Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild

Tomasz Kalbarczyk
The University of Texas at Austin
tkalbar@utexas.edu

Christine Julien
The University of Texas at Austin
c.julien@utexas.edu

ABSTRACT

Device-to-device (D2D) communication technologies are growing in availability and popularity and are commonly used to facilitate applications in Internet of Things (IoT) environments. Such environments are characterized by heterogeneous devices, often employing diverse communication technologies with varying energy consumption, discovery ranges, and transmission rates. These complexities pose a daunting setting for the development of IoT applications that could leverage direct communication with proximal mobile and embedded devices. While current approaches focus either on device discovery in the IoT setting or content transfer assuming established communication channels, none facilitate the intelligent discovery of useful devices and the seamless formation of temporary D2D connections to transfer content directly between devices. Our Omni middleware provides both of these features critical in the development of applications that leverage proximal devices in IoT settings. Using Omni, we demonstrate the feasibility of building applications that use heterogeneous D2D communication channels in an efficient and realistic (in terms of energy and time) manner.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous and mobile computing systems and tools*; • **Networks** → Mobile networks;

KEYWORDS

internet of things, middleware, device to device communication

ACM Reference Format:
Tomasz Kalbarczyk and Christine Julien. 2018. Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild. In *19th International Middleware Conference (Middleware '18), December 10–14, 2018, Rennes, France.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3274808.3274821

1 INTRODUCTION

As wireless communication technologies become increasingly commonplace, it is possible to support novel opportunistic interactions among pervasive and mobile computing devices. Such interactions generally entail one device *discovering* the availability of *services* on another nearby device and then *interacting* with those services. Existing approaches to service discovery and interaction in mobile environments are semantically rich, but they assume a user's device has pre-established network connectivity to devices providing services or they require the user's device to spend time and energy discovering and connecting to such networks. These connections are most commonly established when a user's device authenticates in some way with a gateway, which provides the connectivity that mediates service discovery and interaction. Further, a given application is commonly implemented to use only a single communication technology; emerging domains such as smart cities or homes may require a single application to connect to remote resources using myriad different technologies, depending on the nature of a particular interaction and the capabilities of nearby devices.

We develop the Omni middleware, which leverages device-to-device (D2D) capabilities to provide opportunistic use of wirelessly available local area services provided by embedded Internet of Things (IoT) devices or made available on other end-user devices in the surroundings. Omni abstracts away details of underlying communication so that applications that leverage wirelessly connected services can be implemented in a technology-agnostic manner. Omni addresses a gap heretofore not addressed by other multi-network mobile service discovery middleware: the initial discovery of the neighboring devices that provide services. Omni integrates diverse neighbor discovery technologies with lightweight service discovery that can then seamlessly transition to more semantically expressive high-level service tasking and interaction, potentially using a different communication technology. That is, instead of assuming that the network-level discovery of neighboring devices is handled by some lower-level implementation, we ask what benefits can be garnered when we bring D2D neighbor discovery capabilities "into the fold" of service discovery and interaction.

The term *service discovery* often implies the use of canonical service discovery protocols. In Omni, we take a more general approach, wherein service discovery includes the discovery of any capabilities on wirelessly connected devices that the discoverer can leverage via wireless communication. This could include traditional views of services (e.g., connecting to and using a wireless printer), interactions more germane to mobile social networks (e.g., sharing profiles with nearby users) or interactions with embedded IoT devices (e.g., interacting with beacons in a smart city).

While bridging the gap between neighbor discovery and service interaction in the IoT is conceptually simple, actualizing the implementation is challenging because of the limitations of technologies for neighbor discovery. To ensure quick, low-energy discovery of transient peers, these neighbor discovery mechanisms typically exchange very small lightweight beacons, e.g., as used in discovery in Bluetooth Low Energy (BLE). Omni encapsulates a new paradigm for D2D interaction that explicitly differentiates between

Middleware '18, December 10–14, 2018, Rennes, France

ACM ISBN 978-1-4503-5702-9/18/12…$15.00
https://doi.org/10.1145/3274808.3274821

lightweight context and heavyweight data and tasks an appropriate available D2D technology based on the needs of a particular communication task (e.g., service discovery versus service interaction).

This effort leads to the observation that a subset of D2D communication tasks are periodic in nature and tend to entail sharing small amounts of data in a localized area. Both network-level neighbor discovery and application-level service discovery are examples. The remainder of D2D communication tasks entail intentional and directed (whether unicast or multicast) sharing of larger chunks of data with designated peer devices. Applications' interactions with available services are an example. Further, these different communication tasks have very different requirements in terms of network throughput, latency, and the need for energy optimization. Based on these observations, Omni provides an application programming interface (API) that explicitly separates periodic content from heavyweight data. We term the former *context* based on its common use by applications to quickly and efficiently ascertain conditions of the immediate surroundings.

The result is our Omni middleware, which extends the capabilities of existing multi-radio networking middleware [4, 26] to promote the explicit distinction of context and data and to achieve the delivery of periodic context by integrating low-level D2D discovery mechanisms. We demonstrate that Omni enables seamless formation of transient D2D networks using available communication technologies. As in prior work, multi-radio networking is invisible to applications; contrary to prior work, the conceptual separation of context and data enables substantial energy and latency efficiency not possible without considering the unification of neighbor discovery with service discovery and interaction, but critical to practicality when deployed on power constrained mobile devices [5]. The energy savings are achieved primarily because, unlike prior work, Omni only communicates across power hungry channels used for data after prompting from context. Omni also distinguishes itself by not relying on discovering and connecting to any pre-established networks (which entails expensive operations in protocols such 802.11) in order to retrieve information regarding neighboring devices and their services. Instead, Omni leverages lightweight neighbor discovery mechanisms to efficiently find peers and their services; forming connections only when data transfer is required. Omni presents these capabilities to application developers through an API that is similar to the asynchronous sending and receiving of requests that developers are already accustomed to, for instance in web APIs used in infrastructure networks.

Our concrete contributions are the following:

- We leverage innovations in D2D communication by connecting protocol-specific neighbor discovery with application-level service discovery and data transfer through temporary network formation that is invisible to applications.
- Omni differentiates lightweight contextual information (e.g., service availability or application specific context) from heavyweight data (e.g., service invocations that involve large data transfer) to optimize interactions.
- Omni supports the selection of an appropriate communication technology for data transfer without relying on pre-established network connections.
- Omni improves performance in terms of overall throughput, energy consumption, and latency. Omni provides application flexibility by not requiring an application to be statically tied to particular technologies or established networks.

In the next section, we discuss the myriad of related pieces of work, from communication technologies for neighbor discovery, to approaches for semantically-expressive service discovery in mobile networks, to existing middleware for multi-radio networking. We highlight related industrial efforts, particularly those related to supporting D2D interactions in the IoT. We use the related work, in conjunction with a detailed application scenario, to motivate the specific gaps that Omni addresses. Section 3 present Omni's conceptual architecture, built around the novel separation of context and data. In Section 4, we describe the current state of our implementation, and our evaluation that focuses largely on the efficiency and practicality gained in Omni. Beyond the multi-radio networking already achieved by existing middleware, we measure the benefits in energy usage, bandwidth utilization, and service discovery latency that come from Omni's approach.

2 BACKGROUND AND MOTIVATION

We start by reviewing the diverse related work that both supports and motivates Omni's contributions. We then present a detailed motivating application scenario and walk through the implementation of that scenario given the current state of the practice, the current state of the art, and what we envision with the Omni middleware.

2.1 Related Work

The challenges surrounding taking advantage of IoT devices in a D2D fashion can be separated into three parts: neighbor discovery, service discovery, and data transfer. In this section, we focus on approaches related to the first two since the main focus of Omni is on enabling efficient joint neighbor and service discovery in advance of the heavyweight data transfer.

Connecting devices to one another in a physical space is fundamental to emerging IoT applications [3]. Technologies used to support these capabilities range from ZigBee to BLE to WiFi and also include proprietary technologies. Off-the-shelf, these technologies have some limited neighbor discovery mechanisms built in; generally these approaches rely on devices assuming asymmetric *roles*, i.e., the discovered or the discoverer, and send and receive small, periodic beacon messages containing identity information. Research on continuous neighbor discovery, largely from the wireless sensor network domain [2, 8, 13–15, 21, 27, 28], promotes symmetric discovery in which a device can both discover and be discovered. These approaches build on the low-level technologies' lightweight beacon mechanism and jointly optimize discovery latency and energy consumption. EDiscovery [10] uses the number of discovered neighbors to guide subsequent discovery intervals. Other protocols use multiple radios to save energy by scanning for devices using a secondary low powered radio and waking the primary radio only upon discovery [23]. For example, ZiFi [31] leverages Zigbee to identify interference signatures from WiFi beacons.

While neighbor discovery is undoubtedly critical for IoT applications, it does not directly address the problem posed by a lack of

continuous connections among devices. Connectivity in opportunistic environments is expensive in time and energy, exacerbated by the brittle and transient nature of D2D connections. It is therefore critical for mobile devices to only connect to and transfer data with neighboring devices and peripherals that have relevant services and/or data. To support these interactions, proprietary and research approaches alike layer limited service discovery on these low-level beaconing mechanisms. For instance, LTE Direct [29] builds such mechanisms into time slices interspersed in regular LTE operations. Use cases for LTE Direct include enabling proximal interactions for social or business services or to connect first responders[1]. Because LTE Direct runs on licensed spectrum, users are subject to the restrictions of their carrier network. Moreoever, carriers are ultimately in control of the content transfer, and potentially even the content itself. Google's physical web uses proximal interactions over Bluetooth to discover and leverage a nearby device's capabilities[2]. iBeacons are used to support tourism[3] and smart cities[4]. Smart home devices from various manufacturers use a myriad of (often proprietary) technologies to connect users to deployed devices.

Many systems attempt to support multi-platform and technology-agnostic service discovery [1, 4, 9, 16, 19]. ubiSOAP [4] enables network-agnostic connectivity wherein applications can discover available services and communicate seamlessly across an array of heterogeneous communication technologies, choosing the technology that best serves an application's quality of service requirements. UbiSOAP relies on the existence of external mechanisms for network formation (such as previously paired Bluetooth devices or WiFi devices connected to the same Local Area Network). As discussed earlier, opportunistic IoT networking environments are characterized by the lack of such pre-existing connections. Haggle [22, 26] similarly abstracts away details of networking from the application, using the most suitable channels for data requests. However, network formation is limited to devices connecting to the same WiFi-adhoc network and making requests without any knowledge of the data availability from peers.

Bluetooth provides its own service discovery (Bluetooth SDP[5]). WiFi-Direct allows DNS-based service discovery. However, these approaches suffer in applicability in opportunistic networking environments. Bluetooth SDP requires devices to discover one another prior to exchanging service information, resulting in expensive (both in terms of time and energy) connections. DNS-based service discovery uses multicast to beacon service information; however, frequent sending and receiving of multicast packets in 802.11 is prohibitively expensive [7]. The lack of connectivity and *a priori* knowledge regarding neighboring devices in opportunistic networking environments necessitates frequent device/service discovery.

Bluetooth Low Energy Generic Attributes (BLE GATT) profiles define a standard for seamless discovery of services provided by nearby BLE peripherals[6]. Unfortunately, this standard is only designed for peripherals to communicate with a single device in a master-slave relationship. Beetle [17] provides a system service to work around this limitation, but the focus is on providing access across applications on the same device and not across multiple devices in the neighborhood. Other approaches enable seamless communication to multiple devices, assuming the existence of a mutually accessible WiFi gateway [12], but this requires devices to establish a connection to the gateway's WiFi network, which entails an expensive sequence of interactive operations.

Alljoyn[7] and Iotivity[8] unify IoT installations across manufacturers, but they rely on WiFi gateways to facilitate to communication with devices, so direct D2D operations are not supported. Thread[9] defines a single standard to which all IoT devices must conform. However, a heterogeneous set of protocols may be important since IoT devices are characterized by a wide range of limiting factors (energy, throughput, processing power, etc.).

In general, existing work has focused on many aspects of neighbor discovery and jointly service discovery and data transfer across varied communication technologies; however, a gap remains in how to translate neighbor and service discovery on one communication channel to data transfer on another. We next delve into how this gap manifests itself in modern application scenarios.

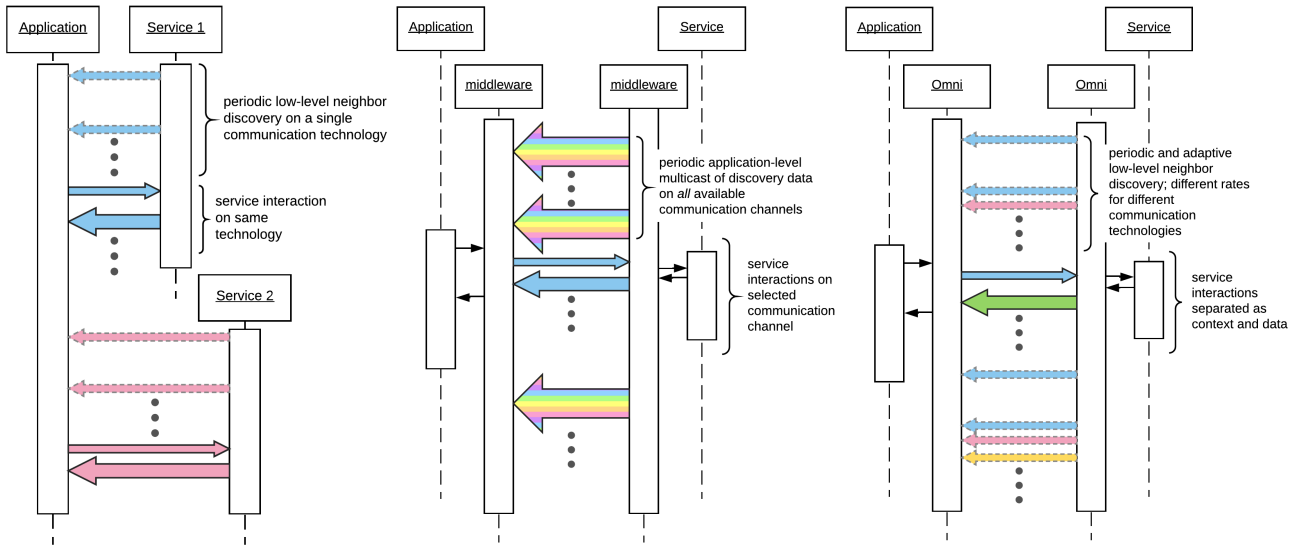
2.2 Motivating Applications

In the Internet of Things and other pervasive computing domains, applications on end-users devices expect to be able to directly leverage digitally enabled resources available in the immediate surroundings. In smart buildings, users entering a building, expect to be able to control the lights, the thermostats, or access maps and media about the building. In social pervasive computing applications, end users' devices exchange contact information, profiles, or even media with other users in the immediate surroundings [6, 20, 24]. An IoT enabled medical records application executing on a patient's or provider's tablet computer may need to incorporate medical data collected from several medical sensors in an exam room.

All of these applications entail (1) discovering nearby devices (at the physical network level); (2) discovering the potential capabilities (i.e., services) on those nearby devices; and (c) interacting with the other devices' provided services. All of this should be accomplished in a way that is transparent to the application (so as to ease the application development task) and can leverage the available communication facilities in the most resource efficient manner to provide pervasive services in a situation of potentially high mobility and dynamics.

Take as a concrete example a smart city tourism application. A given user enters the city as part of a tour group; his device should discover and connect to a device carried by the tour group's leader; this device will offer services, which may include the guide's real-time streaming audio, a map of the planned route, etc. The tourist's device might also connect to services provided by the city, for example smart crossing lights might expose a service that allows a user's device to activate the crossing light as he approaches. The city's historical buildings might have attached beacons that offer services that enable access to media about those buildings, including visualizations of how the buildings or streets looked in

[1] https://www.qualcomm.com/invention/technologies/lte/direct
[2] https://google.github.io/physical-web/
[3] www.imapp.it/portfolio-item/piacenza/
[4] blog.beaconstac.com/2016/02/internet-of-things-for-smart-cities-how-beacons-are-leading-the-way/
[5] https://www.bluetooth.com/specifications/assigned-numbers/service-discovery
[6] https://www.bluetooth.com/specifications/gatt/generic-attributes-overview
[7] https://openconnectivity.org/developer/reference-implementation/alljoyn
[8] https://iotivity.org
[9] https://www.threadgroup.org/



(a) The State of the Practice **(b) The State of the Art** **(c) The Vision of Omni**

Figure 1: Contrasting implementation options. Different colored arrows indicate different communication technologies. Dashed lines around arrows indicate the use of lightweight purely broadcast based discovery mechanisms; solid arrows indicate the use of established communication channels (e.g., an established multicast group, paired Bluetooth devices, etc.)

the past. Local businesses may provide services that expose their business hours, coupons or sales, or advertise a menu.

As a tourist walks through this digitally enabled world, his devices should seamlessly discover and interact with these services, regardless of the communication technology (e.g., NFC, Bluetooth, WiFi, etc.) that the service provider uses to expose the service. Further, the service interactions themselves are extremely heterogeneous. Some interactions entail the exchange of very small bits of data (e.g., business operating hours or wait times). Others, however, are much heavier weight (e.g., streaming audio or media files). Further, the tour group may be highly dynamic, connecting and disconnecting to service providers as the tour moves through the digitally enhanced city. Appropriately leveraging the best communication technology for each interaction is non-trivial for an application developer to navigate. Finally, the tourist desires responsive, low-latency interactions but will not sacrifice energy; the battery in the tourist's smart phone must last the entire day.

2.3 State of the Practice and State of the Art

Such applications are implementable using technologies and middleware available today to varying extents. Figure 1 contrasts the resulting implementations from a message passing standpoint. Figure 1(a) shows how such applications are most commonly implemented today. Managing communication capabilities is relegated entirely to the applications and services directly; as a result instead of providing general purpose application implementations, developers create solutions that tie application-service combinations to specific technologies. This results in limited interaction capabilities and implementations that are brittle and hard to maintain in the face of protocol updates. Figure 1(b) shows how existing multi-network middleware (e.g., ubiSOAP [4]) decouple the application and service implementations from the communication technologies. These implementations enable multi-network communication, and can allow a particular application/service interaction to select the most appropriate technology given stated quality of service requirements. However, because they layer on top of the communication technologies at the application layer, the result is that the applications and services advertise and discover using *all* of the available communication technologies and rely on pre-established communication channels (e.g., an established multicast group or WiFi direct group, paired Bluetooth devices, etc.). In ubiSOAP, for example these periodic application-level multicast discovery messages serve to maintain a multinetwork overlay, which is persistently available to the applications and services. This is a boon when the applications and services are ready to use the communication, but it also entails a significant amount of communication overhead. Further, multi-network middleware systems often do not even consider neighbor discovery; instead they start from an assumption that the application is already aware of (i.e., has discovered) the networks over which it wants to communicate. Figure 1(c) shows that, from the simplest perspective, Omni attempts to capture the best of both of these worlds: leveraging lightweight discovery mechanisms in an adaptive way, while still enabling highly flexible use of multiple network technologies. Omni attempts to achieve application functionality similar to the multi-network middleware but in a way that is more flexible, lightweight, and adaptable to changing networks.

3 ARCHITECTURE

Omni comprises three pieces: (1) the Developer API, through which application developers interact with Omni; (2) the Communication Technology API, which manages the modular integration of D2D communication technologies with Omni; and (3) the Omni Manager, which coordinates requests from the application and manages

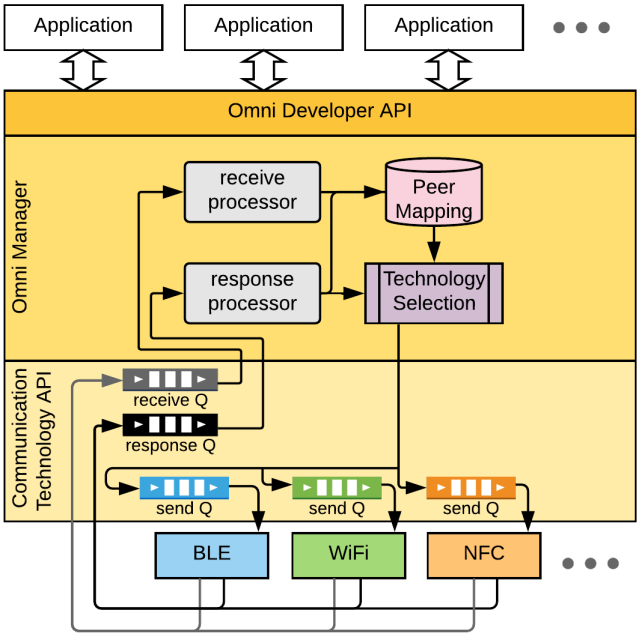


Figure 2: Overall Omni Architecture

content sent and received via each of the underlying D2D communication technologies. Figure 2 shows an abstract representation of the entire Omni system. The figure shows the three main components of Omni in the center, while the applications and implementations of the communication technologies themselves reside outside of Omni. In the remainder of this section, we provide details of the smaller pieces of the three primary Omni components.

Omni's primary function is to provide a communication abstraction to the application developer; we chose an abstraction that is similar to familiar asynchronous APIs used by web applications, with the added twist of allowing the developer to distinguish between sharing *lightweight context* and transferring *heavyweight data*. Omni processes requests for each type of communication task and then selects an appropriate communication technology for transmitting the content, based on the nature of the content, e.g., context or data, and on the connectivity situation in the current network environment. For context, Omni favors low-energy D2D communication technologies (e.g., Bluetooth Low Energy (BLE) Beacons); for data, Omni favors high-throughput communication technologies (e.g., WiFi-Mesh). In addition to the application-driven context and data, Omni also distributes address beacons, which simply serve to allow neighboring Omni devices to discover the presence of other nearby Omni devices. The mechanisms for distributing this discovery information is described in detail in Section 3.3.

On the receiving side, Omni-enabled devices must be prepared to receive content on any communication technology the device has available. Omni only distributes context on communication technologies with built-in energy-efficient neighbor discovery (e.g., as in BLE, ZigBee, or NFC). Data can be distributed on any communication technology; the Omni manager will choose the most suitable for a given interaction. Omni provides applications a mechanism for receiving content by allowing the applications to specify callback functions that are invoked when content is received (similar to asynchronous web APIs). Similar to other multi-network middleware approaches, Omni maintains an internal mapping of a device to its available communication technologies; each device is referenced using a single address, the `omni_address`. The application is exposed only to this unified address (and not the device's addresses for the underlying technologies); therefore the application can remain agnostic to the technologies used for content transfer.

To elucidate the functions and consequent benefits of Omni, we use a scenario based on the tourism application in Section 2.2. Imagine a tourist with a smart city tourism application on his device who joins an organized tour. In addition to receiving streaming audio from the tour guide, when the user passes landmarks, he should see interactive visualizations of what the landmarks used to look like. To realize this scenario, either each landmark needs to advertise that it provides this service or each tourist device needs advertise its interest in the service. State of the art applications[10] pre-program the landmark devices to transmit the service information via a technology such as BLE beacons to a user's smartphone. The smartphone uses this beacon to retrieve the corresponding landmark visualization from the Internet. However, there are a number of reasons why a D2D solution could be useful. A tourist device may not have access to the Internet (e.g., visiting a foreign country) or its user may not desire to use access to the Internet (e.g., for privacy or cost reasons). More importantly, the visualization may be dynamic and interactive itself based on the weather, the time of day or the users nearby. In such a scenario it makes sense for the landmark device to stream the visualization data to the user device over an appropriate and available D2D technology.

Omni allows either or both advertisement mechanisms (e.g., advertisements containing a tourist's interest or advertisements containing the beacon's service). These advertisements are sent as context in Omni, using lightweight, energy-efficient D2D discovery mechanisms, e.g., BLE beacons, if available on both devices. Subsequent interactions between the beacon device can use a mixture of technologies, and a mixture of context and data content, depending on the needs of the service. For instance, the bulk of the visualization data may be streamed as data over a WiFi-Mesh connection, while small updates to the situation of the visualization may be packaged as periodic context and shared over BLE. In this case, service advertisement/discovery and neighbor discovery could be accomplished using BLE, a technology suitable for continuous, periodic transmission (even during the periods when the tourist device is not near any landmarks). Mapping the discovered BLE address to the same device's WiFi interface is handled internally to Omni. No connections need to be formed and no data needs to be sent or received on the WiFi channel to facilitate this mapping.

Figure 3 shows this scenario in action. The example entails several tourist devices (to the left), which are smartphones capable of BLE, Wifi-Mesh, and NFC, with context sharing enabled on both BLE and NFC. There are also potentially beacons in the environment; the tourists' devices can communicate with these on a combination of BLE (primarily for context) and WiFi-Mesh (for data). The tourists' devices also communicate with the tour guide's device via a combination of BLE and WiFi-Mesh. The key demonstrated

[10] www.imapp.it/portfolio-item/piacenza/

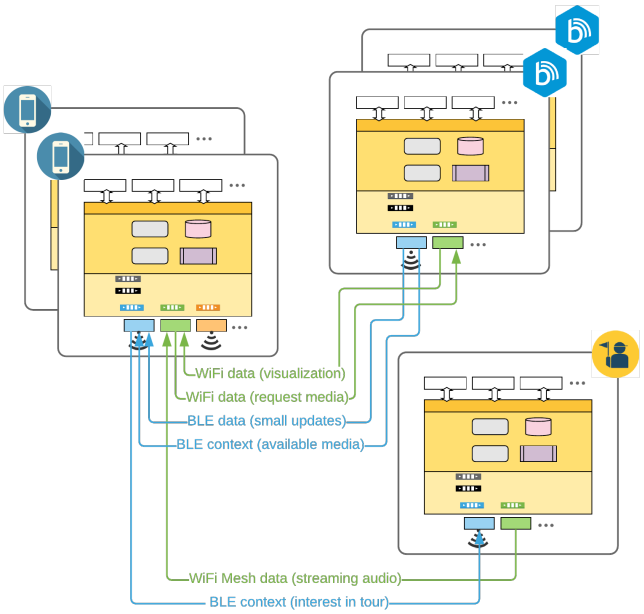


Figure 3: Omni in action in the tourist application. The radio symbols indicate the communication technologies that each device is using to distribute (and collect) Omni address beacons in support of low-level neighbor discovery.

benefits of Omni are (1) the application is agnostic to the communication technologies; (2) Omni is able to leverage the lightweight resource-aware neighbor discoveries when they are available; and (3) Omni selects the technology for data transfer without relying on a pre-established network connection.

Existing approaches suffer from a number of issues. Since existing multi-network middleware systems do not directly incorporate neighbor discovery, they need to know the WiFi-Mesh address of the landmark device *a priori*. Even with this knowledge, devices also need to periodically scan using the WiFi radio to detect the presence of the desired WiFi-Mesh device. Finally, existing approaches perform application-level multicast for service advertisement and discovery along all available networks. In this particular scenario, every device would transmit the service advertisement over both multicast WiFi (upon connecting to the WiFi-Mesh) and over BLE. All of these steps cause substantial energy drain due to the activity of the WiFi radio and would therefore not be feasible on a power constrained mobile device. By performing both neighbor discovery and service discovery using the same appropriate D2D technology, Omni offers a more efficient and practical solution.

The remainder of this section demonstrates how Omni achieves the components of this example application by separating content into data and context. We begin by presenting the Developer API and showing how it can be used to build this sample application.

3.1 Developer API

Omni provides a basic API through which developers can asynchronously delegate sending and receiving content. The simple API is designed to mimic familiar asynchronous web APIs. The Omni middleware will distribute context periodically to all neighboring devices; in contrast, data is sent directly to specific peers.

Omni runs as a system service; all applications interact with a single instance of Omni known as the `OmniManager`. The `OmniManager` participates in low-level neighbor discovery by exchanging address beacons, independent of application interactions. The remainder of the API, invoked via the singleton instance, allows application developers to leverage Omni's novel paradigm of bifurcating content into lightweight context and heavyweight data. The API is shown in Table 1. We next describe each segment of the API in more detail.

Status Callbacks. Omni allows applications to specify status callback methods to facilitate asynchronous responses to requests made by the application. A status callback has the following signature: `status_callback(code, response_info)`. The first argument, `code`, indicates the type of response, while the second argument carries additional details. A subset of these codes and their details are specified in Table 2. For errors, `response_info` provides details regarding the error where as for successes it contains the argument passed or an identifier associated with the successful request. More details regarding the use of these callbacks is presented with the discussion of the remainder of the API.

Sending Context. The API provides three methods to facilitate context transmission: `add_context`, `update_context` and `remove_context`. An application uses the `add_context` method to begin periodically sharing the specified context. The `parameters` argument contains the frequency with which the application wants to advertise the specified `context`. The application provides a `status_callback` to be notified of responses to the request. Omni expects the application to handle the `ADD_CONTEXT_SUCCESS` code to retrieve the corresponding `response_info` containing the reference identifier (`ID`) of the successfully added context transmission. The application can subsequently use the reference identifier to update the existing context transmission using the `update_context` method. The application can change the frequency, the context information itself, or specify a new callback. Similarly, to stop sharing a context, the application supplies the reference identifier to the `remove_context` method. For the tourism application on the tourist's device to transmit a context indicating interest in the media data from the landmark, the application needs to call the `add_context` method with the description of the interest. The format of the interest description is application-specific; its format is shared at the application level between the tourist application on the smartphone and the service implementation on the beacon.

Sending Data. The API provides a single method to send data: `send_data`. Omni allows the application to specify multiple destination peers for a piece of data using the `omni_address` for each peer. This address does not a particular communication technology to use to contact that peer. Under the hood, the Omni Manager delegates the data transmission to the most suitable D2D technology (or technologies). Similarly to context, the application supplies a `status_callback` to be informed regarding the status of the transmission. In our example application, when the landmark device sends the historical visualization data, it simply calls the `send_data` method, specifying the tourist device based on the tourist device's `omni_address`. As with the context information sent in the previous section, the particular format of the data is application-specific.

Receiving via Asynchronous Callbacks. To enable applications to receive context and data relayed via Omni, the API provides two simple methods, `request_context` and `request_data`,

Table 1: Omni API

Sending Context	
`add_context(params, context, status_callback)`	instructs Omni to share `context` periodically according to the frequency in `params`; the callback sends `ID` and other status to the application
`update_context(id, params, context, status_callback)`	changes the parameters, context information, or callback associated with a context pack (identified by`id`)
`remove_context(id, status_callback)`	instructs Omni to cease sharing the context pack identified by `id`
Sending Data	
`send_data(destinations, data, status_callback)`	instructs Omni to send specified `data` to the `destinations`; callback allows the application to be notified of the status of the send operation
Receiving Context and Data	
`request_context(receive_context_callback)`	registers a callback with Omni for the application to receive context packs that Omni receives
`receive_context_callback(source, context)`	signature of `receive_context_callback`; called when context is received
`request_data(receive_data_callback)`	registers a callback with Omni for the application to receive data that Omni receives
`receive_data_callback(source, data)`	signature of `receive_data_callback`; called when data is received

Table 2: Status Callback Codes

CODE	Response_Info
ADD_CONTEXT_SUCCESS	Context_ID
ADD_CONTEXT_FAILURE	Failure_Description
UPDATE_CONTEXT_SUCCESS	Context_ID
UPDATE_CONTEXT_FAILURE	(Failure_Desc, Context_ID)
REMOVE_CONTEXT_SUCCESS	Context_ID
REMOVE_CONTEXT_FAILURE	(Failure_Desc, Context_ID)
SEND_DATA_SUCCESS	Destination
SEND_DATA_FAILURE	(Failure_Desc, Destination)

used solely to allow the application to register callbacks for when context and data is received by Omni. The signatures of these callbacks are provided in Table 1. The `source` argument indicates the `omni_address` of the device that transmitted the content. The same `omni_address` can be used by the application to transmit additional context or data using the methods in the API described earlier.

In the example, the tourist's device would register the callback, `process_context(source, context)`, to retrieve the context describing the landmark's available service as well as the landmark device's address. The landmark device would register the callback, `process_data(source, data)` to retrieve the data request from the tourist's device, and the tourist's device would register a similar callback to ultimately retrieve the visualization data (as shown in Figure 3). An alternative implementation could swap the advertisement burden, with the landmark registering to receive context containing the tourist device's interest, and the tourist device simply receiving the visualization data. Both implementations have merit and are possible in Omni; specific application design decisions are left where they should be: in the application.

At no point must either side manually perform neighbor discovery, manage connections, or select the communication technology to use. Instead, neighbor discovery is handled transparently and continuously via address sharing (described in detail in Section 3.3); service advertisement/discovery is done using an appropriate connection-less technology (e.g., BLE); and data transfer is done via an appropriate high-throughput technology (e.g., WiFi-Mesh).

Handling Failures. When a request from the application fails, Omni attempts to resolve this failure internally by switching between D2D technologies to complete the request. However, in cases where communication across all available (and applicable) D2D technologies is unsuccessful, Omni employs the `status_callback` method provided by the application to notify the application of failures. Detailed discussion of how Omni handles switching between D2D technologies when one fails is provided in Section 3.2

Future Considerations. For flexibility, we currently allow the application to specify the frequency of context sharing. However, providing the frequency is potentially an unnecessary complication for a developer who does not understand the latency and energy implications of adjusting the frequency. In the future, we plan to allow a developer to omit this parameter in favor of plugging in existing neighbor discovery protocols that use adaptive transmission frequencies based on physical network conditions [10]. As shown in our evaluation, by leveraging modern technologies used for continuous neighbor discovery, our existing Omni implementation already offers improvements in energy consumption and service discovery latency. We make this note to indicate that Omni is designed to integrate sophisticated continuous neighbor discovery protocols to provide even more pronounced benefits.

3.2 Communication Technology API

Omni is designed to allow modular integration of new underlying D2D technologies. With this goal in mind, D2D communication technologies simply abide by a minimal contract. At the heart of this contract is a queue sharing system between the Omni Manager and each D2D technology, as shown in Figure 2. At initialization, each D2D technology is supplied with three queues shared with the Omni Manager: a *receive_queue* shared across all D2D technologies, a *response_queue* shared across all D2D technologies, and a *send_queue* unique to each D2D technology. We describe how each queue is used from the perspective of the D2D technology; we then describe the existing implementations for our two categories of D2D technologies: context and data.

Setup. To integrate with Omni, each D2D technology only needs to implement two methods. The `enable` method takes the three queues as arguments and returns a tuple containing the type of D2D technology and the low-level address where the technology is reachable. The `disable` method, which the Omni Manager uses

to disable the D2D technology and signal that it should gracefully shutdown by processing any remaining requests in its send queue and adding the requisite responses to the response queue.

The Receive Queue. As shown in Figure 2, all D2D communication technologies share a single `receive_queue` with the Omni Manager. Whenever a transmission of an `omni_packed_struct` is received by any technology, it is deposited in this queue for processing by the Omni Manager. To support a simple and modular integration of additional D2D technologies, the contents of this structure remain agnostic to the technology (details are discussed in Section 3.3). Instead, each D2D technology only includes the low-level address (for example, the Bluetooth MAC address or WiFi IP address) along with the type of D2D technology, so that the Omni Manager can properly process the *omni_packed_struct*.

The Send Queue. Each D2D communication technology is supplied by Omni with its own *send_queue*, which it monitors for requests to transmit `omni_packed_structs` according to `parameters` supplied with the send request as a map or dictionary structure. These requests contain both periodic context packets and one-time data packets. The `parameters` vary based on the type of request. For context, the frequency of transmission, the type of operation (add, remove, update), and optionally the identifier for the context (remove, update) are supplied. For data, only the type of operation (send) and the low-level destination address are supplied. Finally, a `status_callback` is supplied to be forwarded at response time.

The Response Queue. All D2D technologies and the Omni manager share a single `response_queue`, used to notify Omni of the status of a particular transmission or a change in the status of a D2D technology. After a D2D technology completes processing a request taken from its `send_queue`, it puts a response message into this `response_queue`. The response includes the `status_callback` (forwarded from the initial request), the response `CODE` (indicating success or failure) and the corresponding `response_info` (see Table 2). On failure, Omni also forwards all of the details from the send request, including the parameters and payload, since the Omni Manager needs this information to perform a re-transmission using an alternative technology. Details of the Omni Manager's failure handling are provided in Section 3.3. Finally, a response is also generated when the status of the D2D technology itself changes, for example, when the radio is turned off or the address changes.

Technologies for Distributing Context. Each D2D technology used by Omni for distributing periodic context packets must have some mechanism for performing neighbor discovery, ideally via a built-in resource-efficient neighbor discovery protocol. To support context transmissions, each of these technologies must support four operations corresponding one-to-one to those available in the Developer API: `add`, `update`, `remove`, and `receive` context.

Our implementation supports context transmission via BLE beacons or multicast UDP over WiFi-Mesh. Multicast over WiFi is provided as a proof of concept since it is one of the primary technologies used by state of the art solutions for address sharing and service discovery. However, as we show in our evaluation, multicast is not practical for continuous neighbor and/or service discovery on power constrained mobile devices. With new lightweight technologies for discovery on the horizon, such as WiFi-Aware (also known as Neighbor Awareness Networking), we aim to eventually replace multicast over WiFi as a technology for context transmission.

Technologies for Distributing Data. Each D2D technology that Omni uses to distribute data must support two operations: `send` and `receive`. Our implementation provides three: unicast TCP over WiFi-Mesh, multicast UDP over WiFi-Mesh and BLE beacons. In practice, unicast TCP is superior to multicast UDP (even when transmitting data to several peers), since existing implementations of multicast in 802.11 are slow to accommodate transmission to devices with the weakest signal strength and slowest radios.

Modularity Considerations. Omni's queues are designed with modularity in mind so that D2D technologies operate entirely separately from the Omni manager and only communicate using queues that can be accessed concurrently.

3.3 Omni Manager

In this section, we describe the primary structures at the heart of the Omni Manager, the core component that facilitates interoperability between the application and the D2D technologies. The primary functionality of the Omni Manager is to route application requests to transmit context and data to the appropriate D2D technologies and to maintain a mapping of available peers to the technologies on which they are accessible.

Peer Mapping. Upon initialization, the Omni Manager generates a unique 64-bit id for a device, known as the `omni_address`, using a hash of the hardware MAC addresses for the interfaces available on that device. This address is used by an application to identify itself to peer devices (and *vice versa*) since the application is agnostic to the D2D technologies used to service send and receive requests. The Omni Manager maintains a dynamic, real-time mapping of a peers' `omni_address` to the D2D technologies available at that peer. For each D2D technology, the necessary concrete addressing information is also provided to facilitate a connection with that peer using that D2D technology.

Context Mapping. The Omni Manager maintains a mapping of each currently active context transmission to the relevant D2D technology, allowing Omni to forward requests regarding context update or removal to the appropriate D2D technologies.

The Omni Address Beacon. To map `omni_addresses` to D2D technologies, Omni periodically transmits an `address_beacon`. For simplicity we have fixed the interval for this beacon to be every 500 ms. Every Omni-enabled device sends the `address_beacon` using its accessible D2D technology with lowest energy cost.

However, to accommodate discovering peers who cannot transmit using the same D2D technology, Omni uses a simple algorithm that engages the other available context D2D technologies to periodically send `address_beacons`. At a much lower frequency (e.g., every five seconds), the Omni Manager listens on each of the other available context D2D technologies. If the Omni Manager receives a beacon on a technology A from some unknown peer X, the Omni Manager begins periodically sharing contexts (including `address_beacons`) on technology A as well as listening on technology A. As long as beacons continue to arrive from at least one peer that is not also transmitting on a lower energy technology, Omni will continue employing technology A.

To actually transmit using a specific technology, Omni packages the address beacon into an `omni_packed_struct` (described next) and places it in each relevant D2D technology's `send_queue`.

The Omni Packed Struct. To minimize overhead, Omni tightly packs all content for transit into a sequence of bytes we call the `omni_packed_struct`. The first byte of every transmission indicates whether it is context, data, or an address beacon. The address beacons in Omni are completely hidden from the application. The following eight bytes are the `omni_address`. The remainder of the structure is a variable-length payload. Currently, 14 additional bytes are needed for the address beacon: 8 for the Wifi-Mesh address and 6 for the BLE address. The `omni_address` does not need to be transmitted with context and data, since the low-level address can be used to look up the `omni_address` at the receiver via the receiving device's peer mapping. However, by including the `omni_address`, we are able to refresh part of the peer mapping with each message.

Receiving Content. Omni receives content by monitoring the `receive_queue` shared with D2D technologies described in Section 3.2. Omni decodes any received `omni_packed_struct` to determine whether the content is context, data, or an address beacon. For context and data, Omni decodes the next eight bytes to retrieve the `omni_address`. Omni uses the address to update the peer mapping for the relevant D2D technology and calls either the context or data callback provided by the application (shown in Table 1).

Sending Content. Since Omni splits content semantically into context and data, it uses two mechanisms for transmitting content. For context, Omni follows the process described above for address beacons. Put simply, Omni transmits context using the lowest energy technologies that allow it to reach all known neighbors.

For data, Omni determines which D2D technologies are available at a designated peer and selects the technology that minimizes the expected time to deliver the data. Omni considers the expected throughput of the radio, the size of the data, and the time needed to form a connection. The data is packaged, the low-level address is specified, and the data is added to the relevant queue. In the future, we may implement more sophisticated selection to navigate a tradeoff between energy consumption and delivery time; since Omni is already designed to minimize the number of transmissions across high throughput and high energy D2D technologies (by allowing applications to intelligently select peers via context transmissions), this tradeoff is largely already accounted for, allowing applications to get the best of both worlds: frequent low-bandwidth and low-energy transmissions combined with infrequent high-bandwidth and high-energy transmissions.

Handling Failures. D2D technology failure is common, especially in opportunistic networks where peer accessibility is transient. If a D2D technology fails to send data or add/update a context, the technology generates a relevant response and places it in the shared `response_queue` (Section 3.2). The Omni Manager processes the response and attempts to use an alternative D2D technology until all applicable D2D technologies are exhausted. Only at this point is the `status_callback` provided by the application employed to indicate that the request failed.

Omni's approach to periodic context transmission using D2D technologies that require no network connectivity offers a substantial benefit in terms of resiliency to failures, since connection-less technologies by design have no connections to break. Since initial neighbor and service discovery is no longer tied to brittle connectivity to a transient network, Omni is able to recover more easily from environmental changes (such as peers leaving and entering proximity). When a connection to a network accessible via a connection-oriented D2D technology (such as WiFi-Mesh) breaks, Omni is still able to provide the application with dynamic information regarding the peers that remain available.

3.4 Security Considerations.

Security in terms of authentication of nearby devices and encrypting D2D traffic is important for applications operating in IoT environments. Although extensive discussion of security requirements is outside the scope of this paper, we briefly discuss some of the security challenges unique to these environments and how they can be addressed without disrupting Omni's functionality.

Omni allows applications to interact with unknown devices, which presents potential security vulnerabilities that would not be present if the application were operating in a traditional secured infrastructure network. To address this issue, beacons for sharing context can be encrypted using symmetric encryption. The key to decrypt the beacon could be shared out of band, for example, by registering the user device with a centralized authority using an infrastructure network. While this means that the application would not be strictly D2D, a single provisioning step does not present, in our view, an unreasonable requirement, especially when many applications on the market already necessitate user registration.

Wifi-Mesh supports Simultaneous Authentication of Equals (SAE) [11] for secure password-based authentication. This password could be shared securely as context via encrypted beacon.

Additionally, Omni may reveal information across applications that might not otherwise be available presenting a potential security issue. One mitigating factor is that we envision Omni to be implemented as an operating system service that filters content by application and invokes the receive callbacks provided by each application when relevant.

4 EVALUATION

We prototyped Omni for evaluation on a Raspberry Pi testbed. While in a commercial deployment, Omni would need to run all of the above applications on consumer devices, for ease of iterative development and availability of more modern D2D features, we implement the Omni prototype in Python on Raspberry Pi 3s (Model B) running Ubuntu 16.04. The Raspberry Pis mimic the constraints in processing power associated with mobile devices, while providing flexibility in low-level implementations of D2D technologies. All of the Raspberry Pi hardware features we leverage could be feasibly made available on any mobile device (e.g., an Android device).

Our Python implementation matches the description of the architecture with one limitation: the prototype creates a singleton instance of Omni for each application, mainly for simplicity in development. Omni would most appropriately be implemented as an operating system service shared among applications. Many optimizations can leverage overlap in requests from applications (for example, consolidating context into fewer beacons), address beacons can be aggregated, and the leakage across applications mentioned in Section 3.4 can be prevented.

Our prototype provides the API in Table 1. We first benchmark the behavior of Omni against systems providing the State of the

Art and State of the Practice capabilities, as described in Section 2. We then show Omni's behavior "in the wild" in a pair of real-world examples from the literature. The goal of these application-driven evaluations is two-fold: we exercise the Omni API, and we demonstrate how Omni improves the performance of these applications relative to other approaches. Both of these applications are from the Delay Tolerant Networking (DTN) literature, where reliance on D2D connections is already commonplace. DTNs are characterized by a lack of persistent end-to-end connections between devices and are therefore highly reliant on transient D2D connections among devices. As motivated in Section 2.2, we expect interactions very similar to these to become commonplace in the IoT.

The first example is a D2D file-sharing application in which co-located users download media (e.g., photos and videos) from an infrastructure network and share them among themselves [30]. In particular, we replicated the general behavior of Disseminate, in which devices exchange meta-data describing their available and desired data before exchanging the (much larger) data itself [25]. In the second example, we layer Omni underneath the PRoPHET DTN router [18], in which information is buffered by intermediate devices and then forwarded when communication links are available. PRoPHET selects devices as carriers based on a local assessment of their potential to encounter the final destination. To assess these conditions, devices continuously share summaries of their historical encounters with neighboring peers.

We compare Omni to an implementation representative of the State of the Practice (SP) and one representative of the State of the Art (SA). For the former, we implement the applications to directly interact with the underlying communication technologies. For the latter, existing multi-radio middleware systems are dated and lack support for modern D2D technologies, especially those supporting neighbor discovery. Since a direct comparison is therefore not possible, we implement a generalized multi-radio approach that contains the relevant features to operate in our setting, including support for the new D2D technologies, but adopts the paradigms specific to these approaches. In particular, these approaches do not integrate with low-level neighbor discovery and instead interact with D2D communication protocols only at their provided application-level APIs. The goal of this comparison is to demonstrate the efficiency of leveraging neighbor discovery and the distinguishing between context and data as part and parcel of providing transparent application interaction across multiple communication technologies.

4.1 Baseline Measurements

Before jumping into the application-level characterizations of Omni and comparisons to other approaches, we benchmark our testbed environment with respect to D2D communication technologies. Our prototype implementation of Omni relies on the integration of two underlying D2D technologies: BLE and WiFi-Mesh; we implemented the interfaces of these technologies to Omni through the Communication Technology API as exemplars. To support WiFi-Mesh, we custom-fit our Raspberry Pis with 2.4 Ghz USB WiFi Adapters (Atheros AR9271) operating on 802.11n. For BLE, we use the onboard Bluetooth radio running a BlueZ 5.48 stack. To measure the current draw on the Raspberry Pi, we connect an AVHzY CT-2 USB power meter between the power supply and the Raspberry Pi.

Table 3: Baseline current draw for D2D technology operations in Raspberry Pi testbed

Operation	Current (mA)
WiFi-receive	162.4
WiFi-send	183.3
WiFi-scan for networks	129.2
WiFi-connect to network	169.0
BLE-scan	7.0
BLE-advertise	8.2

We used this setup (without Omni or any other middleware) to characterize the energy consumption for the D2D technologies. We compute the current of the device at steady state, where both the Bluetooth and WiFi radios are switched off. Using this value as a floor, we also compute the current draw during WiFi-standby (92.1 mA) which remains constant during the operation in our experiments (unless the WiFi radio is turned off completely). The BLE-standby current draw is too low to detect with our hardware, and we assume it to be 0 mA. Recall that Omni shares all accessible D2D technologies and their addresses using its `address_beacon`, hence the WiFi-Mesh technology must be assigned some ip address to be reachable. This is not unique to Omni; in fact, all other multi-radio middlewares who need to periodically transmit multicast address and service information messages along WiFi do the same. Given this new floor, we report the peak current draw (in mA) for both BLE and WiFi-Mesh in Table 3. The values in Table 3 are the peak current draw for the given operation, relative to WiFi-standby.

Although the raw values are not identical to those that would result from the same tests on a mobile phone, they follow the expected trend. Specifically, analogous operations using BLE and WiFi generally require substantially less energy when using BLE. This supports our thesis that for small payloads such as those involved in exchanging addressing information or service information, lightweight technologies such as BLE offer substantial energy savings. For example, transmitting a single service request (WiFi-send) using multicast over WiFi-Mesh causes a spike in current draw by 183.3 mA. Meanwhile, transmitting the same request using BLE (BLE-advertise) draws over an order of magnitude less current. Said another way, it is not sufficient simply to determine which among many network technologies is best suited for transmitting application data; one must also consider what technologies are best suited for discovery in the first place. In scenarios where neighboring devices are accessible on a low-energy D2D communication technology, Omni can select it as a best-suited *context* technology for transmitting address and service information, whereas other approaches need to transmit on all available technologies. While instantaneous current draw is not directly indicative of overall energy consumption, these baseline measurements help elucidate the differences in energy consumption that we observe when computing current draw over time in the following subsections.

4.2 Controlled Comparisons

We characterize the State of the Practice, the State of the Art, and Omni. We assume a pair of devices engaged in service discovery followed by service interaction. We measured the energy consumption and interaction latency on the initiating device. In particular, the "application" on this device remained idle for 60 seconds. Instead of

Table 4: Performance comparison across approaches

Context Tech.	Data Tech.	Total Energy (avg. mA) SP	SA	Omni	Service Latency (ms) SP	SA	Omni
BLE	BLE	-92.07	23.47	7.52	82	82	82
BLE	$WiFi_{30B}$	N/A	22.25	9.11	N/A	2793	16
BLE	$WiFi_{25MB}$	N/A	43.41	36.14	N/A	5982	3112
WiFi	BLE	N/A	N/A	N/A	N/A	N/A	N/A
WiFi	$WiFi_{30B}$	21.86	22.60	23.12	3216	3175	3229
WiFi	$WiFi_{25MB}$	39.78	42.03	41.41	6499	6013	6162

active application behavior, during this time, the underlying system transmitted address and service information every 500ms. In the State of the Practice implementation, this periodic transmission was programmed by hand. In the State of the Art, we modeled behavior like that exhibited in ubiSoap [4][11], where this discovery information is transmitted via multicast on all of the active D2D communication technologies.[12] In Omni, this discovery relies on the available technology with the lowest energy consumption, and uses other technologies only as described in Section 3.3.

After this initial warmup period of 60 seconds, the device performs a send and receive interaction with the discovered remote service. The interaction with the remote service entailed transferring either 30 bytes of data (some small sensor reading, for instance) or 25MB of data (some large media file).

Table 4 shows the overall energy consumption and latency of service interaction for different pairs of technologies; Figures 4 and 5 show the information pictorially. When BLE is used as the data technology, the size of the data is 30 bytes; BLE packets cannot carry the larger data file. We did not use the combination of WiFi-Mesh for context and BLE for data; if both technologies are available, no application would choose this combination. In the State of the Practice, we assume that a natively implemented application will use only one technology for both context and data (as is commonly the case for modern service discovery to service interactions patterns); for these reasons, we do not have results for the State of the Practice for the combined BLE/WiFi-Mesh case.

The energy consumption values are relative to baseline operation. In the case of the State of the Practice, when BLE is used for both context and data, the WiFi radio can be turned off entirely, resulting in a negative relative energy consumption. In the State of the Art and Omni, because both approaches need to be available on all of the active D2D communication technologies, the WiFi radio must remain on. In the State of the Art, address packets are sent on both BLE and WiFi-Mesh, resulting in the increased relative energy usage. Latency values are measured in milliseconds from the initiation of the service interaction until its completion. The dramatically higher latency for the State of the Practice is also due to the fact that application-level multicast is used for address discovery. This entails periodic WiFi scans for relevant networks (WiFi-Mesh or otherwise) and connections to such networks to send multicast packets. Omni achieves a big win relative to the State of the Art because it leverages lightweight neighbor discovery and separates lightweight context to be periodically disseminated from the heavier weight application-level data.

[11] Based on the documentation at: https://gforge.inria.fr/frs/?group_id=699

[12] One might argue that these approaches only transmit this information upon connecting to a network, and that we are mischaracterizing the necessary transmissions. However, to support IoT applications, discovery must handle constantly changing environments where the available networks cannot be assumed to be known *a priori*.

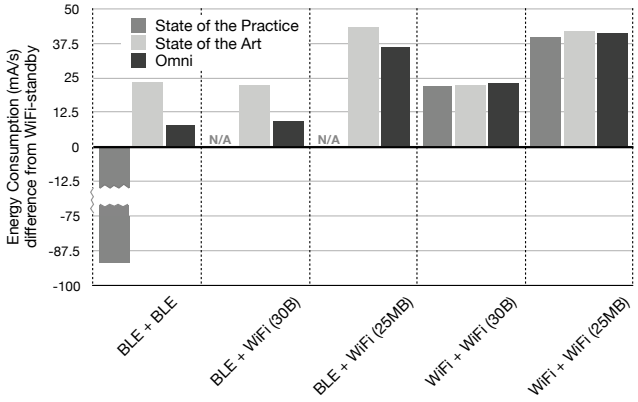


Figure 4: Energy Consumption Comparison

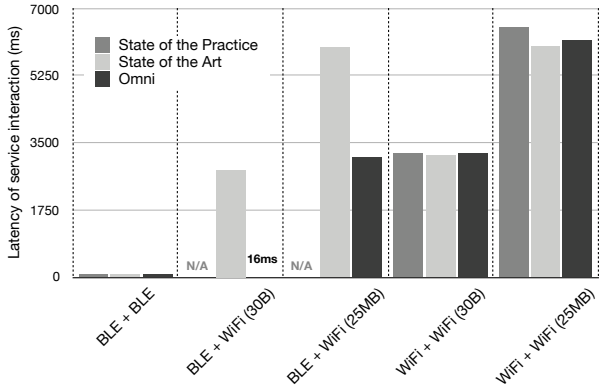


Figure 5: Application Interaction Latency Comparison

4.3 Real Application Comparisons

To demonstrate Omni's behavior in the wild relative to other approaches, we conducted experiments for the two applications described earlier. We show the performance of the State of the Practice (i.e., how the application would currently be implemented), the State of the Art (i.e., how the application would be implemented using a multi-networking middleware like that in Section 2), and Omni.

Our goal in running experiments with an application similar to Disseminate is to showcase Omni's performance differences in a high-throughput application relative to other approaches. In our experiment, three devices initiate a download of pieces of a single 30 MB file from a mock infrastructure network using two different data rates (100 KBps and 1000 KBps). Table 5 and Figure 6 show the times of transfer and energy consumed by an arbitrary device downloading the entire file (1) directly from the infrastructure and (2) by collaborating with the other two devices using the three implementation options: State of the Practice, State of the Art, and Omni. As above, the State of the Practice approach purely uses multicast over WiFi-Mesh which, as described in Section 3.2, is often slow, particularly in an application scenario where high-throughput is the goal. As such, it is outperformed by the other collaborative approaches. While the average energy consumed by the State of the Practice appears lower (due to the slower transfer rate), the longer duration of transfer actually means that more energy is consumed to retrieve the entire file. By multiplying the average energy consumed by time required, we can retrieve the current

Table 5: Energy and latency for Disseminate-like application over different implementations. Device collaborates with two other devices to download 30 MB file. Time and Energy measured from first transmission until device downloaded entire file.

		Measurement Type	Direct Download	SP (WiFi only)	SA (BLE + WiFi)	Omni (BLE + WiFi)
Rate (Kbps)	100	Avg energy consumed (mA)	N/A	72.39	67.12	66.91
		Time to complete download (s)	300	229.588	102.679	101.292
	1000	Avg energy consumed (mA)	N/A	80.03	267.79	270.288
		Time to complete download (s)	30	30	13.100	11.965

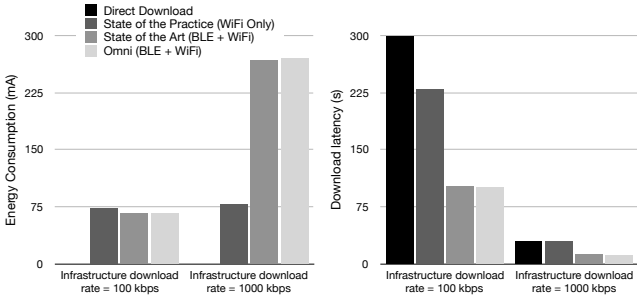


Figure 6: Energy and transfer time for D2D media downloads under different implementation options

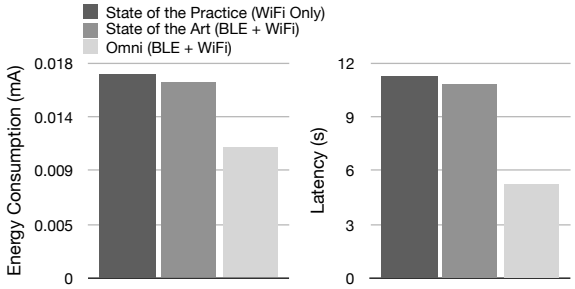


Figure 7: Energy and Latency for Prophet interactions under different implementation options

dissipated in each case. For the 100 KBps rate, this is 6777 mAs for Omni and 16619 mAs for the State of the Practice.

The more interesting comparison is between the State of the Art and Omni. The State of the Art needs to periodically transmit multicast packets, even on WiFi-Mesh; these packets impede the overall transfer rate. While this effect is marginal when the channel is not congested (the 100 KBps scenario), at higher data transmission rates (the 1000 KBps scenario), we see a lower transfer time for Omni relative to the State of the Art. The State of the Art takes 13.10 seconds, while Omni only takes 11.97 seconds (Table 5). While a delta of 1.13 seconds may not seem like much in absolute, it represents an 8.6% drop, which becomes substantial given multiple transmissions. Besides the benefits to throughput, the shorter transmission time means that less overall energy is consumed. Figure 6 shows that although in Omni's case data is transmitted at a faster rate, the average energy consumed is marginally higher than for the State of the Art. For the 1000 KBps rate, the cost to transfer the data is 3508 mAs for the State of the Art, while only 3234 mAs for Omni. This validates Omni's paradigm of splitting context and data and transmitting context in as low-energy fashion as possible.

Having demonstrated the efficacy of using Omni as the backbone of a high throughput application, we also show how it performs for a low-throughput application, PRoPHET. This experiment uses three devices labeled A, B and C. Device A is out of range of C, but intends to deliver a single 1 KB file to C. Device B encounters A, who shares the file with B for forwarding to Device C at some later interval (five seconds in our experiment). As Figure 7 shows, aside from the flexibility garnered in moving from the State of the Practice to the State of the Art, there is negligible improvement in energy and latency. This is because, in the absence of an integrated neighbor and service discovery approach (only available on Omni), data transfer over WiFi necessitates network discovery. Meanwhile, the vast majority of the latency when using Omni is inherent to the delayed nature of the application scenario (i.e., the five seconds it takes to encounter Device C). Moreover, the lack of need for periodic transmission of multicast packets substantially reduces the energy consumption for Omni. These benefits demonstrate the importance of marrying neighbor and service discovery and providing it as a separate mechanism to data transfer.

5 FUTURE WORK AND CONCLUSIONS

Omni substantially extends the capabilities of multi-networking middleware by recognizing that it is not sufficient simply to determine which among many technologies is best suited for transmitting a piece of data. One must also consider what technologies are best suited for discovery in the first place. Further, as our evaluation shows, recognizing that periodic context is fundamentally different than one-time data transfers is pivotal in enabling practical and efficient communication support for D2D applications.

In the future, sharing context (and data) with more than just one-hop neighbors could extend the range of a device's knowledge about the environment. BLE Mesh offers a promising solution for low-energy context sharing across longer ranges; future work will integrate BLE Mesh with Omni. In the same vein, the BLE beacons used in our current implementation are limited in size, but advancements such as Bluetooth 5 promise to support expanded beacon sizes. Larger beacons have the potential to enhance the richness of information in both service requests and advertisements, while still maintaining one of the key benefits of Omni: integration with technologies dedicated to low-level neighbor discovery.

In conclusion, as neighbor discovery becomes more sophisticated and the number of low-energy communication technologies grows, applications require programming frameworks like Omni. The novelty of Omni, in particular the distinction between lightweight context, which can be transmitted frequently and inexpensively, and heavier weight data will be integral in the development of practical IoT applications in the wild.

ACKNOWLEDGEMENTS

This work was funded in part by the NSF, Grant #CNS-0844850.

REFERENCES

[1] Ken Arnold, Robert Scheifler, Jim Waldo, Bryan O'Sullivan, and Ann Wollrath. 1999. *Jini specification.* Addison-Wesley Longman Publishing Co., Inc.

[2] Mehedi Bakht, Matt Trower, and Robin Hilary Kravets. 2012. Searchlight: Won't you be my neighbor?. In *Proceedings of the 18th annual international conference on Mobile computing and networking.* ACM, 185–196.

[3] O. Bello and S. Zeadally. 2014. Intelligent Device-to-Device Communication in the Internet of Things. *IEEE Systems Journal* (2014).

[4] Mauro Caporuscio, Pierre-Guillaume Raverdy, and Valerie Issarny. 2012. ubiSOAP: A service-oriented middleware for ubiquitous networking. *IEEE Transactions on Services Computing* 5, 1 (2012), 86–98.

[5] Xiaomeng Chen, Abhilash Jindal, Ning Ding, Yu Charlie Hu, Maruti Gupta, and Rath Vannithamby. 2015. Smartphone Background Activities in the Wild: Origin, Energy Drain, and Optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15).* 40–52.

[6] S. Cho and C. Julien. 2016. ChitChat: Navigating Tradeoffs in Device-to-Device Context Sharing. In *Proceedings of the International Conference on Pervasive Computing and Communications.*

[7] Yousri Daldoul, Djamal-Eddine Meddour, Toufik Ahmed, and Raouf Boutaba. 2016. Performance and scalability evaluation of IEEE 802.11 v/aa multicast transport. *Wireless Communications and Mobile Computing* 16, 14 (2016), 1987–2000.

[8] Prabal Dutta and David Culler. 2008. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM conference on Embedded network sensor systems.* ACM, 71–84.

[9] Erik Guttman. 1999. Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing* 3, 4 (1999), 71–80.

[10] Bo Han and Aravind Srinivasan. 2012. eDiscovery: Energy efficient device discovery for mobile opportunistic communications. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on.* IEEE, 1–10.

[11] Dan Harkins. 2008. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In *Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on.* IEEE, 839–844.

[12] Syed Rafiul Hussain, Shagufta Mehnaz, Shahriar Nirjon, and Elisa Bertino. 2017. Seamblue: Seamless bluetooth low energy connection migration for unmodified iot devices. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN).*

[13] Christine Julien, Chenguang Liu, Amy L Murphy, and Gian Pietro Picco. 2017. BLEnd: practical continuous neighbor discovery for Bluetooth low energy. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks.* ACM, 105–116.

[14] Arvind Kandhalu, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. 2010. U-Connect: A Low-Latency Energy-Efficient Asynchronous Neighbor Discovery Protocol. In *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks.* ACM, 350–361.

[15] P.H. Kindt, D. Yunge, G. Reinerth, and S. Chakraborty. 2017. Griassdi: Mutually assisted slotless neighbor discovery. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks.* ACM.

[16] Choonhwa Lee and Sumi Helal. 2002. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research* 11, 1 (2002), 1–12.

[17] Amit A Levy, James Hong, Laurynas Riliskis, Philip Levis, and Keith Winstein. 2016. Beetle: Flexible communication for bluetooth low energy. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services.* ACM, 111–122.

[18] A. Lindgren, A. Doria, and O. Schelen. 2003. Probabilistic Routing in Intermittently Connected Networks. *ACM SIGMOBILE Mobile Computing and Communications Review* 7, 3 (2003), 19–20.

[19] Toby Nixon, A Regnier, Vipul Modi, and Devon Kemp. 2009. Web Services Dynamic Discovery (WS-Discovery), version 1.1. *OASIS Standard Specification* (2009).

[20] A.K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. 2009. MobiClique: Middleware for Mobile Social Networking. In *Proceedings of the 2nd ACM Workshop on Online Social Networks.* 49–54.

[21] Ying Qiu, Shining Li, Xiangsen Xu, and Zhigang Li. 2016. Talk More Listen Less: Energy Efficient Neighbor Discovery in Wireless Sensor Networks. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications.* IEEE, 1–9.

[22] J. Scott, J. Crowcroft, P. Hui, and C. Diot. 2006. Haggle: A Networking Architecture Designed Around Mobile Users. In *Proceedings of the 3rd Annual Conference on Wireless On-Demand Network Systems and Services.* 78–86.

[23] Eugene Shih, Paramvir Bahl, and Michael J Sinclair. 2002. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th annual international conference on Mobile computing and networking.* ACM, 160–171.

[24] V. Srinivasan, T. Kalbarczyk, and C. Julien. 2015. Disseminate: A Demonstration of Device-to-Device Media Dissemination. In *Proceedings of the 2015 IEEE International Conference on Pervasive Computing and Communication (Workshops).* 196–198.

[25] Venkat Srinivasan, Tomasz Kalbarczyk, and Christine Julien. 2015. Disseminate: A demonstration of device-to-device media dissemination. In *Proceedings of the 2015 International Conference on Pervasive Computing and Communication Workshops.* IEEE, 196–198.

[26] J. Su, J. Scott, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton. 2007. Haggle: Seamless Networking for Mobile Applications. In *Proc. of Ubicomp.* 391–408.

[27] K. Wang, X. Mao, and Y. Liu. 2015. BlindDate: A Neighbor Discovery Protocol. *IEEE Transactions on Parallel and Distributed Systems* 26, 4 (2015), 949–959.

[28] L. Wei, B. Zhou, X. Ma, D. Chen, J. Zhang, J. Peng, Q. Luo, L. Sun, D. Li, and L. Chen. 2016. Lightning: A High-Efficient Neighbor Discovery Protocol for Low Duty Cycle WSNs. *IEEE Communications Letters* 20, 5 (2016), 966–969.

[29] Xinzhou Wu, Saurabha Tavildar, Sanjay Shakkottai, Tom Richardson, Junyi Li, Rajiv Laroia, and Aleksandar Jovicic. 2013. FlashLinQ: A synchronous distributed scheduler for peer-to-peer ad hoc networks. *IEEE/ACM Transactions on Networking (TON)* 21, 4 (2013), 1215–1228.

[30] Yibo Wu, Yi Wang, Wenjie Hu, Xiaomei Zhang, and Guohong Cao. 2016. Resource-aware photo crowdsourcing through disruption tolerant networks. In *Proveedings of the 36th IEEE International Conference on Distributed Computing Systems.* IEEE, 374–383.

[31] Yongping Xiong, Ruogu Zhou, Minming Li, Guoliang Xing, Limin Sun, and Jian Ma. 2014. ZiFi: Exploiting cross-technology interference signatures for wireless LAN discovery. *IEEE Transactions on Mobile Computing* 13, 11 (2014), 2675–2688.