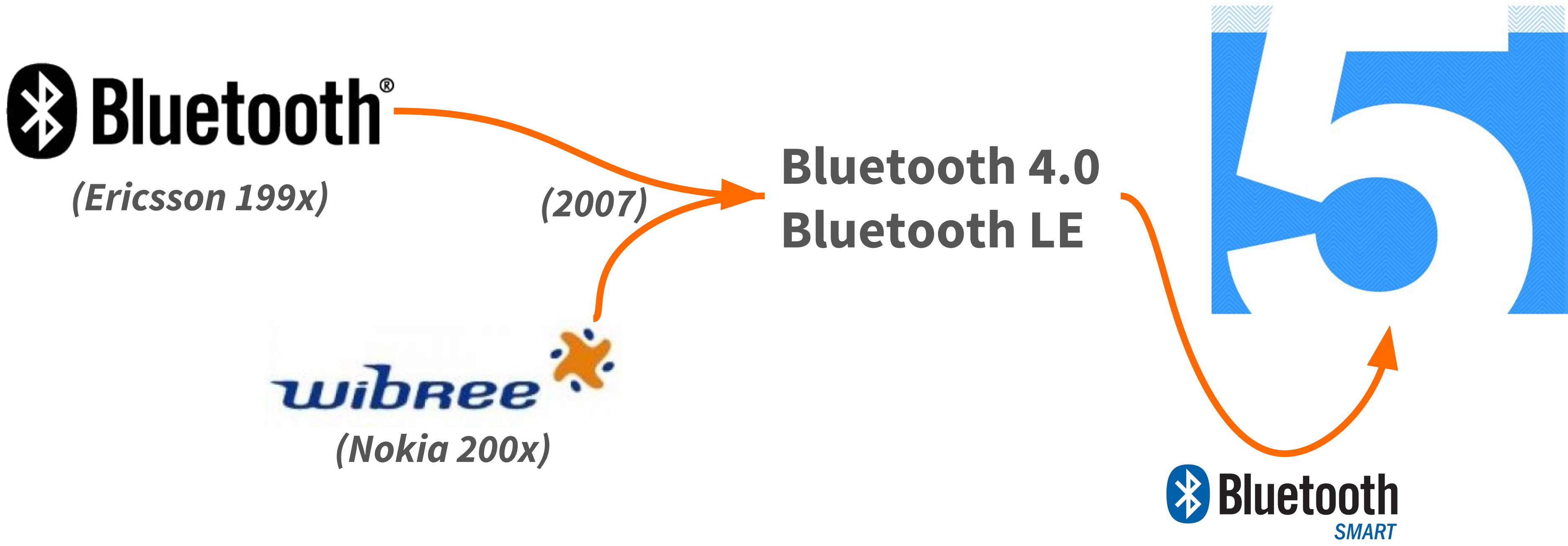
BLE as Active RFID

Tutorial presented by Jeffrey Dungen
at IEEE RFID 2017

What's BLE? (Bluetooth Low Energy)



What's Active RFID?

Device which spontaneously transmits,
via radio frequencies,
its identifier,
using its own source of power.

Is BLE Active RFID?

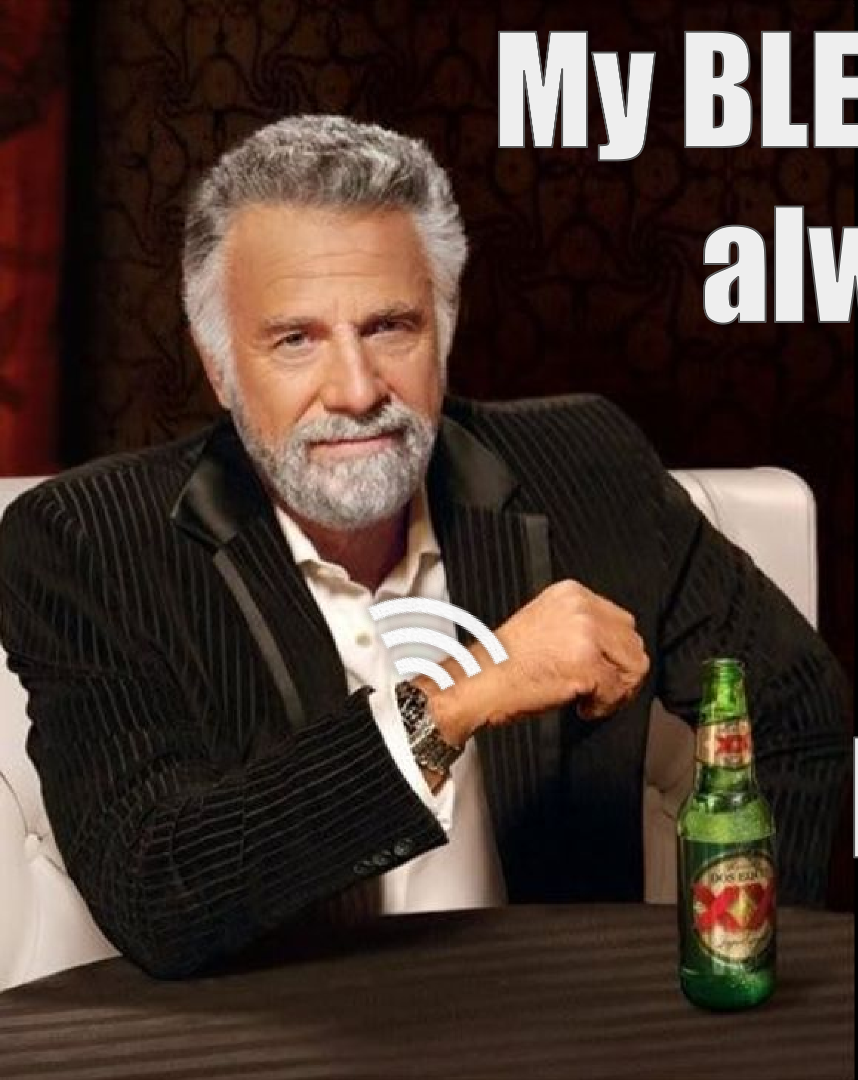
- ✓ spontaneously transmits ("advertises")
- ✓ radio frequencies (2400MHz)
- ✓ identifier
- ✓ own source of power

Is BLE anything else?

Indeed!

***Many* other things!**

But let's talk about the Active RFID part because it's often overshadowed by the rest...



My BLE wearable doesn't always advertise

but when it does it's Active RFID.

Do you realise what this means???

Because I'm not sure I do...

Motivation #1

BLE has become <u>the</u> de facto standard.
No longer need to create yet-another-standard.

I've had the (dis)pleasure of developing Active RFID protocols from scratch at Purelink Technology (5.8GHz) and at reelyActive (sub-GHz).

Couldn't be happier to adopt BLE as a global standard!

Motivation #2

Billions of products, places and even *people* are carrying Active RFID devices right now!

If you had told me a decade ago that this would happen *(voluntarily even!)*, I would not have believed you.

IncrediBLE! Now let's put this to good use!

This. Changes. Everything.

Questions we'll answer

→ *How* are BLE devices **identified**?

→ *What* can you include in the **payload**?

→ *What* about **privacy** and **security**?

→ *What* best (and worst) practices are emerging?

→ *Can you* build a **RTLS** with BLE?

→ *What* **tools** are available?

How are BLE devices identified?

BLE Device Identification

MANDATORY

- ➔ **48-bit advertiser address**

Example

```
48:b1:7a:dd:4e:55
```

OPTIONAL

- ➔ Short name (ASCII)
- ➔ 128-bit UUID
- ➔ 16-bit company code
- ➔ 16-bit member services
- ➔ EUI-48 / EUI-64
- ➔ User-defined IDs

48-bit Advertiser Address

A single header bit, **txAdd**, affords *two* options:

PUBLIC OPTION

→ IEEE-assigned MAC

→ Static

RANDOM OPTION

→ Choose your own!

→ *Change it* whenever and as often as you like!

MANDATORY

Local Name

Choose a short ASCII string, ex:

```
((( I <3 RFID )))
```

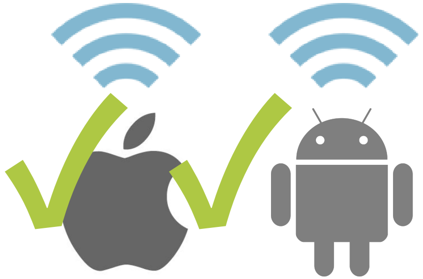
OPTIONAL

128-bit UUID

Choose your own, ex:

`128B171D-1EEE-4F1D-2017-85004C090517`

OPTIONAL



16-bit Company Code

Request from the Bluetooth SIG, ex:

`004C` → **Apple**

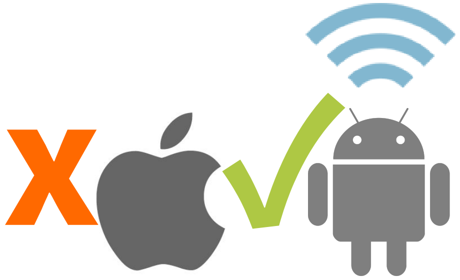
OPTIONAL

16-bit Member Services

Purchase from the Bluetooth SIG, ex:

FEAA → Eddystone

OPTIONAL



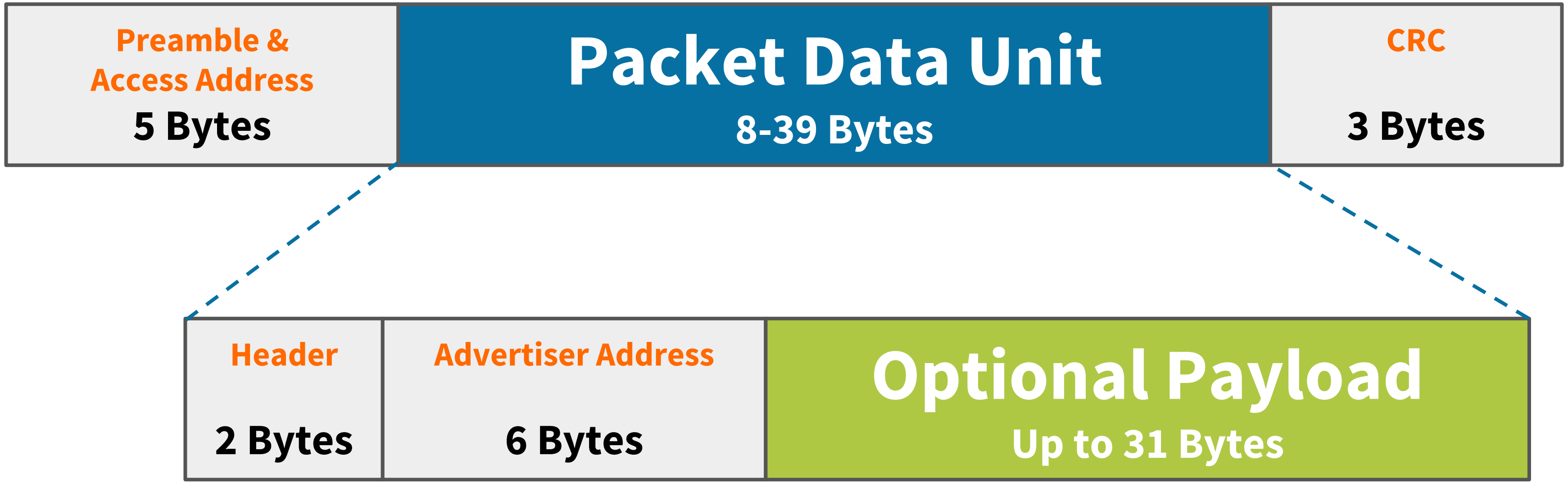
Identification Summary

Every packet includes a **48-bit advertiser address**.

Each packet *may also contain* one or more **additional identifiers**, limited by the *max payload* of the packet.

What can I include in the payload?

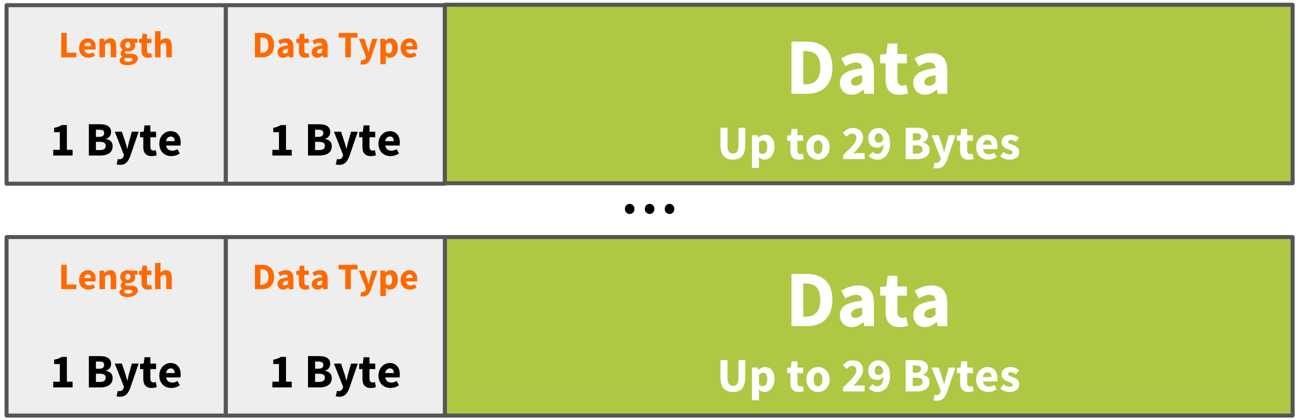
BLE Packet Overview*



** Bluetooth 4.x advertising packets*

31 Bytes of Payload Freedom?

Sure, as long as you respect the **Generic Access Profile** (GAP):



Pick and choose data types, as long as together they all fit!

What’s a GAP Data Type?

`0x01`	Flags
`0x07`	Complete List of 128-bit Service Class UUIDs
`0x09`	Complete Local Name
`0x16`	Service Data - 16-bit UUID
`0xff`	Manufacturer Specific Data

Full list: www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

Examples

How about some **ASCII text** and a **128-bit UUID**:

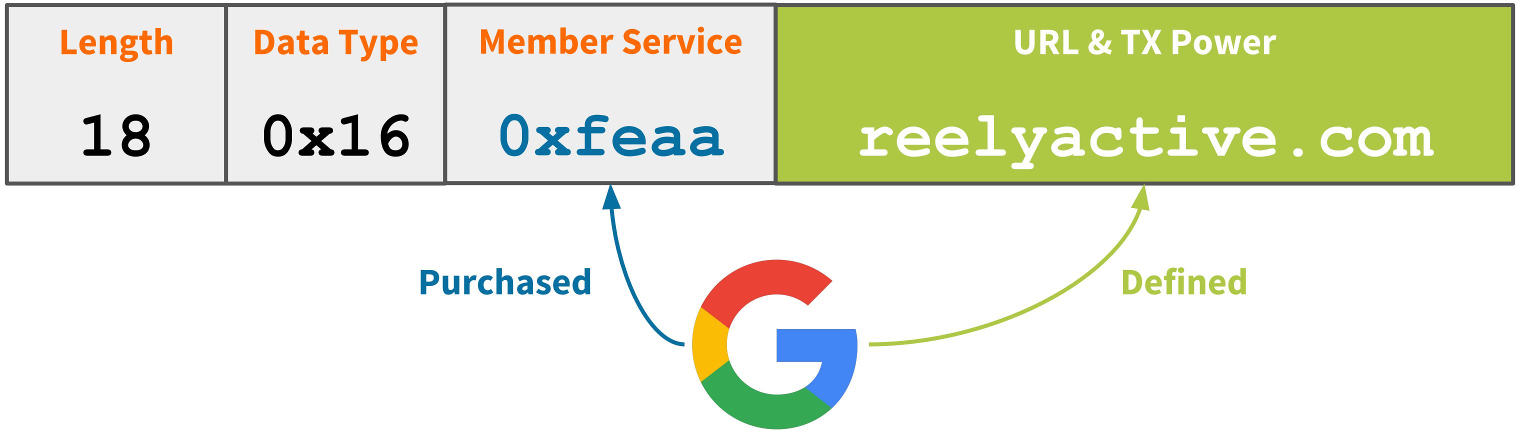
Length	Data Type	Complete Local Name
18	0x09	(((I <3 RFID)))

Length	Data Type	Complete List of 128-bit Service Class UUIDs
17	0x07	128B171D-1EEE-4F1D-2017-85004C090517

Together they're over 31 bytes so *won't fit* in a single packet!

Service Data

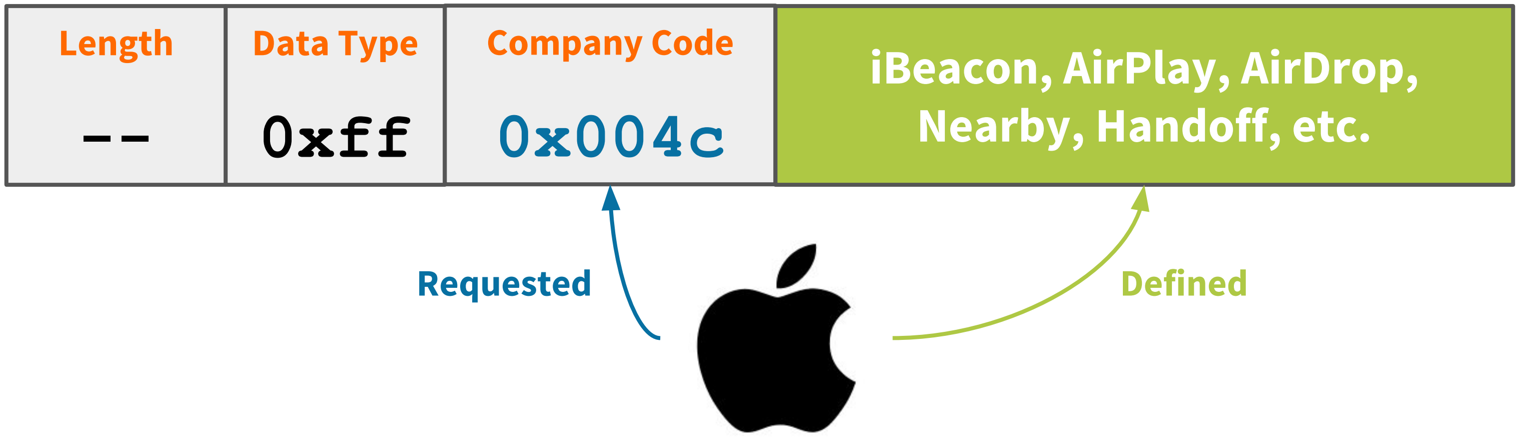
Eddystone uses **member service data** to squeeze in a **URL**:



Eddystone specification: github.com/google/eddystone

Manufacturer Specific Data

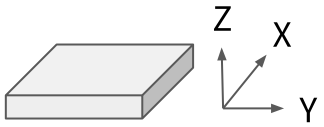
Apple uses **manufacturer specific data** extensively:



iBeacon is an open standard. Others are not.

Payload Data we've Observed

Battery Level
Appearance
URL



Accelerometer
Gyroscope
Magnetometer

Temperature
Pressure
Humidity

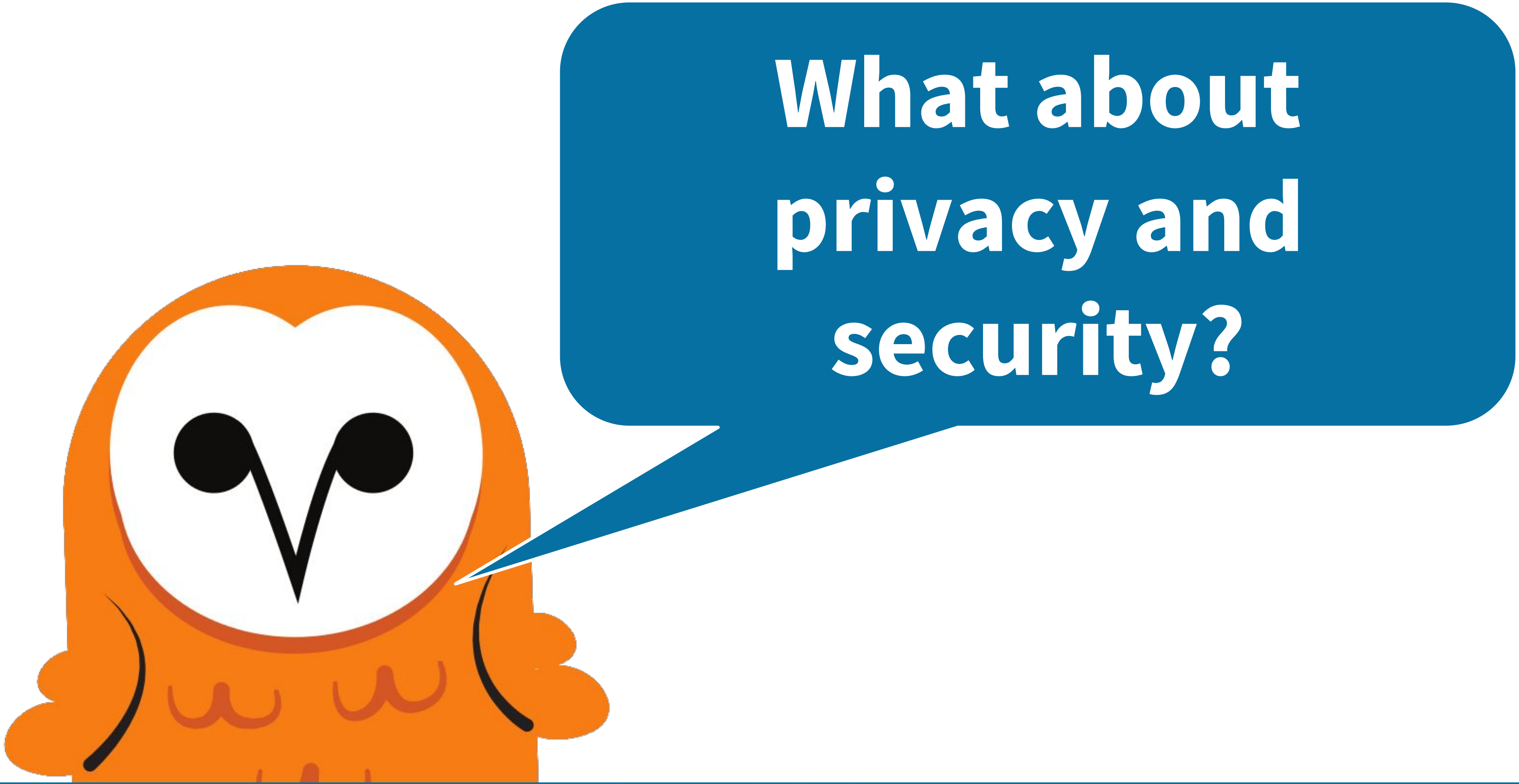
Typically *closed/proprietary* standards, *poorly documented* or *incorrectly implemented!*

➔ Nonetheless, can often be deciphered through observation

Payload Summary

Up to 27-bytes which you can stuff as you please.

Respect **GAP** and **vendor-defined** open standards.


What about privacy and security?

Overview of Concerns

Can I now be **identified** & **tracked** by all the BLE devices I carry???

Can my identity or sensor payload be **spoofed**???

YES.

Advertiser Beware

Transmissions on the advertising channels can be *observed* on the advertising channels.

BLE affords plenty of flexibility for privacy/security.

Apply **best practices** to reach **the best compromise**!

Best and worst practices?

NotaBLE Practices

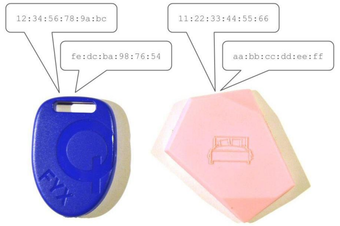
→ Privacy-sensitive identification

→ Making standards work for you

→ Security by obscurity

Privacy-Sensitive Identification

Periodically cycle the 48-bit advertiser address to hamper repeat-visit tracking and spoofing:



Type: **random**
Cycle: **every TX**

EXCESSIVE

Type: **random**
Cycle: **~15 mins**

BALANCED

Type: **random**
Cycle: **never**

INSUFFICIENT

GOOD: ~15 min cycle

An **observer** can:

- → ***easily*** **track** you for up to ~15 mins *(ex: store visit)*
- → ***possibly*** **track** you for longer, while in range
- → ***not*** **associate** you with a previous visit
- → ***identify*** **device type**, at best, by company code or other identifiers, *if present*

(Potentially) BAD: no cycle

Jeff's Fitbit Charge HR has used the **same identifier** for over two years now...

`d9:01:4f:6b:a8:b2`

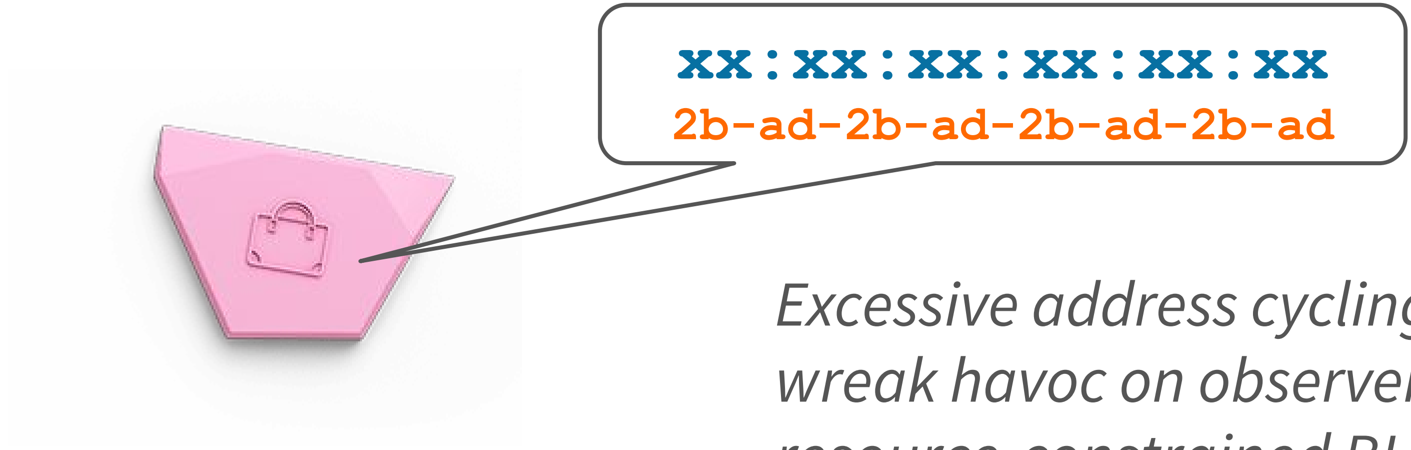
Not good for **privacy**.

- but -

Convenient for **demos**!

BIZARRE: cycle + static ID

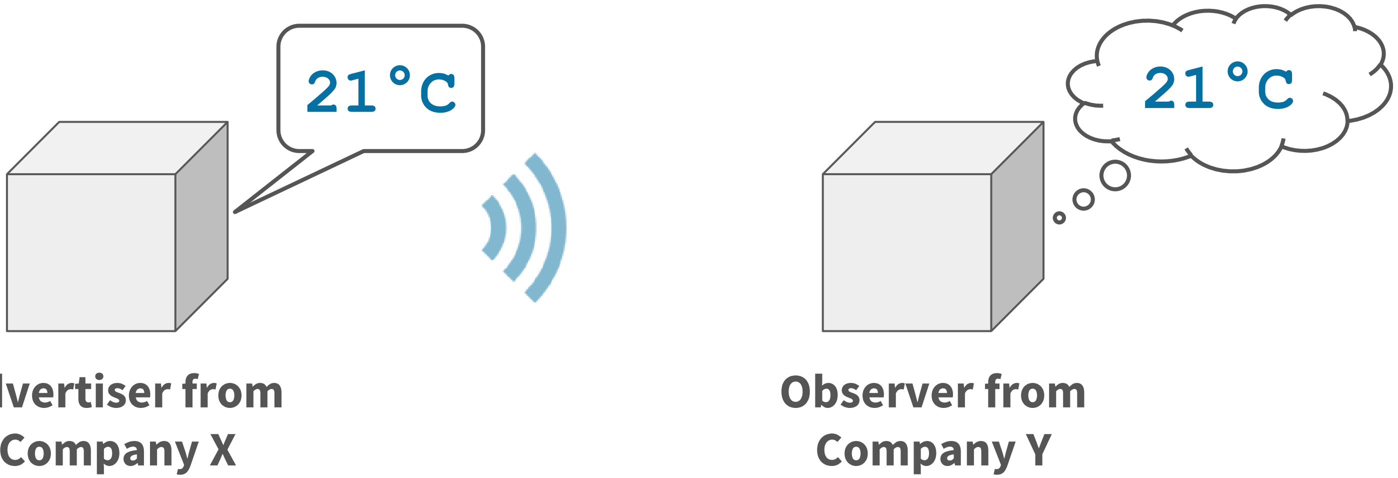
Estimote sticker changes its address **constantly**, but includes **static ID** in payload...



Excessive address cycling can wreak havoc on observers with resource-constrained BLE stacks!

Standards = Interoperability

Beneficial that ***any* observer** understand your broadcasts?



OBSERVE EXISTING STANDARDS

Standard Precedence

1. Check Bluetooth **GAP Types**
2. Check Bluetooth **GATT Services**
3. Check **open standards** by vendors

No standard? Check again.

Still no? Create your own **open** standard.

Temperature Example

GAP: No.

https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

GATT: Yes, service & characteristic.

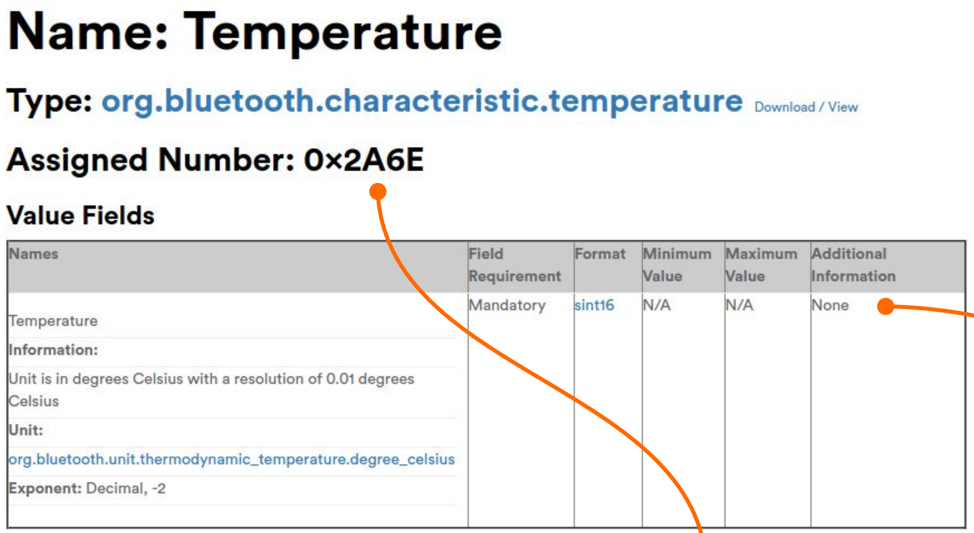
https://www.bluetooth.com/specifications/gatt/services

Service **0x181a:** Environmental Sensing | Characteristic **0x2a6e:** Temperature

Open Standards: Yes. Eddystone-TLM, ...

https://github.com/google/eddystone/tree/master/eddystone-tlm

Temperature-as-a-Service



Name: Temperature

Type: org.bluetooth.characteristic.temperature Download / View

Assigned Number: 0x2A6E

Value Fields

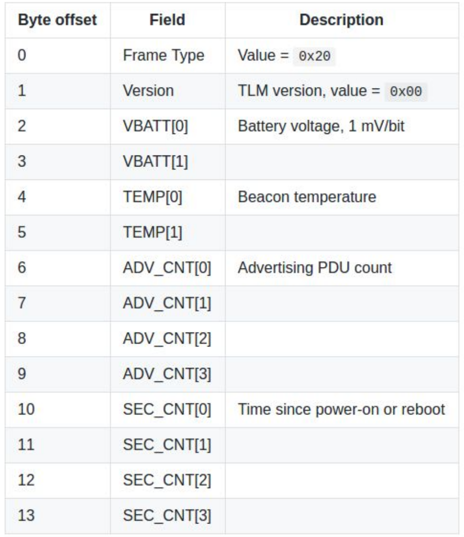
Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Temperature **Information:** Unit is in degrees Celsius with a resolution of 0.01 degrees Celsius **Unit:** org.bluetooth.unit.thermodynamic_temperature.degree_celsius **Exponent:** Decimal, -2	Mandatory	sint16	N/A	N/A	None

** we've observed this practice from reputable vendors and assume it conforms to the core specification!*

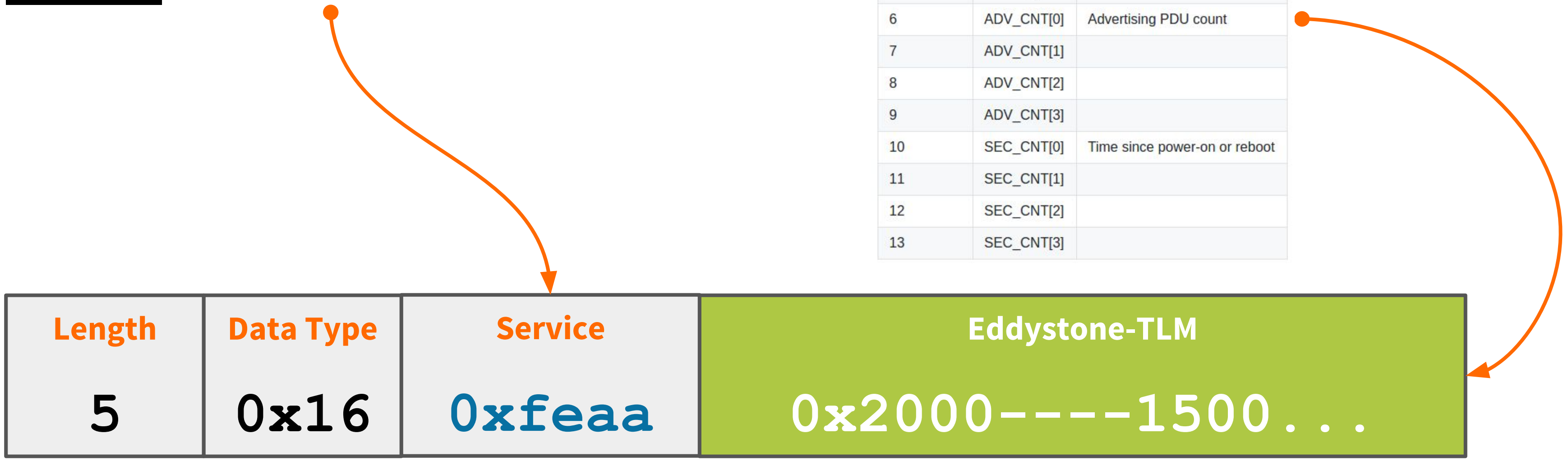
Length	Data Type	Service	Temperature
5	0x16	0x2a6e	2100 = 0x0834 = 21°C

Temperature as Eddystone-TLM

Eddystone

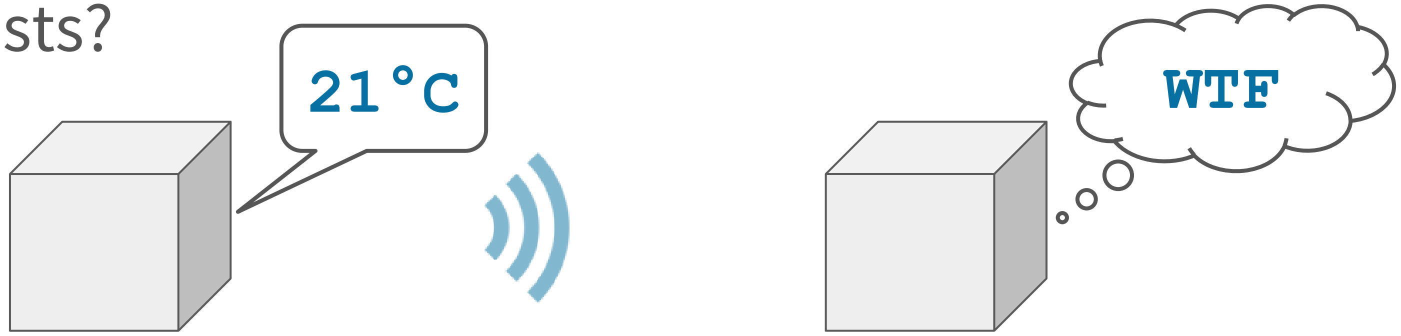


Byte offset	Field	Description
0	Frame Type	Value = `0x20`
1	Version	TLM version, value = `0x00`
2	VBATT[0]	Battery voltage, 1 mV/bit
3	VBATT[1]	
4	TEMP[0]	Beacon temperature
5	TEMP[1]	
6	ADV_CNT[0]	Advertising PDU count
7	ADV_CNT[1]	
8	ADV_CNT[2]	
9	ADV_CNT[3]	
10	SEC_CNT[0]	Time since power-on or reboot
11	SEC_CNT[1]	
12	SEC_CNT[2]	
13	SEC_CNT[3]	



Security by Obscurity

Beneficial that ***no foreign*** **observer** understand your broadcasts?



Advertiser from Company X

Observer from Company Y

DESIGN YOUR OWN CLOSED STANDARD

Obscure Thoughts

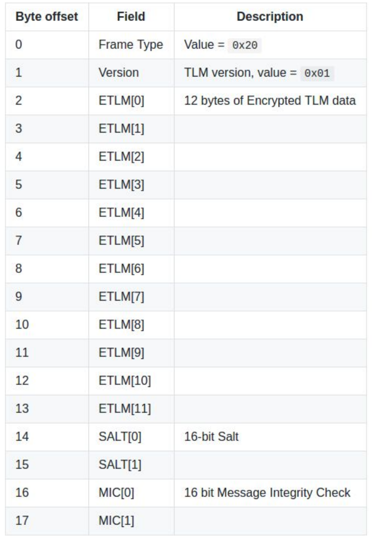
- → Encryption keys
- → Cyclic counts
- → Random noise bits
- → Secret, deterministic address cycling (id & period)

A clever security design will allow your packet to be transported via *any* channel and subsequently decoded and authenticated by a *trusted* recipient. **Think M2M.**

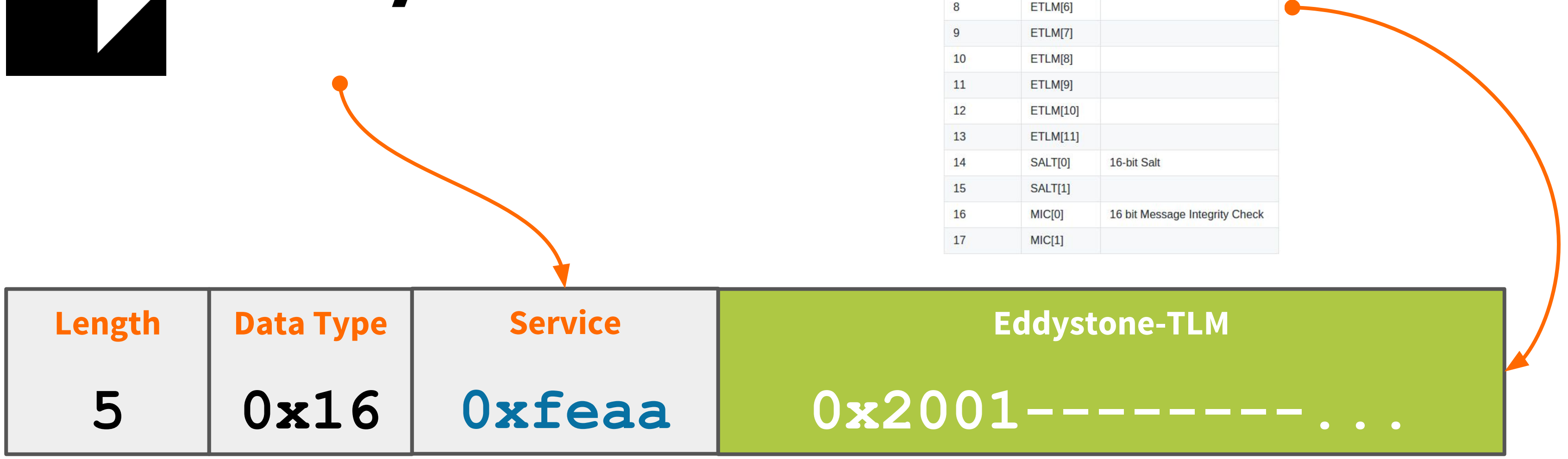
Encrypted Eddystone-TLM

Alternatively, use or inspire yourself from *existing* open standards:

Eddystone



Byte offset	Field	Description
0	Frame Type	Value = `0x20`
1	Version	TLM version, value = `0x01`
2	ETLM[0]	12 bytes of Encrypted TLM data
3	ETLM[1]	
4	ETLM[2]	
5	ETLM[3]	
6	ETLM[4]	
7	ETLM[5]	
8	ETLM[6]	
9	ETLM[7]	
10	ETLM[8]	
11	ETLM[9]	
12	ETLM[10]	
13	ETLM[11]	
14	SALT[0]	16-bit Salt
15	SALT[1]	
16	MIC[0]	16 bit Message Integrity Check
17	MIC[1]	

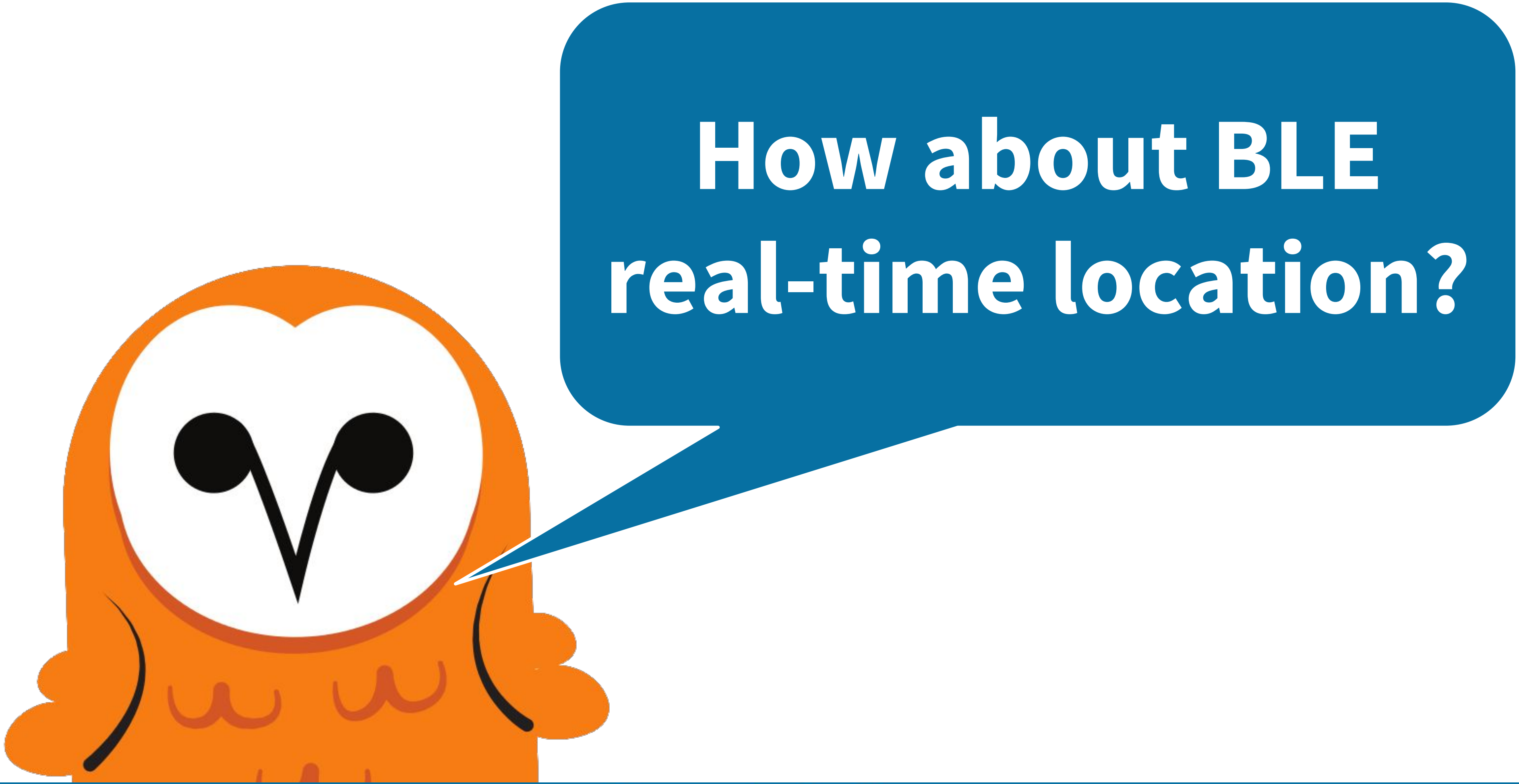


Best Practices Summary

Be sensitive to privacy concerns. Understand it's a compromise.

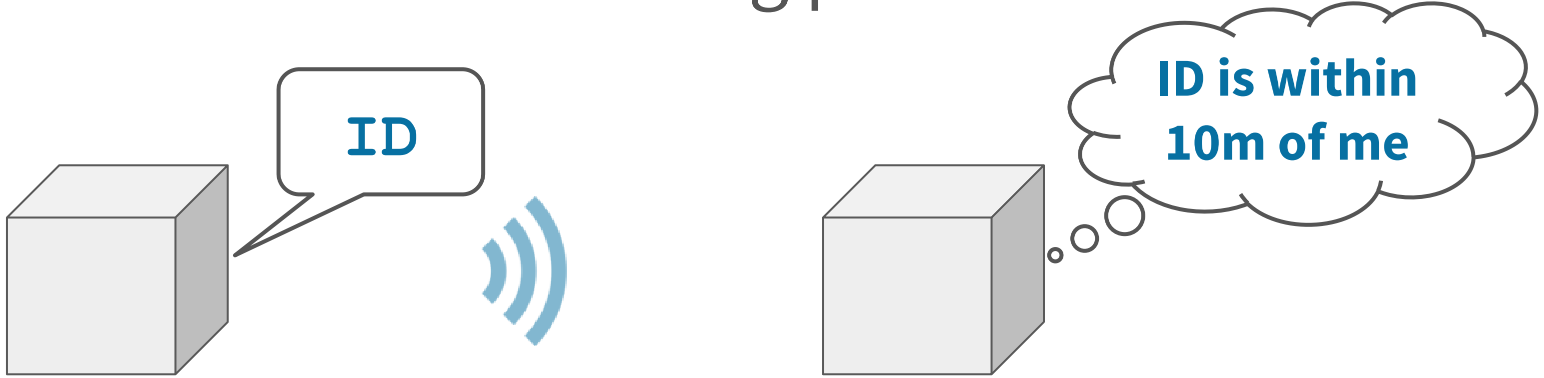
Stick to standards whenever possible.

Leverage BLE's flexibility for elegant **DIY security**.


How about BLE real-time location?

BLE RTLS Overview

Observers can **estimate** the location of a device each time it transmits an advertising packet.

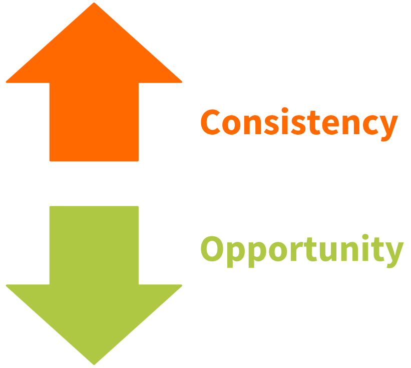


The flexibility of BLE affords *many* options...

BLE RTLS Approaches

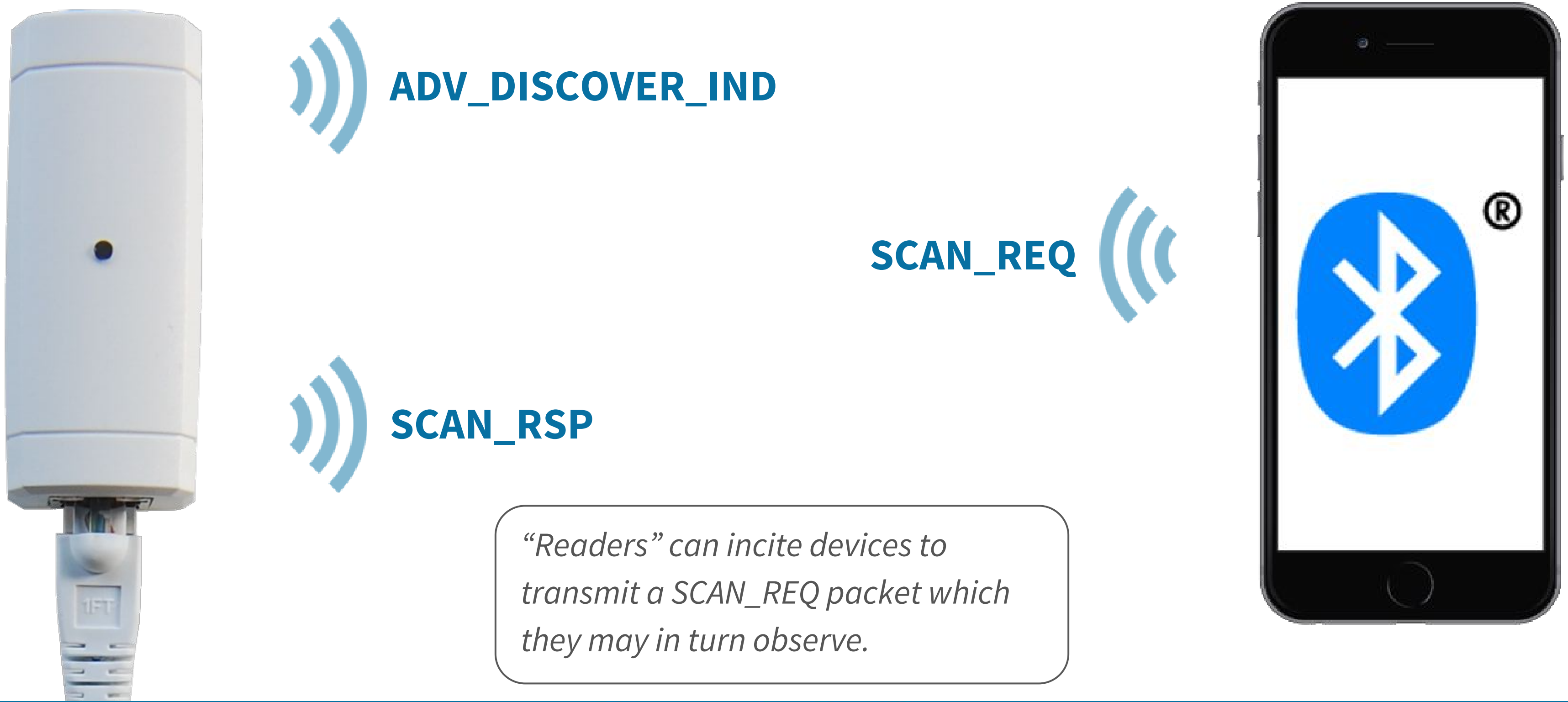
“Bring-your-own-device” & *“use-our-device”* strategies:

Broadcaster	Observer	Vendors
Vendor	Vendor	9Solutions, Kontakt.io, ...
Any*	Vendor	Quuppa
Any	Vendor	Bluvision, (reelyActive), ...
Any	Any	reelyActive



* requires specific bit-pattern in payload

BLE SCAN is “Exciting” stuff



“Readers” can incite devices to transmit a SCAN_REQ packet which they may in turn observe.

What tools are available?

Overview of Tools

As BLE matures, an increasing number of tools and documentation are becoming available ***- but -*** most focus on ***paired*** applications (**central-peripheral**) rather than ***Active RFID*** (**broadcaster-observer**).

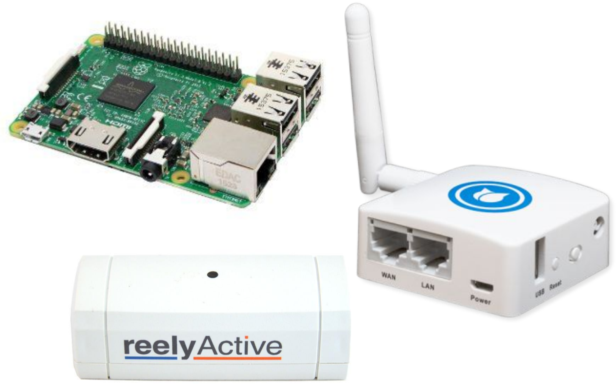
Heed the distinction!

Breakdown of Tools

Advertise

- ➔ Mobile apps/SDKs
- ➔ Commercial beacons
- ➔ Dev kits

Observe

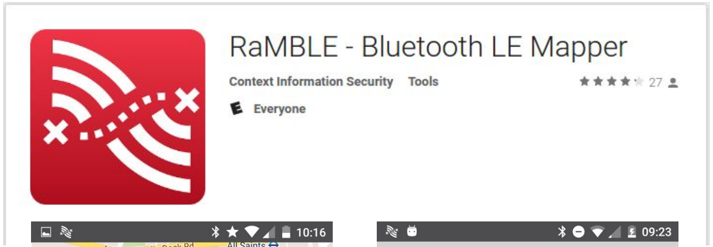


- ➔ Your PC / SBC
- ➔ Commercial sniffers
- ➔ Dev kits

Interpret

- ➔ Open source software
- ➔ Commercial software
- ➔ Develop from scratch

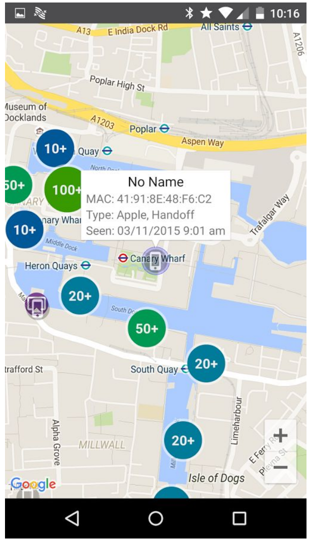
Sniff and Learn on Mobile!

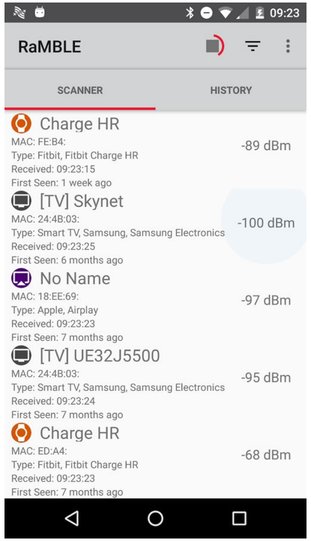


RaMBLE - Bluetooth LE Mapper

Context Information Security Tools ★★★★☆ 27

Everyone





09:23

RaMBLE

SCANNER | HISTORY

Charge HR
MAC: FE:B4:
Type: Fitbit, Fitbit Charge HR
Received: 09:23:15
First Seen: 1 week ago
-89 dBm

[TV] Skynet
MAC: 24:4B:03:
Type: Smart TV, Samsung, Samsung Electronics
Received: 09:23:25
First Seen: 6 months ago
-100 dBm

No Name
MAC: 18:EE:69:
Type: Apple, Airplay
Received: 09:23:23
First Seen: 7 months ago
-97 dBm

[TV] UE32J5500
MAC: 24:4B:03:
Type: Smart TV, Samsung, Samsung Electronics
Received: 09:23:24
First Seen: 7 months ago
-95 dBm

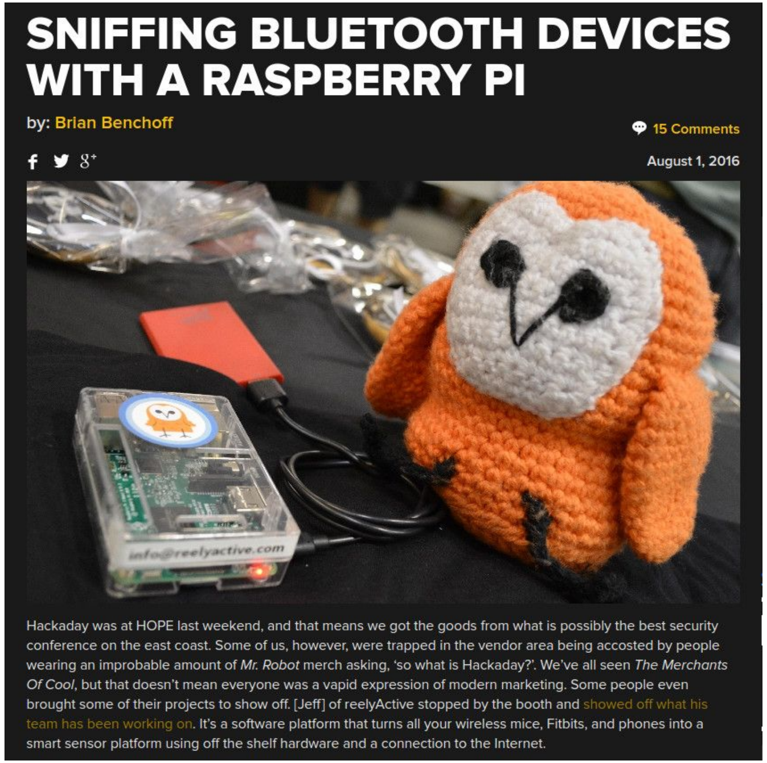
Charge HR
MAC: ED:A4:
Type: Fitbit, Fitbit Charge HR
Received: 09:23:23
First Seen: 7 months ago
-68 dBm

RaMBLE for Android

"RaMBLE collects BLE advertising packets, and tries to identify devices based on their MAC address and the content of these packets."

www.contextis.com/services/research/ramble-android-bluetooth-le-scanner/

Sniff and Learn on a Pi!



SNIFFING BLUETOOTH DEVICES WITH A RASPBERRY PI

by: Brian Benchoff

15 Comments

August 1, 2016

Hackaday was at HOPE last weekend, and that means we got the goods from what is possibly the best security conference on the east coast. Some of us, however, were trapped in the vendor area being accosted by people wearing an improbable amount of *Mr. Robot* merch asking, 'so what is Hackaday?'. We've all seen *The Merchants Of Cool*, but that doesn't mean everyone was a vapid expression of modern marketing. Some people even brought some of their projects to show off. [Jeff] of reelyActive stopped by the booth and showed off what his team has been working on. It's a software platform that turns all your wireless mice, Fitbits, and phones into a smart sensor platform using off the shelf hardware and a connection to the Internet.

Raspberry Pi 3 BLE Sniffer

Detect, visualise and explore BLE advertising packets using the ubiquitous Raspberry Pi, open source software and an easy to follow tutorial.

reelyactive.github.io/make-a-pi-hub.html

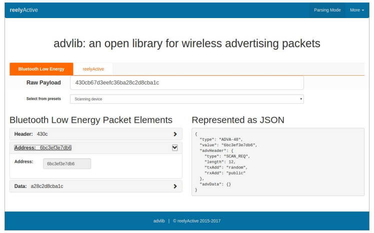
Open Source Projects

advlib

Javascript library to decode BLE packets.

reelyactive.github.io/advlib

Presented at IEEE WF-IoT 2015



advlib: an open library for wireless advertising packets

Raw Payload 430cb67d3eefc36ba28c2d8cba1c

Select from presets Scanning device

Bluetooth Low Energy Packet Elements

Header: 430c

Address: 6bc3ef3e7db6

Address: 6bc3ef3e7db6

Data: a28c2d8cba1c

Represented as JSON

```
{
  "type": "ADVA-48",
  "value": "6bc3ef3e7db6",
  "advHeader": {
    "type": "SCAN_REQ",
    "length": 12,
    "txAdd": "random",
    "rxAdd": "public"
  },
  "advData": {}
}
```

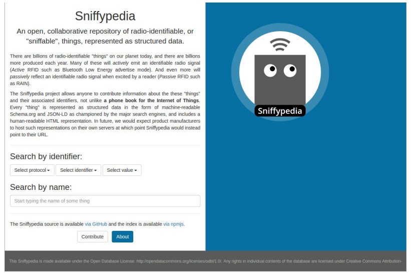
advlib | © reelyActive 2015-2017

Sniffypedia

“Phone book” of BLE identifiers and metadata.

sniffypedia.org

Open Database License



Sniffypedia

An open, collaborative repository of radio-identifiable, or "sniffable", things, represented as structured data.

There are billions of radio-identifiable "things" on our planet today, and there are billions more produced each year. Many of these will *actively* emit an identifiable radio signal (*Active* RFID such as Bluetooth Low Energy advertise mode). And even more will *passively* reflect an identifiable radio signal when excited by a reader (*Passive* RFID such as RAIN).

The Sniffypedia project allows anyone to contribute information about the these "things" and their associated identifiers, not unlike **a phone book for the Internet of Things**. Every "thing" is represented as structured data in the form of machine-readable Schema.org and JSON-LD as championed by the major search engines, and includes a human-readable HTML representation. In future, we would expect product manufacturers to host such representations on their own servers at which point Sniffypedia would instead point to their URL.

Search by identifier:

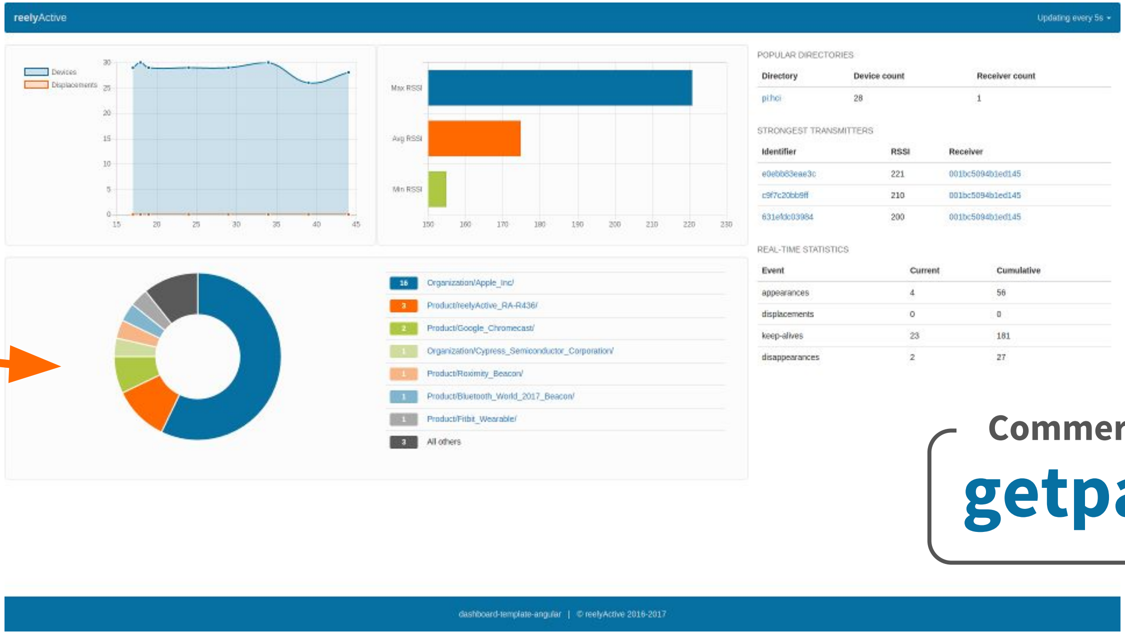
Search by name:

The Sniffypedia source is available via GitHub and the index is available via npmjs.

This Sniffypedia is made available under the Open Database License: http://opendatacommons.org/licenses/odbl/1.0/. Any rights in individual contents of the database are licensed under Creative Commons Attribution-

Live Demo!

This dashboard is open source under MIT License:



POPULAR DIRECTORIES

Directory	Device count	Receiver count
pi:hci	28	1

STRONGEST TRANSMITTERS

Identifier	RSSI	Receiver
e0ebb63eae3c	221	001bc5094b1ed145
c9f7c20bb9ff	210	001bc5094b1ed145
631efdc03984	200	001bc5094b1ed145

REAL-TIME STATISTICS

Event	Current	Cumulative
appearances	4	56
displacements	0	0
keep-alives	23	181
disappearances	2	27

16	Organization/Apple_Inc/
3	Product/reelyActive_RA-R436/
2	Product/Google_Chromecast/
1	Organization/Cypress_Semiconductor_Corporation/
1	Product/Roximity_Beacon/
1	Product/Bluetooth_World_2017_Beacon/
1	Product/Fitbit_Wearable/
3	All others

dashboard-template-angular | © reelyActive 2016-2017

advlib + Sniffypedia

Commercial version

getpareto.com

reelyactive.github.io/dashboard-template-angular

BLE as Active RFID

@reelyActive | jeff@reelyactive.com