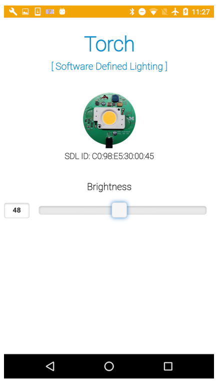
Figure 8: A selection of real "browsable" devices



BLEES
C0:98:E5:30:00:67
Last updated 0 seconds ago

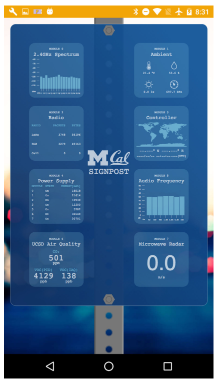
Temperature:	5.5 °C 41.9 °F
Humidity:	81.06%
Lux:	144 lux
Pressure:	741.94 mmHg 0.9762 atm
Movement:	

(a)



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

(b)



(c)

Figure 9: Subset of web apps for real devices. All interact directly with the corresponding devices (Figures 8a to 8c) directly over BLE using a browser-defined JavaScript API.

4.4 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 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.

4.5 Web Apps

Concurrently, a number of web apps have been built specifically for use with our browser application using standard web tools. Device advertisements link to the corresponding web app, which may be fetched online or from the phone itself (if the interface is cached), and can interact 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 9.

During a two year period, approximately 20 embedded developers created web apps for use with corresponding embedded devices and the 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 based on discussions about the relative ease of the development and deployment process for device UIs. 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 [32]. 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 Evaluation

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.

5.1 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 8 and Figure 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.

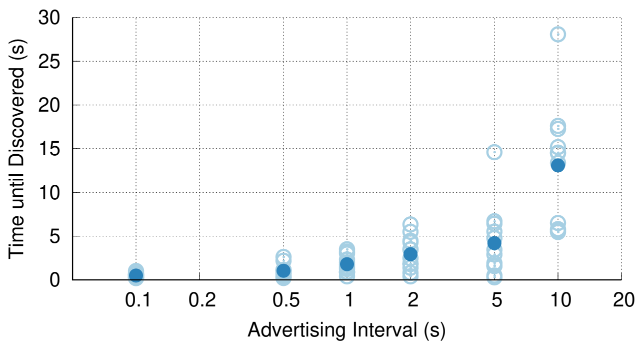


Figure 10: Device discovery latency for varying advertising intervals. The solid point is the mean of recorded latencies at each interval. Measurements taken on a Google Pixel.

Advertisement-only. The 1-inch round environmental sensor system shown in Figure 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 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 8b, continuously advertises its URL. After the browser discovers the device and its corresponding web app, shown in Figure 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 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.

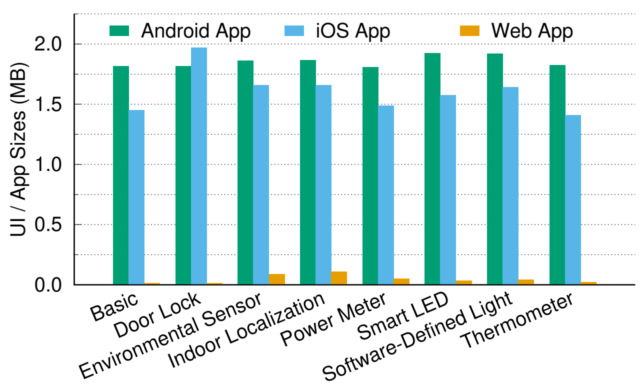


Figure 11: Size of example web apps vs native apps. Web app size accounts for bare web resources—HTML, JS, CSS, images, etc. Browser-provided JavaScript APIs allow web apps to use native smartphone features like BLE at run time. Because native apps repackage large commonly-used libraries into their binaries, they are significantly larger than web UIs.

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 9c, it can scan for, obtain data from, and display appropriate interfaces for each associated module.

5.2 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 10 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, have higher chance 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.

5.3 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 [23], the browser's environment seems to be more conducive of higher quantities of interfaces for short connections and

	Physical Web	Web Bluetooth	Phys. Web + Web BT	Native App	*Summon* (Our Browser)
Steps for ambient device/ UI discovery	- Open list screen - 1 (obfuscates device info) $\approx$ 1 gesture	N/A (no ambient discovery) $\approx \infty$ gestures	- Open list screen - 1 (obfuscates device info) $\approx$ 1 gesture	N/A (no ambient discovery) $\approx \infty$ gestures	- Open list screen - 1 $\approx$ 1 gesture
Steps for 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 - Create username - n - Create password - p - Press sign-up / log-in - 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 - Create username - n - Create password - p - Press sign-up / log-in - 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 - Create username - n - Create password - p - Press sign-up / log-in - 1 - Confirm/add device - 1 $\approx 6+q+n+p$ gestures	- Open list screen - 1 - Press listing - 1 - Create username - n - Create password - p - Press sign-up / log-in - 1 - Confirm/add device - 1 $\approx 4+n+p$ gestures
Steps for 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 $\approx$ 5 gestures	- Open app - 1 (app autoconnects) $\approx$ 1 gesture	- Open list screen - 1 - Press listing - 1 (UI autoconnects) $\approx$ 2 gestures

Table 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.

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.

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 11 shows that the size of web-based interfaces are low (~kBs) when compared to the size of native apps (~MBs). 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 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 1.

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, 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.

	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 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.

5.5 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 [38]. 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 [54] 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 [39] is used to help determine energy usage distribution among hardware components in finer detail. Table 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 ~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.

6 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.

6.1 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.

6.2 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 [7]). 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 5a, 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.

6.3 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.

6.4 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. Recent work also explores how augmented reality might improve the browsing approach by making discovery a more tangible user experience [52].

7 Conclusions

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 would finally provide a user-interfacing solution that adequately enables, supports, and handles the advent of an increasingly global, robust, and intimate Internet of Things.

8 Acknowledgments

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 also based upon work partially supported by the National Science Foundation under grant CNS-1824277, as well as by the NSF/Intel Partnership on CPS Security and Privacy under grant CNS-1822332.

9 References

[1] ABI Research. Average size of mobile games for iOS increased by a whopping 42% between march and september. https://www.abiresearch.com/press/average-size-of-mobile-games-for-ios-increased-by-/, Oct. 2012.

[2] J. Adkins, B. Campbell, B. Ghena, N. Jackson, P. Pannuto, and P. Dutta. The signpost network: Demo abstract. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*, SenSys '16, pages 320–321, New York, NY, USA, 2016. ACM.

[3] J. Adkins and P. 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, pages 190–201, New York, NY, USA, 2016. ACM.

[4] J. Adkins, B. Ghena, and P. Dutta. Freeloader's guide through the google galaxy. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, HotMobile '19, pages 111–116, New York, NY, USA, 2019. ACM.

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

[6] Apache Software Foundation. Apache Cordova. https://cordova.apache.org/, Mar 2019.

[7] Apple. Extending your app's background execution time. 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, Sep 2019.

[8] Apple. Homekit - apple developer. https://developer.apple.com/homekit/, Sep 2019.

[9] Apple. ios - home - apple. https://www.apple.com/ios/home/, Sep 2019.

[10] S. Bae, D. Kim, M. Ha, and S. H. Kim. Browsing architecture with presentation metadata for the internet of things. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 721–728, Dec 2011.

[11] Bluetooth Special Interest Group. Bluetooth core specifications. https://www.bluetooth.com/specifications/bluetooth-core-specification, Jul 2017.

[12] Bluetooth Special Interest Group. Bluetooth sig. https://www.bluetooth.com/, Sep 2019.

[13] S. DeBruin, B. Ghena, Y.-S. Kuo, and P. Dutta. Powerblade: A low-profile, true-power, plug-through energy meter. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, New York, NY, USA, 2015. ACM.

[14] M. Firtman. Pwas are coming to ios: Cupertino, we have a problem. https://medium.com/@firt/pwas-are-coming-to-ios-11-3-cupertino-we-have-a-problem-2ff49fd7d6ea, Jan 2018.

[15] J. A. Garcia-Macias, J. Alvarez-Lozano, P. Estrada-Martinez, and E. Aviles-Lopez. Browsing the Internet of Things with sentient visors. *Computer*, 44(5):46–52, May 2011.

[16] M. Gaunt. Service worker. https://developers.google.com/web/ilt/pwa/introduction-to-service-worker, Sep 2019.

[17] H. Gellersen, C. Fischer, D. Guinard, R. Gostner, G. Kortuem, C. Kray, E. Rukzio, and S. Streng. Supporting device discovery and spontaneous interaction with spatial references. *Personal Ubiquitous Comput.*, 13(4):255–264, May 2009.

[18] Google. The physical web. https://google.github.io/physical-web/, Jun 2017.

[19] Google. Android things. https://developer.android.com/things, Aug 2019.

[20] Google. Progressive web apps. https://developers.google.com/web/progressive-web-apps/, Sep 2019.

[21] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In *2010 Internet of Things (IOT)*, pages 1–8, Nov 2010.

[22] D. Guinard, V. M. Trifa, and E. Wilde. Architecting a mashable open world wide web of things. In *Technical report/Swiss Federal Institute of Technology Zurich, Department of Computer Science*, volume 663. ETH Zurich, Feb 2010.

[23] HTTPArchive Mobile. Interesting stats. http://mobile.httparchive.org/interesting.php, Feb. 2016.

[24] L. Iftode, C. Borcea, N. Ravi, P. Kang, and P. Zhou. Smart phone: an embedded system for universal interactions. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pages 88–94, May 2004.

[25] IFTTT. IFTTT helps your apps and devices work together. https://ifttt.com/, Sep 2019.

[26] S. Jenson, R. Want, B. N. Schilit, and R. H. Kravets. 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, pages 3–6, New York, NY, USA, May 2015. ACM.

[27] B. Kempke, P. Pannuto, and P. Dutta. Polypoint: Guiding indoor quadrotors with ultra-wideband localization. In *2015 ACM Workshop on Hot Topics in Wireless*, HotWireless '15, September 2015.

[28] Y.-S. Kuo, P. Pannuto, and P. Dutta. System architecture directions for a software-defined lighting infrastructure. In *1st ACM Workshop on Visible Light Communication Systems*, VLCS '14, September 2014.

[29] Lab11. Summon [Lab11] - apps on google play. https://play.google.com/store/apps/details?id=edu.umich.eecs.lab11.summon, Oct. 2016.

[30] Lab11. Summon [Lab11] on the app store. https://apps.apple.com/us/app/summon-lab11/id1051205682, Oct. 2016.

[31] Lab11. BLEES: Bluetooth low energy environmental sensors. https://github.com/lab11/blees, Feb. 2017.

[32] Lab11. Summon: Browser for the local web of things. https://github.com/lab11/summon, May 2017.

[33] A. A. Levy, J. Hong, L. Riliskis, P. Levis, and K. Winstein. Beetle: Flexible communication for bluetooth low energy. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 111–122, New York, NY, USA, June 2016. ACM.

[34] M. Mikusz, S. Clinch, R. Jones, M. Harding, C. Winstanley, and N. Davies. Repurposing web analytics to support the IoT. *Computer*, 48(9):42–49, Sep 2015.

[35] R. M. Nayak. Discontinuing support for android nearby notifications. https://android-developers.googleblog.com/2018/10/discontinuing-support-for-android.html, Oct 2018.

[36] Nest Labs. Weave. https://nest.com/weave/, Mar 2018.

[37] Nest Labs. Works with nest. https://nest.com/works-with-nest/, Mar 2018.

[38] Nordic Semiconductor. nRF51822 - multiprotocol bluetooth low energy/2.4 ghz rf system on chip. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.1.pdf, 2014.

[39] Qualcomm Technologies Inc. When mobile apps use too much power: A developer guide for Android app performance. https://developer.qualcomm.com/download/trepn-whitepaper-power.pdf, Dec. 2013.

[40] D. Raggett. The web of things: Challenges and opportunities. *Computer*, 48(5):26–32, May 2015.

[41] C. 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)*, pages 4–12, May 2010.

[42] E. Rukzio, S. Wetzstein, and A. Schmidt. A framework for mobile interactions with the physical world. In *Wireless Personal Multimedia Communication*, WPMC '05, Aalborg, Denmark, Sep 2005.

[43] M. Ruta, S. Ieva, G. Loseto, and E. Di Sciascio. 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.

[44] Samsung. Smart home - home monitoring, smart things. https://www.samsung.com/us/smart-home/, Sep 2019.

[45] Thread Group. Thread group. https://threadgroup.org, Jun 2019.

[46] N. K. Tran, Q. Z. Sheng, M. A. Babar, and L. Yao. Searching the web of things: State of the art, challenges, and solutions. *ACM Computing Surveys*, 50(4):55:1–55:34, Aug 2017.

[47] UBM Tech. 2017 Embedded markets study. In *Embedded Systems Conference*, Embedded Systems Conference, May 2017.

[48] R. Want, B. N. Schilit, and S. Jenson. Enabling the internet of things. *Computer*, 48(1):28–35, Jan 2015.

[49] Web Bluetooth W3C Community Group. Web bluetooth draft community report. https://webbluetoothcg.github.io/web-bluetooth/, Aug 2019.

[50] World Wide Web Consortium. World wide web consortium (w3c). https://www.w3.org/, Sep 2019.

[51] L. Yao, Q. Z. Sheng, A. H. H. Ngu, X. Li, and B. Benattalah. Unveiling correlations via mining human-thing interactions in the web of things. *ACM Transactions on Intelligent Systems and Technology*, 8(5):62:1–62:25, Jun 2017.

[52] T. Zachariah and P. 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, pages 129–134, New York, NY, USA, 2019. ACM.

[53] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The Internet of Things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 27–32, New York, NY, USA, Feb 2015. ACM.

[54] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. 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, pages 105–114, New York, NY, USA, 2010. ACM.

[55] Zigbee Alliance. Zigbee alliance. http://www.zigbee.org/, May 2019.