Title

Middleware for IoT-Cloud Integration Across Application Domains

Permalink
https://escholarship.org/uc/item/6j09m75b

Journal
IEEE Design and Test, 31(3)

ISSN
2168-2356

Authors
Huo, Chengjia
Chien, Ting-Chou
Chou, Pai H

Publication Date
2014-06-01

DOI
10.1109/mdat.2014.2314602

Peer reviewed

Middleware for IoT-Cloud Integration across Application Domains

Chengjia Huo, Ting-Chou Chien, and Pai H. Chou, *Member, IEEE*

***Abstract*—Profile-based protocols such as Bluetooth 4.0 Low Energy (BLE) Technology have enabled very low-power devices to efficiently participate across multiple application domains in the Internet of Things (IoT). We propose a middleware layer called rimware to enable today's profile-based IoT nodes to realize the full potential of inter-application participation. First, the nodes need to be able to establish authenticated, secure connections to the cloud through trusted gateways using an adapter structure when the smartphone or tablet is not available. Second, a knowledge base in the cloud is needed to establish mapping between profiles on the device side and application semantics on the cloud side. Results show our rimware to provide a modular, extensible structure for integration across three applications while incurring minimal code size and communication overhead on BLE devices.**

***Index Terms*—Middleware, Rimware, Bluetooth Smart, Internet of Things, Cloud Computing, Low Power, Management, Wireless Sensor Networks, Sensor Modeling, Security.**

I. Introduction

The Internet of Things (IoT) has been receiving growing attention in recent years as the next wave of computing revolution made possible by low-cost, miniature low-power systems-on-chip (SoC) with computing and communication capabilities. The term IoT could take on the narrower meaning of an IP (Internet Protocol) network of relatively simple devices; however, the broader sense of IoT refers to devices that *interact across application domains*. In contrast, wireless sensor networks (WSN) are designed for a single purpose. What makes it possible is the use of *profiles* in modern communication protocols.

A. Profile-Based Protocols for Cross-Application Devices

A profile-based protocol is one whose message formats are grouped by the type of service. For example, the human interface device (HID) profile in Bluetooth is for mouse, keyboard, or a joystick; the advanced metering infrastructure (AMI) profile in ZigBee is for on-demand or periodic reading of a smart meter, real-time pricing, and for temperature alerts. A profile enables cross-vendor interoperability, and more importantly, a device may also implement multiple profiles. BLE devices can dynamically discover each other's capabilities in terms of what profiles they implement. This is what allows a BLE device to participate in multiple applications efficiently without having to know in advance the higher-level purpose of the application. Our proposed work will take care of the mapping between the profiles and the application semantics.

B. Connecting BLE and Cloud Computing via Smartmobiles

Bluetooth 4.0 Low Energy Technology (BLE) has gained wide popularity in recent years as the fastest growing wireless protocol for IoT, thanks to the very low power consumption and direct compatibility with *smartmobiles* (i.e., smartphones or tablets) without using a dongle or gateway. Its low average-power consumption enables a BLE device to last for one year on a CR2032 coin-cell battery. Today, smartmobiles serve as the gateway between BLE devices and the corresponding cloud backends to form a powerful combination.

Let us examine three seemingly distinct applications. First, a *proximity tag* (PT) is a BLE device adhered on everyday objects such as key chains, purses, wallets, pets, remote controls, and other commonly misplaced items. They are paired with the owner's smartphone, which logs time and location stamps. To find a lost item, the smartphone scans the item's tag within its RF range or inspect its log for the time and location last seen. A cloud backend enables different users to help each other find lost items. Second, a *heart rate monitor* (HRM) for sports and fitness also pairs with the owner's smartphone to upload the user's activity data to a cloud that then disseminates the user's achievements on social networks. The third is *smart lighting control* (SL), where light switches are equipped with BLE devices for wireless control from not only smartphones but also occupancy sensors, other macro-buttons, or timers. The devices may connect to the cloud via a smartphone or a set-top box acting as a gateway.

Profiles enable BLE devices to cross application boundaries at both device level and cloud level. For example, an HRM, a smartphone, and a smart watch are all capable of serving as PTs if they implement the PT profile. A BLE light switch should also be able to help finding lost items by acting as a beacon in a PT profile, i.e., by performing scanning, logging, and uploading PT IDs to the cloud. An HRM, smartphone, or a PT should also be able to participate in the SL application by providing user identity upon handling each event for personalized response.

C. Need for Middleware for IoT-Cloud Integration

While most clouds for IoT provide support for data store and web API, two major features are still needed for inter-application integration. First, the devices need support to be able to establish secure, authenticated, access-controlled connections to the cloud through alternative gateways when

The authors are with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92617, USA. email: phchou@uci.edu.

Manuscript received August 31, 2013; revised December 27, 2013.

the primary one (i.e., smartphone or set-top box) becomes unavailable. Second, the cloud side needs to be able to model each device's capabilities by translating between the *BLE profile hierarchy* of a device and the *web API* that it provides. To accomplish this, we propose a new kind of middleware, called *rimware*, that spans the cloud and gateways to collaboratively provide these two features. The term rimware is coined from the analogy that the proposed middleware resides in a rim that wraps the cloud (i.e., cyber space) on the inside and *networks of things* (NoTs, in physical space) on the outside. This paper describes an implementation of rimware called BlueRim to exploit the specific properties of BLE in achieving cross-application interoperability of IoT devices.

II. Background and Related Work

Cross-application interoperability of IoT devices can be supported at both device level and cloud level. This section provides a background and related work on approaches to date.

A. Device Description for Interoperability

Interoperability across applications at the device level requires devices to be able to identify or describe their own capabilities at the time they are discovered by another device. Two ways of achieving this are markup languages and profiles.

1) Markup-Language Descriptions: A general, web-oriented way to describe the device explicitly is to use a markup language such as SensorML by OGC's Sensor Web Enablement (SWE) initiative [1]. It makes sensor devices discoverable and accessible over the Internet through web service interfaces by defining the discovery process and describing the sensor's capabilities in XML and wrapped as Sensor Observation Service (SOS), the web service standard from OGC for publishing on web. In practice, however, devices need assistance from a translator or an agent such as SOS [2], SLS [3], or SSA [4] before registering as a service due to the use of vendor-specific interfaces and resource constraints. While suitable for connecting sensor networks to the web, it is too heavyweight for M2M devices. For example, such a PT may not be able to directly interact with an SL switch unless a gateway and the respective cloud backends are involved.

2) Profile-Based Protocols: Most low-power RF protocols for IoT (including Bluetooth, BLE, ZigBee, Z-Wave, ANT+) use profiles for devices to discover and match their capabilities. In BLE, a profile consists of a set of *characteristics*, each of which contains a *value* and metadata known as *descriptors* (e.g., value range, unit of measure, or human-readable text). Profiles are often standardized for well-established application classes, such as lighting control, HRM, HID, etc. Turning on or off a light switch entails changing the corresponding characteristic value. One profile implemented by all BLE devices is GATT, or *Generic Attribute* Profile, which allows devices to discover each other's supported profiles with only a few bytes of data exchange, without having to involve the gateway or the cloud. Because profile access can be done locally and in a very energy-efficient way, we believe it matches the widest range of IoT applications.

TABLE I: Device Capability Description.

Approach	Work	Described by	Device Types
SWE	Mitton [2]	SOS Agent	all
	PULSENet [3]	SLS	all
	SensorSA [4]	SSA	all
Profile	WuKong [5]	device self	Wu-device
	rimware (ours)	device self	any BLE

TABLE II: Classification of IoT-Cloud Integration.

Work	Connection	Cloud Domains
Kurschl [7]	via gateway	single
Rajesh [8]	self-enabled	single
Hassan [9]	via gateway	single
Mitton [2]	via gateway	mult. (archived data)
rimware	g.w. or smartphone	mult. (data + control)

WuKong [5] defines profiles that are independent of the transport protocol such as ZigBee, Z-Wave, or Wi-Fi. Profile enables WuKong-compliant devices (called Wu-devices) to interoperate and task mapping from a *flow-based* program. In case of a device crash, the mapper can re-task it to substitute devices (i.e., with matching profile). However, certain types of M2M interaction that require physical-layer compatibility (e.g., PT) may be more difficult to implement. Table I summarizes the relevant approaches described above.

B. IoT-Cloud Integration

IoT requires devices to be accessible and manageable over the Internet, which is the de facto way to connect everything with worldwide accessibility, massive storage and powerful computation capability. AutoHome [6] uses service-oriented approaches to make devices accessible as web services, though its use of cloud is limited. Cloud technology is used by [7]–[9] for establishing a self-contained sensor network with a single cloud domain. Multiple cloud domains can be supported by [2] but for sharing archived data among different clouds rather than at device level. Table II summarizes the different approaches for IoT-Cloud integration.

III. Overview

We envision an enhanced architecture for IoT by introducing our proposed rimware, a middleware layer that spans the cloud and the gateway. We assume that a cloud for a given application already provides web services for data, configuration, and commands associated with devices belonging to authenticated users via *gateway processes*. Today, that gateway process is predominantly an app running on a smartphone that logs in to the user's account on one side and connects to BLE devices on the other side. However, this structure limits cross-application and cross-user interaction. To overcome this problem, we propose a plug-in structure as part of our rimware on the gateway device so that it can take over the gateway processes that run as one or more smartphone apps today with security features. Another new proposed function is rimware on the cloud side to capture the device's capabilities by mapping between the device profiles and web API. This covers not only access-controlled reading and writing of data and state on the

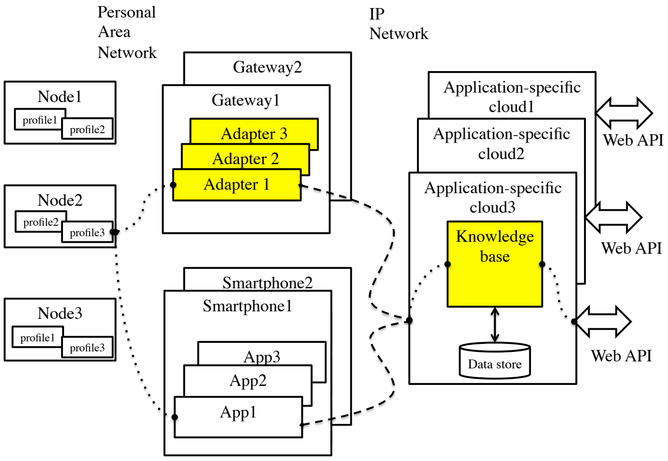


Fig. 1: Overview of rimware

device but also notification of events. This section presents an overview of the rimware structure to support these two main features.

A. Overview of Proposed IoT-Gateway-Cloud System

An overview of an IoT-gateway-cloud system running rimware is shown in Fig. 1. Our rimware has components that run on both the gateway and the cloud for each application domain, as shown in the highlighted boxes. The cloud side is usually assumed to be a cloud-based *Software as a Service* (SaaS) application, and our cloud-side rimware components are deployed on a *Platform as a Service* (PaaS) such as OpenShift. The cloud is assumed to provide a REST interface to the users for accessing uploaded data and controlling or configuring devices. Web services of service and characteristic profiles are described in JSON-WSP and provided in JSON format and exposed as REST interfaces to the users. The next two subsections briefly describe the standard cloud features and our rimware layer.

B. Core Components of Cloud

We assume that the core functionality of the application-specific cloud is implemented by several components. They can be called the *state monitor* and *data store*.

The *state monitor* consists of a monitoring process, a database table for storing gateway (which could be a smartphone app) information, and a table for tracking the set of connected devices. It may monitor the state of registered NoTs by periodically sending requests to all gateways to track connectivity inside each NoT, or the NoT may actively push updates to it. In either case, it maintains the status of the system in terms of the gateways and joined devices.

The *data store* is cloud-based storage for all sorts of user data and settings, such as heart rate data with time stamps and other geo-location data. It is usually implemented with a database engine. It may be invoked via a web API, by the state monitor, or by our rimware components.

C. Rimware Components

Rimware components are (1) *adapters* running on the gateway, (2) knowledge base running in the cloud, and (3) access controller, also running in the cloud.

1) Adapters on Gateway: An *adapter* running on the gateways acts as the interfacing process between a device and the cloud. In simple terms, each adapter can substitute the role of an app running on a user's smartphone. The gateway instantiates an adapter for a connected device to make use of its profiles on one hand and to interact with the cloud side on the other. A gateway can run multiple adapter instances that correspond to different users and different applications. Section IV describes this in more detail.

2) Knowledge Base in the Cloud: The *knowledge base* is what maps between BLE profiles and actions in the cloud, including web API and access to the data store. It imposes constraints on the BLE profile hierarchy [10] to enable mapping with the cloud primitives. The constraints define how a firmware developer uses the BLE profile hierarchy to describe the required security enforcement policy, the device's capabilities, and storing the authentication token. Section V-A discusses the profile hierarchy and mapping by the knowledge base in more detail.

3) Access Controller in the Cloud: The *access controller* also runs in the cloud alongside the knowledge base to enforce access control. It handles *application-level* (rather than device-level) authentication and provides a token mechanism as credentials for devices and the cloud sides to talk to each other in an access-controlled way. Section V-C1 describes this mechanism. Note that in this paper, we assume the gateway is trusted. Conceptually, the gateway is part of the rimware and does not belong to any specific application domain.

IV. Gateway

The gateway system serves the purpose of bridging the BLE devices with the cloud when the smartphone is unavailable (e.g., turned off or goes out of range). This means it must have connectivity with the BLE devices downstream, connectivity with the cloud(s) upstream, and it runs *adapter* processes in a plug-in-style architecture to bridge the two sides while enforcing *application-level security* policies over the existing communication channels.

A. Adapter Instantiation

The gateway proactively discovers and connects nearby BLE devices that are looking for connectivity by *advertising* (in BLE sense). The gateway is set up with the same *device-level* authentication as a smartphone would by pairing if necessary. For every connected device, the gateway instantiates an *adapter* on the gateway and uses the adapter to operate on the device's BLE profiles. From the device's BLE profile defined as part of our rimware, the adapter process discovers the required security policy and applies it on the communication channel (e.g., SSH) accordingly. It accomplishes this by looking for a pre-defined service profile which stores information about the security enforcement approach. If such a service profile exists, then the adapter initializes a security

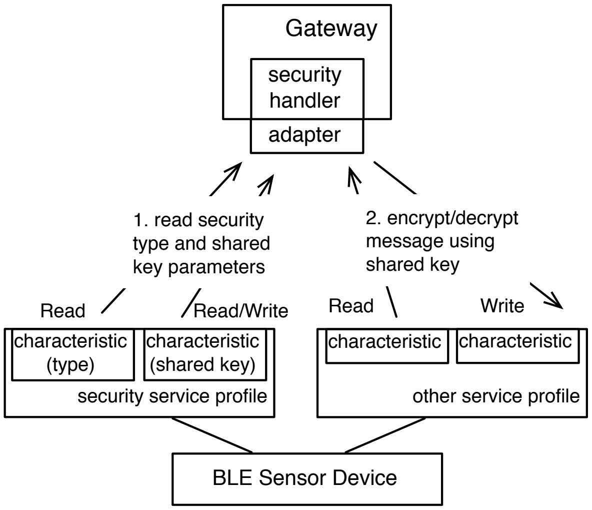


Fig. 2: Example of passing security approach information using BLE profile

handler based on characteristics under the profile for the security type.

B. Authentication

Application-level authentication is initiated by devices against the cloud. As the gateway acts as the conduit, the purpose of authentication is to check whether the gateway is a trusted one. In contrast to smartphone apps that authenticate once and all of its paired devices are *transitively* authenticated, the gateway is not responsible for user authentication. This requires the device and cloud to share a secret, called the authentication token. Details of how the cloud and device sides authenticate are found in Section V-C1.

C. Gateway-to-Cloud Interaction

The gateway also maintains an Internet connection with the knowledge base and access controller in the cloud. An adapter actually does not have much knowledge about the application; instead, it it passes the device's BLE profile hierarchy information to the knowledge base on the cloud side, which also sends messages downstream through the adapter to the BLE devices using BLE profiles. Also, the gateway does not maintain the security handlers (Section IV-A); instead, it downloads and updates the security handlers for different types of security approaches from the cloud side. The advantage is that the adapters can be kept very lightweight, and that is exactly the way middleware components should be.

For example, as shown in Fig. 2, the gateway finds in the security service profile that the device requires a symmetric-key encryption for exchanging messages. Therefore, the gateway holds a shared key either by reading the shared key from the device's characteristic or passing a self-generated shared key to it and uses the key to encrypt/decrypt messages against other service profiles of the device.

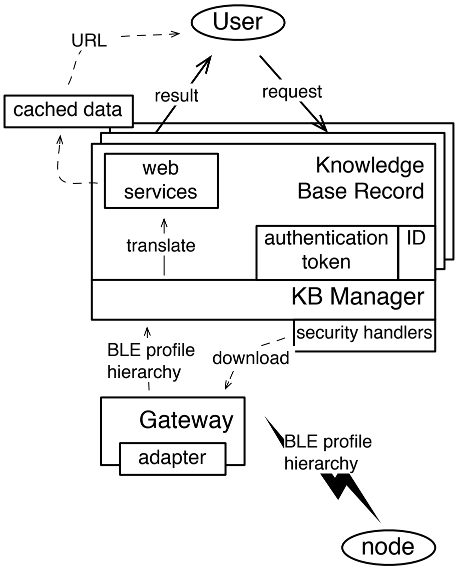


Fig. 3: Structure of Knowledge Base

V. Knowledge Base and Access Controller

Our middleware on the cloud side is composed of components called the *knowledge base* and *access controller* that work with the existing core components called the data store and state monitor. This section first explains how the knowledge base (Fig. 3) performs mapping among the web API, the BLE profiles, data store, and state monitor. Then, we also show how the access controller interacts with the gateway and knowledge base to enforce security policies.

A. BLE Profile-based Capability Description

The knowledge base captures the device capabilities in terms of the BLE profile hierarchy. A *characteristic profile*, which corresponds to a single function on the device, is the basic primitive of a device capability. A characteristic profile contains a set of *descriptors*, or metadata, that add meaning to the characteristic values, such as the unit of temperature, the minimum or maximum value, integer or floating point, etc. A device's capabilities are therefore represented by a set of characteristic profiles. Each *service profile* contains a group of characteristic profiles and therefore represents the functionality of a component such as a sensor or an actuator. Characteristic profiles can be classified into status/data, setting/action, and event characteristic, according to their accessibilities. Our middleware also defines service profiles for configuration.

1) Status/Data Characteristic: A status/data characteristic has *read* accessibility and provides the status of any sensor on the device or the generated data by the sensor. An example is one that provides raw readings from a sensor. An operation on this type of characteristic can be done in a single transaction.

2) Setting/Action Characteristic: A setting/action characteristic has *write* accessibility and receives setting parameters or actions commands. An example is one that receives an on/off signal on an SL device. An operation on this type of characteristic can also be done in a single transaction.

3) Event Characteristic: An event characteristic has *indication* or *notification* accessibility. This means the device pushes a packet upstream when it detects an event. For example, a blood pressure monitor that takes measurements regularly can generate a new data packet to notify the host through an event characteristic. This type of characteristic is not a transactional operation and needs the host to subscribe on the characteristic to be notified when any event is triggered.

4) Rimware-Defined Profiles for Configuration: In addition to capturing device capabilities using BLE profile hierarchy, rimware also defines additional *service profiles* for configuration purposes. They cover metadata such as (non-functional) device information [11], security policy such as crypto keys, and authentication information including identity and credentials.

B. KB Manager for Profile-to-Web-API Mapping

A characteristic may belong to more than one category as BLE profile definition [10], allowing a characteristic to have different accessibilities. The *KB Manager* wraps each characteristic profile from the device's BLE profile hierarchy as a web service by specifying the characteristic's UUID, category and accessibility, and value content. It also attaches the content in the descriptors of the characteristic as the description of the web service.

When a user sends a request against any web service, the KB Manager translates the request into either a datastore operation or a BLE operation and passes the operation messages to the device through its adapter on the gateway. For an event characteristic, due to its specialty on operation, the KB Manager caches the received indication/notification data and returns a URL to the web service request by which the data can be accessed. The KB Manager further groups characteristic web services that belong to the same service profile and provides them as a combined web service for each service profile. The KB Manager maintains these web services in the device's Knowledge Base record and exposes the web service to authorized users to access as well as to further compose the web services to represent complex functions that the device does not or cannot provide on its own. In our implementation, BlueRim, web services of service and characteristic profiles are described in JSON-WSP and wrapped as JSON-based REST interfaces to the users.

C. Access Control

Our proposed rimware supports access control to the devices. Access control is built on top of authentication, which is initiated from the device side. Rimware uses BLE profiles extensively as a fundamental mechanism for access control.

1) Device-Initiated Authentication: The knowledge base and the access controller work jointly to support authentication, as initiated from the device side. For every device that joins rimware via a gateway, the knowledge base maintains a unique record for the device indexed by its identity. The record contains the device's capability descriptions and authentication token. The knowledge base is not responsible for storing user account information, since it is handled by the cloud, rather than the middleware.

2) Authentication Token: When a device joins rimware for the first time, the KB Manager asks the gateway to access the device's identity from the Device Information service profile (Section V-A4) and initializes a new KB record for this device if one does not exist. The authentication token is stored in the device's record in the knowledge base and also written into device's authentication characteristic profile through the gateway. The device then uses the token for all other requests from rimware.

The authentication token is used by the device to check if the connected gateway is a trusted one. If the device needs to disconnect from the gateway and later connects back to it or another gateway, the device will ask the party on the other side of the new connection to write a token to the authentication characteristic profile. The new token must match with the one stored on the device to ensure the new connection is established to an authorized party before the party is allowed to access the other (i.e., other than Device Information) service profiles. This simple authentication mechanism prevents the information including profile's hierarchy, and the value of characteristics from being accessed by a third party instead of a gateway process from rimware for some special cases such as roaming.

3) Access Control by Profile Constraining: For a device to support access control with rimware, the firmware developer needs to follow some constraints when designing the BLE profiles. First, the authentication service and characteristic profiles can only use UUIDs that are defined by rimware for authentication service. Second, an authentication token can be initialized only when there is no token stored on the device, such as when the device join rimware for the first time, or if the token stored in RAM is lost due to power outage. Third, before authentication checking is approved, the firmware should disallow any attempt to access service profiles other than Device Information, Security Enforcement, and Authentication. These are relatively simple rules for firmware designers to follow, and the resulting firmware will have the necessary mechanisms for supporting access control policies defined on the cloud side.

VI. Case Study

We present an end-to-end case study of inter-application interactions of BLE devices using our proposed rimware. This is a new, powerful feature that has not been available previously, as integration have been limited to mashup services at the cloud level. Our case study covers the interactions across the three applications from the introduction: proximity tags (PT), heart-rate monitors (HRM), and smart lighting (SL).

A. Smartphone-based Setup

Initially, the devices for all three applications are set up separately using their respective smartphone apps. The user would open these apps login to the respective cloud accounts for each application. The same smartphone is paired (or otherwise associated without pairing) with the BLE devices using device-level authentication such as the six-digit passcode in BLE. Different apps are granted access to talk to these devices that support the matching profiles. The data that they collect are sent through the smartphone app and sent upstream to the cloud. This is the way BLE-based IoT devices connect to the cloud.

B. Gateway Setup and Adapter Instantiation

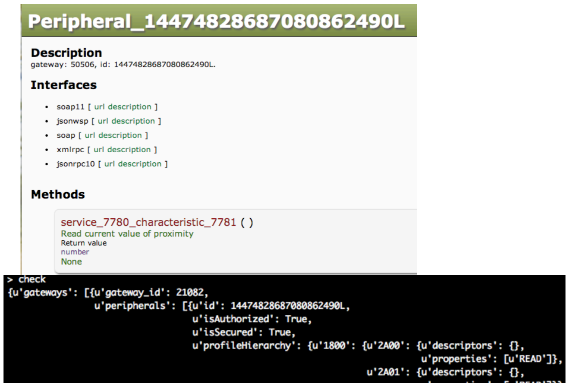
The smartphone setup can be transferred over to a trusted gateway. The gateway is given the credentials for device-level authentication (i.e., pairing) so that it can proactively look for devices to connect when they come in the RF range unpaired. Note that not all IoT devices are required to connect to the cloud at all times. For example, a PT may be attached on a wallet, a remote control, or a pet, while an HRM is usually worn under the shirt on a person, and they can travel in and out of the RF range of the smartphone or the gateway. On the other hand, an SL node is usually stationary. In any case, upon connecting the device, the gateway instantiates an adapter to connect the device to the cloud, and then the device initiates authentication with the cloud.

C. Device to Multiple Clouds

Our rimware supports a device's participation in multiple applications. For example, an SL switch can participate in both the SL and PT applications by implementing the required BLE profiles. The gateway pairs with the SL node just once but can support both SL and PT applications by instantiating separate adapters that share access to the same device. Similarly, the HRM device can implement both the HRM and PT profiles. The device initiates authentication by asking the adapter to obtain and provide an authentication token to the device for each application to see if the gateway is genuine. If so, it can continue.

As a light switch, the SL device can interact locally with other BLE devices such as occupancy sensor, macro buttons, and other smartphones and tablets, either directly with each other or indirectly through relay nodes or the gateway. The SL device may be programmed to log data to the cloud and accepting control commands downstream via the gateway from the cloud, all using the SL profile. At the same time, access control is enforced such that a neighbor's device within the RF range of the gateway will not have the proper credentials to authenticate successfully, even if it would allow the gateway to pair with it without a passcode.

As a beacon for PTs, the SL switch can discover a PT in proximity and send the PT's ID and timestamp upstream through its PT-cloud adapter to the user's PT cloud account. It can also interact with smartphones over the PT profile for indoor localization but without going through the gateway. Moreover, the SL switch can actually use the PT profile for context-aware actions without relying on the PT application. For example, a *macro button* can turn on or off a different set of lights based on the owner of the closest BLE tag (i.e., a device that implements PT profile). Because the HRM can also participate in both the HRM and the PT, the macro button can detect the presence of the HRM, look up the user associated with the HRM (as permitted by the user explicitly), and maps it back to the user in the SL space. Then, this enables the macro button to look up the user's device list for turning on or off and sends the commands to them accordingly.



Peripheral_14474828687080862490L

Description
gateway: 50506, id: 14474828687080862490L.

Interfaces

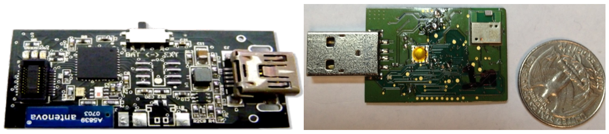
- soap11 [url description]
- jsonwsp [url description]
- soap [url description]
- xmlrpc [url description]
- jsonrpc10 [url description]

Methods

service_7780_characteristic_7781 ( )
Read current value of proximity
Return value
number
None

```
> check
{u'gateways': [{u'gateway_id': 21082,
                u'peripherals': [{u'id': 14474828687080862490L,
                                  u'isAuthorized': True,
                                  u'isSecured': True,
                                  u'profileHierarchy': {u'1800': {u'2A00': {u'descriptors': {},
                                                                            u'properties': [u'READ']},
                                                                  u'2A01': {u'descriptors': {},
```

Fig. 4: UI for Web Services (white) and BlueRim Shell Interface (black)



(a) (b)

Fig. 5: (a) EcoBT Platform for proximity tag and smart lighting, and (b) BLE Heart-rate Monitor

D. Implementation

Our rimware, called BlueRim, is implemented in Python and deployed on a private PaaS, though the concepts are applicable to public cloud technologies. As shown in Fig. 4, a user can discover a device's capability from its BLE profile hierarchy by querying it using the interactive shell interface. In addition, sensor's capability is also exposed as web services through JSON-WSP.

The firmware has been implemented on EcoBT board [12] for all three applications (PT, HRM, and SL), as shown in Fig. 5. We also implemented virtual nodes by emulation on BLE-enabled computers for better observability and controllability, but its behavior is identical to that on the board. It uses the BLE profile hierarchy to declare its requirement on security and authentication enforcement. In the evaluation, we use simple symmetric encryption based on AES in mode CFB for securing the communication channel and use the authentication token-matching mechanism for authentication checking against gateway. The code size of each component is given in Table III. Given that the BLE microcontroller (CC2540) has 256 KB of code flash and 8 KB of SRAM, the memory overhead is marginal. Table IV shows the time on handling initialization of both security and authentication is 135 ms each, so the impact on the overall performance is negligible.

VII. CONCLUSION

We have introduced the concept of rimware for the integration of IoT and the cloud to form a very powerful cyber-

TABLE III: Code Sizes of Firmware and BlueRim (Gateway and Cloud) Components

Component	Functionality	Bytes
Node	Security Service Profile	5406
Node	Authentication Service Profile	1596
Node	Other Profiles	58038
Gateway	Adapter	17718
Gateway	Overall	62053
Core	Security Plugin	2135
Core	Authentication Plugin	992
Core	Overall	65011

TABLE IV: Initialization Time Evaluation

Service	Initialization Time (ms)
Security Service	135
Authentication Service	135

physical system. Our specific implementation, called BlueRim, can scale from isolated networks of things (NoT) to the global IoT. Beyond merely scalability, which has been a goal of wireless sensor networks, we believe that the true power of IoT is in the ability for the devices to across application boundaries. Profile-based protocols are primary candidates for M2M interactions, and BLE has the additional advantage of the very long battery life. To unleash this potential, our rimware provides a middleware layer that enables gateway devices to establish connections between the devices and the cloud with roaming support while requiring them to prove their trustworthiness. The cloud side of the middleware is structured to establish modular mapping among device-level, cloud-level operations, and web API. The effectiveness of these fundamental features have been validated in several real-world applications with different access patterns while retaining their ability to consume very low power. We believe that our approach represents an important technology in taking IoT closer to realizing the full potentials.

References

[1] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC® Sensor Web Enablement: Overview and High Level Architecture," *GeoSensor Networks*, pp. 175–190, 2008. [Online]. Available: http://www.opengeospatial.org/ogc/markets-technologies/swe

[2] N. Mitton, S. Papavassiliou, A. Puliafito, and K. S. Trivedi, "Combining Cloud and sensors in a smart city environment," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, pp. 1–10, 2012.

[3] S. M. Fairgrieve, J. A. Makuch, and S. R. Falke, "PULSENet$^{\text{TM}}$: an implementation of sensor web standards," *International Symposium on Collaborative Technologies and Systems, 2009 CTS'09*, pp. 64–75, 2009.

[4] T. Bleier, B. Bozic, R. Bumerl-Lexa, A. da Costa, and e. a. Costes, S, "SANY: an open service architecture for sensor networks," *The SANY Consortium*, 2009.

[5] N. Reijers, K.-J. Lin, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu, "Design of an intelligent middleware for flexible sensor configuration in M2M systems." SENSORNETS, 2013.

[6] J. Bourcier, A. Diaconescu, P. Lalanda, and J. A. McCann, "Autohome: An autonomic management framework for pervasive home applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 6, no. 1, p. 8, 2011.

[7] W. Kurschl and W. Beer, "Combining cloud computing and wireless sensor networks," in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, ser. iiWAS '09. New York, NY, USA: ACM, 2009, pp. 512–518. [Online]. Available: http://doi.acm.org/10.1145/1806338.1806435

[8] V. Rajesh, J. M. Gnanasekar, R. S. Ponmagal, and P. Anbalagan, "Integration of Wireless Sensor Network with Cloud," in *Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on*, 2010, pp. 321–323.

[9] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, ser. ICUIMC '09. New York, NY, USA: ACM, 2009, pp. 618–626. [Online]. Available: http://doi.acm.org/10.1145/1516241.1516350

[10] S. Bluetooth, "Bluetooth: Bluetooth Core Specification v4.1," 3 December 2013. [Online]. Available: https://www.bluetooth.org/Technical/Specifications/adopted.htm

[11] [Online]. Available: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

[12] T. K. Lai, A. Wang, C.-M. Chang, H.-M. Tseng, K. Huang, J.-P. Li, W.-C. Shih, and P. H. Chou, "Demonstration Abstract: An 8 × 8 mm^2 Bluetooth Low Energy Motion-Sensing Wireless Sensor Platform," in *The 12th ACM/IEEE Conference on Information Processing in Sensor Networks, Demo Session, Berlin, April 2014*.

PLACE
PHOTO
HERE

Chengjia Huo received the A.B. degree in Computer Science and Software Engineering from the Dalian University of Technology, China, in 2006, and the M.S degree in computer science and engineering from University of California, Irvine, in 2009.

He is now pursuing the Ph.D degree at University of California, Irvine, under the supervision of Professor Pai H. Chou. His research interests include wireless sensor network, software modeling, OLAP.

Ting-Chou Chien received a Bachelors Degree in Computer Science and Information Engineering in 2004 from National Chiao Tung University, and M.S. degree in Electrical Engineering and Computer Science from University of California, Irvine, in 2009.

He is pursuing the PhD Degree in Electrical Engineering and Computer Science at University of California, Irvine under the supervision of Professor Pai H. Chou. His research interests include wireless sensor network, distributed system.

Pai H. Chou (M'98) received the A.B. degree in computer science from the University of California, Berkeley, in 1990, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, in 1993 and 1998, respectively.

He is a Professor in the Department of Electrical Engineering and Computer Science, University of California, Irvine. His research interests include wireless sensing systems, low-power design, energy harvesting, and system synthesis..

Dr. Chou is a recipient of the National Science Foundation CAREER Award.