Bluetooth Low Energy for the last 30 Meters of the Internet of Things

Dr. Cuno Pfister
Oberon microsystems, Inc.
Zürich, Switzerland
pfister@oberon.ch

***Abstract*—Bluetooth Low Energy is becoming increasingly important as a technology for the last 30 meters of the Internet of Things. Its GATT protocol can be mapped generically to HTTP, so that the Web can be expanded into a Web of Things reaching down even to low-cost wireless sensors and actuators. The mapping is done in gateways that implement the standardized GATT REST API.**

Keywords—Bluetooth Low Energy; Bluetooth Smart; Internet of Things; Web of Things; GATT; REST API; Gateway; Beacon

I. Bluetooth Low Energy

Bluetooth Low Energy [1], also known as *BLE* or by its marketing name *Bluetooth Smart*, has originally been developed at a Nokia Research Center as a short-range, low power wireless technology under the name Wibree. Nokia eventually passed control over BLE to the Bluetooth SIG [2]. It became an official part of Bluetooth 4.0 in 2010.

Like Classic Bluetooth, BLE operates in the 2.4 GHz spectrum, and uses adaptive frequency hopping for robustness against interferences. Unlike Classic Bluetooth, it uses fewer but broader channels, a different modulation scheme optimized for low power usage, and has a faster connection mechanism. The latter uses three reserved advertisement channels that were carefully chosen to minimize interference with the WiFi bands.

While BLE supports up to 260 kpbs data rate, it is typically used for occasional updates of sensor values, e.g. a temperature value, or for remotely controlling actuators, e.g. a light bulb's on/off state. The main goal is to reduce power consumption so that certain sensors may operate for years on a coin cell battery. The range in practice is roughly 30 meters, strongly depending on obstructions and the used antennas: with free line-of-sight, the range may be larger; in densely packed rooms it may be less.

II. BLE and Apps

Today, BLE sensors and actuators are typically controlled from apps running on mobile devices – smartphones and tablets. Such BLE devices are sometimes called *appcessories* (from *app* and *accessories*). Here is an example of a heart rate monitor that supports BLE (Figure 1, [3]):

Figure 1 - Example of a BLE heart rate sensor

For wearables – e.g. pulse meters, fitness arm bands, smart watches – BLE clearly dominates as the wireless technology of choice. This is due to the low power consumption of BLE chips, their small sizes and low cost. However, at least as important is the fact that by now, every modern smartphone supports BLE in addition to Classic Bluetooth (*Bluetooth Smart Ready*). The cost of adding BLE to a product that already supports Classic Bluetooth is negligible. Apple has started adding BLE support to all its devices starting with the iPhone 4S, including a flexible API for app developers. Meanwhile, BLE support has been added also to Android and Windows Phone. Besides mobile phone networks and WiFi, BLE has thus become the only cross-platform wireless technology from an app developer's perspective.

Mobile devices have thus become the freight train that is pulling along BLE on an impressive growth path. The Bluetooth SIG has extensive market data, but you can

check Google Trends for "Bluetooth Smart" to get an impression yourself. Another indication was the recent first edition of the *Bluetooth Europe* conference in Amsterdam this year: although it was an event for all Bluetooth topics, and Classic Bluetooth still has year-over-year growth of 20%, BLE was the all-dominating topic in Amsterdam.

III. Apps as Temporary Internet Gateways

Many BLE apps connect to the Internet sporadically. For example, a plant sensor (Figure 2, [4]) may sample temperature and humidity in regular intervals, and store them temporarily for up to a few months. The plant sensor app may try once per day to connect to the registered plant sensors, fetch all stored samples, and push them to a gardening web service hosted by the company that produces the plant sensors. The user is then notified by the web service, for example when it detects that some plant needs more (or less) water.

Figure 2 - Example of a BLE plant sensor

Firmware updates are another scenario where the mobile app acts as a temporary Internet gateway: the app regularly checks the gardening web service whether new firmware for the sensors has become available. When this is the case, it downloads it to the mobile device, and then performs an over-the-air update to the registered BLE sensor(s).

The gateway functionality of most BLE apps that connect to the Internet is highly application-specific: the data that is transmitted; the format of the data; whether data is pushed or pulled; whether app or cloud initiate communication; what kind of events or timing schedules trigger communication; whether/how a user is involved; what types of BLE sensors are supported; etc.

However, regardless what the exact gateway functionality of an app is, it allows connecting the Internet to the physical world, to "Things". Often, these things are simply considered to be the sensors and actuators, such as the plant sensor in the previous example. However, the more relevant things are those actually measured by a sensor, or manipulated by an actuator. For example, the interesting thing that a plant sensor makes accessible to the Internet is the stress level of the monitored plant. This thing is useful, because it is actionable: if a plant's stress level is high, and this is due to too much humidity at the roots, then a user can be notified with the recommendation to drain the plant's pot.

IV. Limitations of Temporary Internet Gateways

Wearable BLE devices move along with their users, and therefore along with their users' mobile devices. Thus they typically remain in range with BLE apps running on those mobile devices. Whenever such a BLE device needs access to the Internet, it can use a suitable smartphone app as a temporary gateway.

For non-wearable BLE devices, the opportunities for using a mobile device as a gateway fully depend on the user's behavior, and are thus far more limited and far less predictable. For example, if a machine in a factory is equipped with a BLE module, it can only connect to the Internet if and when a field technician with a suitable smartphone app is nearby. Depending on the use case, e.g. reconfiguration of the machine under human supervision, this may be exactly the right thing. Or it may be too limiting, for example in the following cases:

- for sending an alarm when a gas leak is detected in a plant;
- for periodically sending environmental data about a glacier that is barely accessible to humans, and where the system must work under harsh temperature conditions;
- for health monitoring applications where continuous monitoring is important, without having to rely on users not forgetting to charge their smartphones;
- for the around-the-clock logging of assets equipped with mobile beacons[1] that enter or leave a warehouse.

Generally speaking: whenever continuous and reliable connectivity is desired, when no natural and close association between sensor and human user exists, or when the environment is unsuitable for humans (and their consumer-grade electronics), then temporary Internet gateways can become too limiting.

[1] BLE beacons broadcast a beacon ID, which can be used for indoor positioning, location-specific advertising, logistics, and other applications that use location or proximity as inputs.

V. Beyond Temporary Internet Gateways

The usefulness of BLE devices, and thus their market potential, can be increased by creating permanent, robust and unattended Internet gateways. They may or may not have continuously active Internet connections. For example, Internet connections may be open only during specific time windows, or after an event has been detected. However, they are *not* depending on the presence of human users.

A permanent Internet gateway should be a special-purpose device in the sense that its operation is dedicated purely to the gateway functionality. However, it should *not* be a special-purpose device in the sense that you end up buying a gateway for every specific application, e.g., a gardening gateway, a home security gateway, a home cinema gateway, a lighting gateway, etc.

To avoid a proliferation of nearly identical gateway boxes, it should be possible for all BLE applications to share a single Internet gateway. To achieve interoperability of any BLE Internet gateway with any cloud application, a standard is required for the interface between gateway and the Internet.

Like HTTP is the application layer of the Internet, the *Generic Attribute Profile* (GATT) is the application-layer protocol of BLE. There are mainly two approaches how BLE Internet gateways may operate: either with integration below this GATT layer, or with integration *at* the GATT layer. The Bluetooth SIG has standardized the latter with a bidirectional mapping from GATT to HTTP.

We will first take a brief look at GATT, continuing with this standardized GATT-HTTP mapping.

VI. GATT Protocol

Basically, GATT allows the reading and writing of remote variables (called *characteristics*), e.g. the temperature as measured by a BLE temperature sensor, or the open/closed state of a BLE valve actuator. The length of transmitted values typically ranges between 20 and 40 bytes.

Related *characteristics* are grouped into *services*. A service is a reusable interface. For example, the *Device Information Service* provides information about the device's manufacturer and model number, and the *Heart Rate Service* exposes heart rate information. A growing number of such services are standardized [5], but it is possible to define custom services as well.

Services are a key concept of BLE, enabling a truly service-oriented architecture on the smallest of devices. A BLE device provides one or more services that can be used from a smartphone, tablet, or from a Web application via a gateway. Over the lifetime of a BLE device, the device may provide a changing set of services. For example, after a firmware update, the device may provide useful new services in addition to the old ones.

A service is defined as an immutable set of characteristics. This means that if and when a service is published, no characteristic may be added, removed, or changed anymore. A service can be regarded as a contract that the device (or rather, its developer) has signed, and on which clients of the service can rely.

If new functionality should be added to a product, in a new product version or after a firmware update of existing devices, then at least one new service needs to be provided, which may extend an existing service with added functionality, or which may be used as a replacement for the old version of the service(s).

The motivation for this design is to attain a well-defined and graceful application behavior, no matter what versions of BLE devices and clients happen to come into range with each other. The problem is that a BLE app or other BLE client *depends* on certain capabilities of the BLE devices to which it may connect. In general, there is no way to guarantee that only the most up-to-date BLE devices interact with only the most up-to-date BLE clients, so compatibility cannot be assumed. Services are a way for BLE devices to negotiate a compatible set of capabilities with their BLE clients.

To illustrate this idea, consider the ways in which a BLE device (the server) may meet a client device (here a smartphone with a BLE app), assuming both support the same use case. Further assume that an original service *ServiceA* had been defined for this use case, and later a replacement *ServiceB* was defined that offers improvements in functionality, e.g. by adding more characteristics. Newer versions of the peripheral define both *ServiceA* and *ServiceB*. Then, there are four possible constellations, as visualized in Figure 3:

1) Original app meets an original device: The app discovers *ServiceA* on the device. It can then use the device.

2) Original app meets a new device: The app discovers *ServiceA* on the device. As the app doesn't know about *ServiceB*, it does not try to discover it. However, it can still provide the reduced functionality defined in *ServiceA*.

3) New app meets new device: The app discovers *ServiceB* on the device. The app knows about, and can take advantage of, the additional characteristics in *ServiceB*.

4) New app meets original device: The app looks for *ServiceB* on the device, but the device responds with an error message indicating that the service is unknown. Then, the app tries looking for the older *ServiceA*, which is available. So the app can gracefully fall back to this older service.

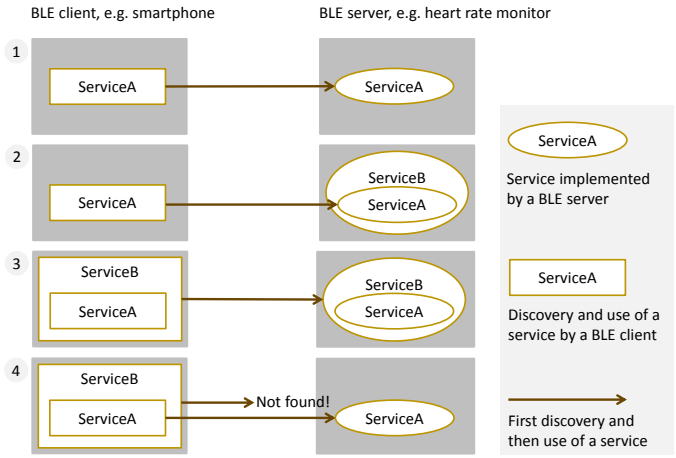

Figure 3 - Possible version configurations

Note that an app does not need to support an old and completely outdated service forever. If the service is not sufficiently relevant in the market anymore, a new app version may leave out support for the old service. Similarly, a new firmware version for a BLE device may not support the old service anymore.

To avoid a frustrating user experience with incompatible devices, a robust dependency management mechanism such as the one described above is needed. Otherwise, end users would quickly become confronted with the Internet of Things version of "Dependency Hell", the opposite of Plug & Play: devices sometimes work, sometimes some features work and others don't, sometimes nothing works, and it is unclear where exactly the problems come from. With inexpensive devices that may remain in use for many years, and may not even provide the possibility for firmware updates, version mismatches between devices will be frequent. GATT provides a very lightweight set of conventions to address this problem in a robust way.

VII. GAP REST API AND GATT REST API

Mapping GATT to HTTP in a RESTful way [6] is largely straight-forward, and standardized by the Bluetooth SIG in the *GATT REST API* [7]. Services and characteristics are REST resources. For obtaining a sensor value, a HTTP GET request is sent to the gateway, where it is translated into a BLE read request. Similarly, for setting an actuator value, a HTTP PUT request is sent to the gateway, where it is translated into a BLE write request. Sometimes, such requests need to be sent only rarely, e.g. once per day. Then it is adequate to poll sensors if and when this needed. If frequent updates of sensor values are needed, however, it is possible to subscribe to notifications of new values, using the *Server-Sent Events* [8] mechanism.

A companion specification, the *GAP REST API* [9], specifies how BLE devices can be discovered and how connections can be managed.

HTTP request content is represented in JSON [10] format. An example request is given here:

```
GET
  /gatt/nodes/00:1B:C5:08:50:70
  /characteristics/0003_0000/value
  HTTP/1.1
Host: gsiot-ffmq-ttd5.try.yaler.net
```

The response to this request has status code 200 (OK), content type "application/json", and the following content:

```
{
  "self":
  {
    "href": "http://gsiot-ffmq-
         ttd5.try.yaler.net/gatt/
         nodes/00:1B:C5:08:50:70/
         characteristics/0003_0000"
  },
  "handle": "0003_0000",
  "value": "0x00000000"
}
```

First implementations of the GATT REST API can be expected in late 2014, e.g. the *Limmat Gateway* reference design (Figure 4, [11]):

Figure 4 – Limmat BLE gateway

VIII. FIREWALLS AND NETWORK ADDRESS TRANSLATION

A BLE gateway is typically behind a firewall and network address translator. Therefore, a suitable mechanism for accessing the gateway from the Internet is needed.

There are various solutions, e.g. port forwarding [12], VPNs, or reverse HTTP proxies (e.g. [13]).

IX. Future Work

The GATT REST API is the standard approach for integrating BLE and the Web, i.e., for integration at the application layer. There is also work being done on a specification for integration *below* the application layer, by using the BLE physical layer for transmitting IP packets all the way to and from BLE devices. A gateway at this level is basically a fully generic IP router. Even with 6LoWPAN [14] header compression, some impact on message size and power consumption must be expected.

One BLE chip vendor has announced to demonstrate IPv6 over BLE still in 2014 (Nordic Semiconductor, [15]).

Although not related to Internet connectivity, two interesting developments will affect the range that can be achieved with BLE:

A BLE module vendor (BlueGiga, [16]) has shown that optimization of a BLE receiver's analog circuitry can increase the range of BLE considerably. In line-of-sight experiments, a range of more than 400 m has been achieved.

Another BLE chip vendor is already shipping a mesh networking protocol implemented on top of BLE (CSR, [17]). This allows accessing BLE sensors over several hops, using other BLE nodes as relays.

X. Conclusions

There will be a plethora of communication protocols for the last meters of the Internet of Things. For wirelessly accessing battery-powered sensors, BLE is very well positioned – not least thanks to BLE support on all modern smartphones. A less visible asset of BLE is its service-oriented GATT protocol, which is light-weight and supports robust service evolution over time. With the standardized mapping of GATT to an HTTP REST API, BLE can be easily integrated into the overarching Internet of Things.

References

[1] M. Galeev, "Bluetooth 4.0: An introduction to Bluetooth Low Energy", EETimes, July 2011, last viewed on 7-Oct-2014, http://www.eetimes.com/document.asp?doc_id=1278927.

[2] Bluetooth SIG, "Bluetooth Developer Portal", last viewed on 7-Oct-2014, https://developer.bluetooth.org/gatt/Pages/default.aspx.

[3] http://www.polar.com/en/products/accessories/H7_heart_rate_sensor

[4] M. Burns, TechCrunch, "Parrot's Flower Power Plant Sensor gives you a Mobile Green Thumb", last viewed on 7-Oct-2014, http://techcrunch.com/2014/02/28/parrots-flower-power-plant-sensor-gives-you-a-mobile-green-thumb/

[5] Bluetooth SIG, "Bluetooth Developer Portal", last viewed on 13-Oct-2014, https://developer.bluetooth.org/gatt/Pages/GATT-Specification-Documents.aspx.

[6] L. Richardson, M. Amundsen, "RESTful Web APIs", O'Reilly Media 2013, ISBN 1449358063.

[7] Bluetooth SIG, "GATT REST API", last viewed on 8-Oct-2014, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=285910.

[8] W3C, "Server-Sent Events", last viewed on 8-Oct-2014, http://www.w3.org/TR/2009/WD-eventsource-20091029/-

[9] Bluetooth SIG, "GAP REST API", last viewed on 8-Oct-2014, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=285911.

[10] D. Crawford, JSON website, last viewed on 8-Oct-2014, http://www.json.org/.

[11] Limmat site of Oberon microsystems, Inc., "Limmat: where Web meets BLE", last viewed on 13-Oct-2014, http://www.limmat.co.

[12] wikiHow, "How to Set Up Port Forwarding on a Router", last viewed on 8-Oct-2014, http://www.wikihow.com/Set-Up-Port-Forwarding-on-a-Router.

[13] Yaler GmbH, "Access devices from the Web", last viewed on 8-Oct-2014, https://www.yaler.net/.

[14] IETF, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, RFC 4919", last viewed on 8-Oct-2014, http://datatracker.ietf.org/doc/rfc4919/.

[15] Nordic Semiconductor, last viewed on 8-Oct-2014, http://www.nordicsemi.com/.

[16] Bluegiga, "BLE121LR Bluetooth Smart Long Range Module", last viewed on 8-Oct-2014, https://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/ble121lr-bluetooth--smart-long/.

[17] CSR, "CSRmesh Development Kit", last viewed on 8-Oct-2014, http://www.csr.com/products/csrmesh-development-kit.