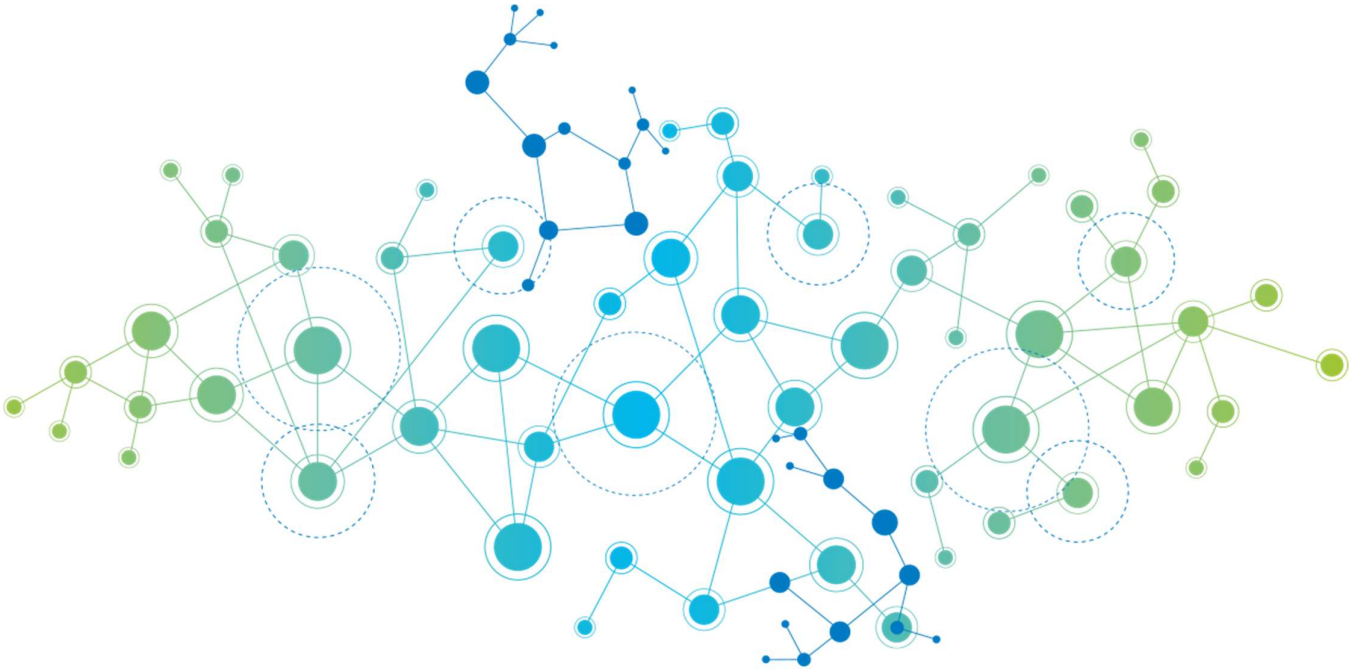
ARUBA IOT GATEWAY SOLUTION

A whitepaper by Yannick SCHAPPLER



Certified Wireless Network Professionals
cwnp®

Durham, NC
866-438-2963 • www.cwnp.com

Elevator pitch

You're about to read a whitepaper about the Aruba IoT gateway embedded in wireless access points; what it is, what it does, how it works, and what you can expect when using it. Some capabilities will be highlighted with concrete examples you could easily reproduce in your lab.

Intended goal: Providing an overview of Aruba's proposal, that can be beneficial at all technical levels.

Table of Contents

Table of illustrations

Something wrong with this document, just want to contact me?
Meet me on slack groups: wi-fipros.slack.com or francewirelesslanpros.slack.com

Introduction

IoT: The Internet of Things, represents the billion autonomous devices connected to a network, primarily the Internet. You could find it at home, in the cities, in the industry, at hospital, and almost everywhere. IoT has given rise to concepts called smart home, smart cities, Industry 2.0 / 4.0 (depending if it's used with or without AI) or smart healthcare. As examples, sensors collect and share data, actuators act on demand or automatically based on simple threshold values, predefined logic (algorithms), and now AI (neural networks and/or machine learning).

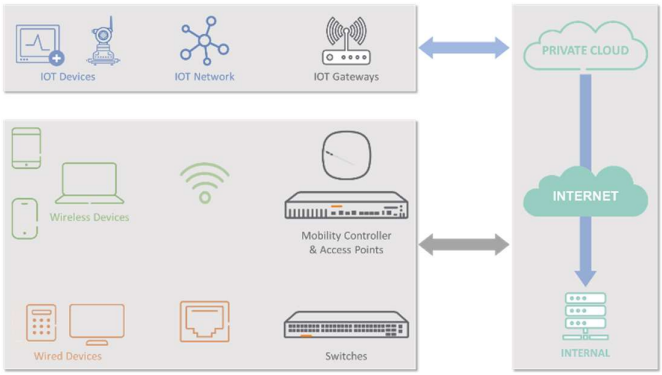
> The ITU has defined the IoT as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" (Recommendation ITU-T Y.2060). The IoT clearly includes M2M (referring specifically to communication directly between devices, used in a vast array of applications and for a variety of purposes), but broader definitions of IoT technologies also include ambient intelligence and smart environments.[1]
>
> CWNP defines IoT as the interconnection of things (physical and virtual, mobile and stationary) using connectivity protocols and data transfer protocols that allow for monitoring, sensing, actuation and interaction with and by the things at any time and in any location.

These things are very often battery powered or use energy harvesting due to their small size and intended usages. By design, they are not built for intensive or massive data transmission. But they are designed for energy saving and limited bandwidth capacities, to achieve small and recurrent messages transmission, and limiting maintenance or battery replacement. You will find things in different sized or structured networks: Personal Area Network (PAN), Local Area Network (LAN), but also over long distances, Point-to-Point, Point-to-Multipoint, Mesh, Clustered Mesh, Star, communicating over centralized or distributed gateways, or routed directly or over intermediate coordinators, relays and gateways.

IoT Gateways: It's the door between the OT (Operations Technology) and IT world, especially telecommunication networks carrying information. Most IoT networks needs backend connectivity to other networks, private or public. Without exchanging with the outer world, objects could only interact between them in a local and isolated manner. The number of gateways follows the IoT network sizing. PAN IoT networks require direct proximity gateways. LAN-like deployments need distributed local gateways in relation with coverage needs or IoT solution range. Operating a Metropolitan Area

[1] Harnessing the Internet of Things for Global Development from ITU/UNESCO Broadband Commission for Sustainable Development.

Network (MAN) or wider IoT network needs distributed gateways spread out on a bigger scale and over farther distances.



Local Networks: If you're operating a Wireless Local Area Network (WLAN) over a building or a campus, you already need to deploy distributed access points handling radio coverage and network capacity in near proximity of users and usages. It's the same for IoT, very close gateways for PAN, more distributed for LAN, but another transport network with different or sometime the same radio media. An interesting concept would be matching or pairing those networks as overlaying layers and mutualizing functions in a unique box (device) that is centrally managed and configured.

Convergence: This whitepaper covers the Aruba IoT gateway solution embedded in its WLAN Access Points, which over shared and dedicated radios merges boxes you would have elsewhere, saving hardware components, reducing global operations costs, and minimizing overall complexity.

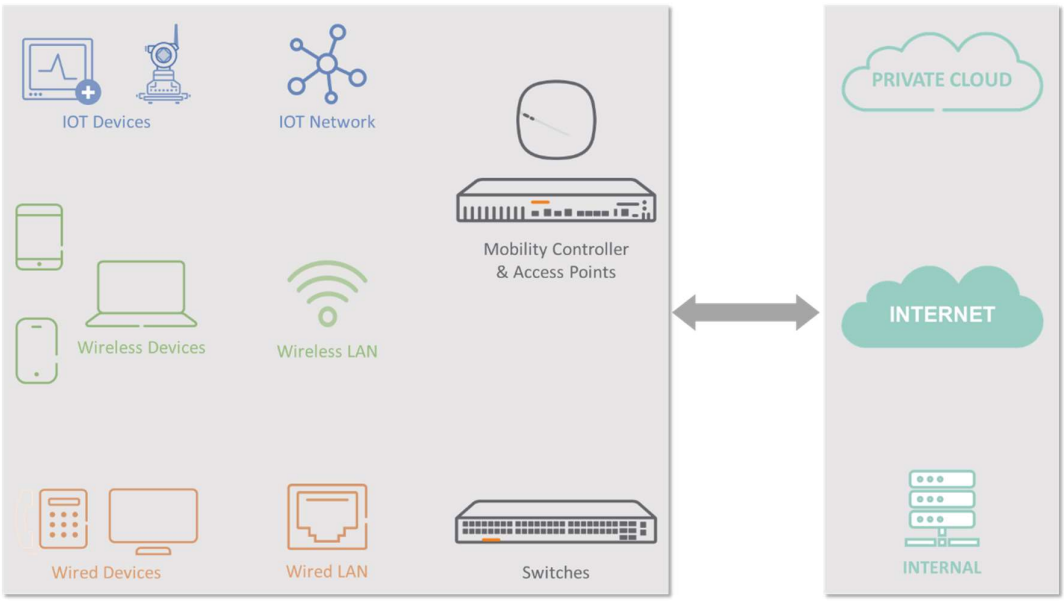


DIAGRAM 1 : ARUBA IOT CONVERGED SOLUTION

Aruba IoT Platform

Keywords and global concepts

Heading North or South?
Both!

As touched briefly in the introduction, a gateway connects different system worlds together, here it is the IT and OT. You've got the IoT ecosystem on one side with its actuators or sensors and the IT world on the other side providing interconnection over networks were data is stored, automatic or manual decisions are taken, where human are interacting with systems through User Interfaces (UI).

Interconnection concepts are sometimes called "polarized": North for the IT world, south for the IoT. The gateway part facing the IT domain will be called northbound, and the one connecting IoT will be called southbound.

Data flow is heading north when a sensor sends temperature reports to the central management system. To do this it will send its protocol messaging flow to the gateway's south interface. If the temperature is considered a bit low, the building automation system decides to set heating in the room two degrees higher. This message is heading south through the north gateway interface.

Communication between "poles":
A translation gateway.

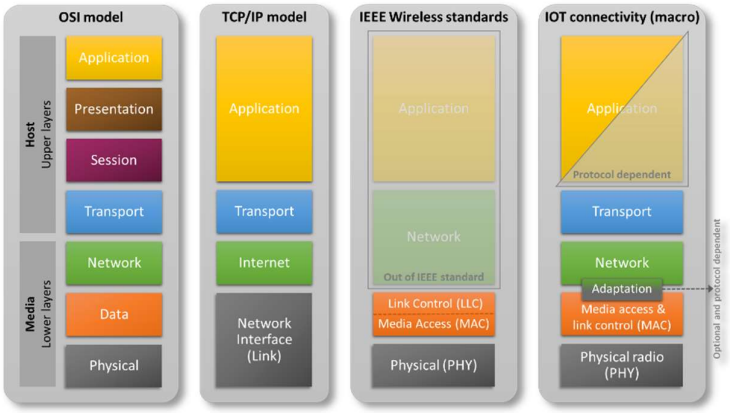
The goal of using a gateway is connecting different worlds together by establishing communication between, even with different languages or protocols, each side. Connecting north to south without sharing a common protocol means that the gateway should do the translation.

The Aruba IoT gateway performs three main functions:

- Media translation → Connections between different physical media and protocol stacks.
- Language translation → Seamless data exchanges between different protocols and languages.
- Interfacing → Interconnect IT & OT functional domains, providing a unique abstracted service point for data presentation, and possibly function execution to consuming systems or users.

The theoretical relationship between different network domains could be related to the OSI model[2] proposed by the ISO, which abstractly describes and organizes the different sub-functions needed in a network communication process based on layers. TCP/IP is the model used by the wired side in most networks, while IEEE standards and global IoT connectivity models are references for wireless connectivity (Wi-Fi, Zigbee, BLE, etc.).

[2] Open Systems Interconnection model (ISO/IEC 7498-1 to 4) from International Standard Organization (ISO).



As per the OSI model: Translation from one protocol to another is achieved by entering a data flow from top to bottom, and then going down in another stack till the base layer, or the Physical Layer (PHY). Physical binary exchanges are done at the PHY, and data exchanges between peers occur in the upper layers. The lower three layers use media and their role is to transmit data on the physical network in a standardized manner, sharing a common format, including addressing and data path decisions.

The OSI model states that each layer establishes a logical relationship with its same level, peer counterpart doing the same process task when protocols are interoperable. Each layer processes the data coming from the upper layer, achieving its specific function and then passes it to the layer below or the data is transmitted on the media if the final PHY Layer is in action.

At the receiver, flow will follow the reverse path from bottom to top decapsulating at each level. In a client to server data flow, data will enter on the application client side, move towards the physical network, be physically transmitted to the receiver, and be processed by the different layers until going out into the server application. Transmission is made at the physical level (physical gateway between two worlds) using a common binary format, but translation effectiveness is distributed at each level where each protocol can exchange with its counterpart.

To summarize: A translation gateway implements two or more different protocol stacks, one per supported interface, binary exchanges are done on the lowest layers and applicative tasks like data conversion, the applicative purpose of a translating gateway, are done on highest applicative layers. The gateway will possess an interface that can "talk" to one world and another interface that can "talk" to the other. Translation between these "talking languages" will occur in the gateway.

Let’s apply this theoretical model to our IoT gateway:

First, a “Thing” necessarily does something, for example by sensing its environment or interacting with it by actuating on something. Its application is reporting some data or executing an action.

Secondly with its connectivity to a network and using a specific protocol, such as Zigbee, BLE or Wi-Fi for example, it connects to the Internet and “other things”, or systems, or with humans through User Interfaces (Human to Machine Interfaces – HMI, Human User Interfaces – HUI, or simply UI).

Thirdly it models and formats the way it interacts with the others, defining I/O interface(s) and a language used to push data. An example would be an API where the data format is JSON (JavaScript Object Notation). This is done at the upper OSI layers, constituting the solution’s application part.

Some IoT protocols, like Zigbee, describe partial or total application layer in the standard specifications. But solutions could also rely on other technologies out of their standard scope, with the examples of BLE or Wi-Fi.

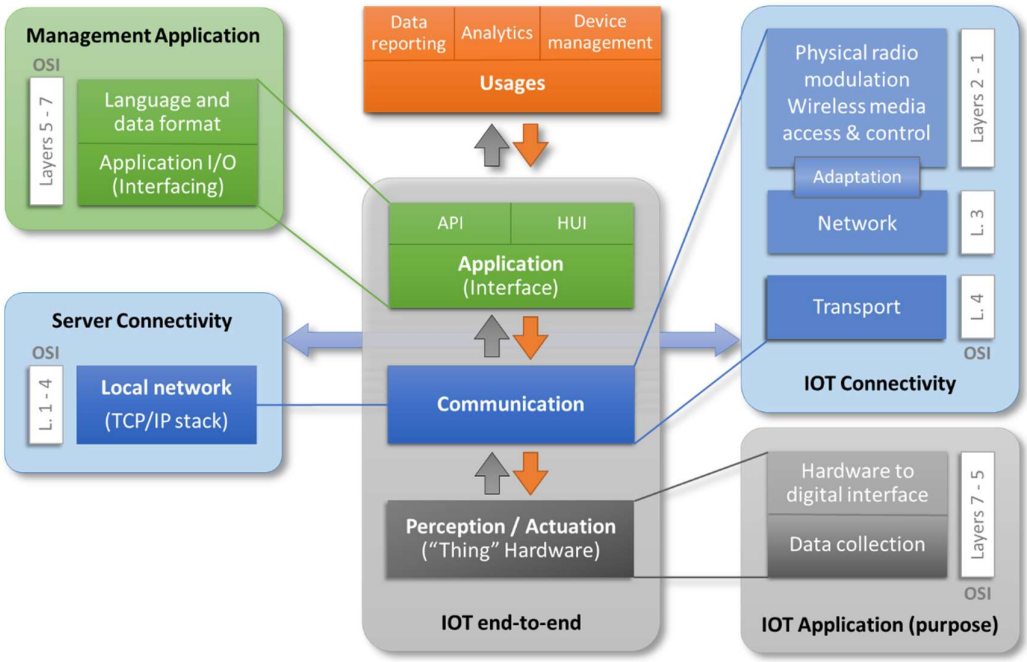


DIAGRAM 2: OSI MODEL APPLIED END-TO-END TO IOT GATEWAY

The translation gateway role is represented by the movement at the platform hardware level from the left column to the right one, or the opposite depending on the flow direction. The AP transfers data frames by adding or removing radio protocol headers on the IoT side, then forwarding or receiving encapsulated data payloads in the IoT server interface to or from the management server.

Focus on the "communication" part, at the gateway level:

Northbound interconnects gateways with enterprise networks using the common IEEE 802.3 ethernet standard and the TCP/IP communication protocol. Recalling the OSI model and its layers, this side relies on the lowest level layers to provide physical media transmission (layer 1), medium access, link and flow control (layer 2), addressing (layer 2 and 3), routing data over the network (layer 3), and data end to end transportation (layer 4). Then, building on top an exchange interface (layer 5 – 7) connecting with management systems, thus reporting analytic data, or taking charge of human interactions though HMI. The upper layers implement proprietary languages connecting to vendor-specific ecosystems, or open systems, to standardize exchanges with heterogeneous data reporting and management platform solutions. As open examples, we can find data export with HTTPS telemetry using the JSON format or bidirectional interaction over WebSocket pipes implementing Google's protocol buffer data serialization format.

Southbound provides connectivity with IoT devices, therefore implementing different protocol stacks and interfaces for supported technology. Each interface or device takes charge of operations

at physical layer (PHY) for physical radio modulation, data link layer processes with medium access control (MAC) for physical addressing, link and flow control, data format definitions, and at the network layer for logical addressing, and path and routing control. The highest layers, application layers, are protocol dependent and will not be detailed here further.

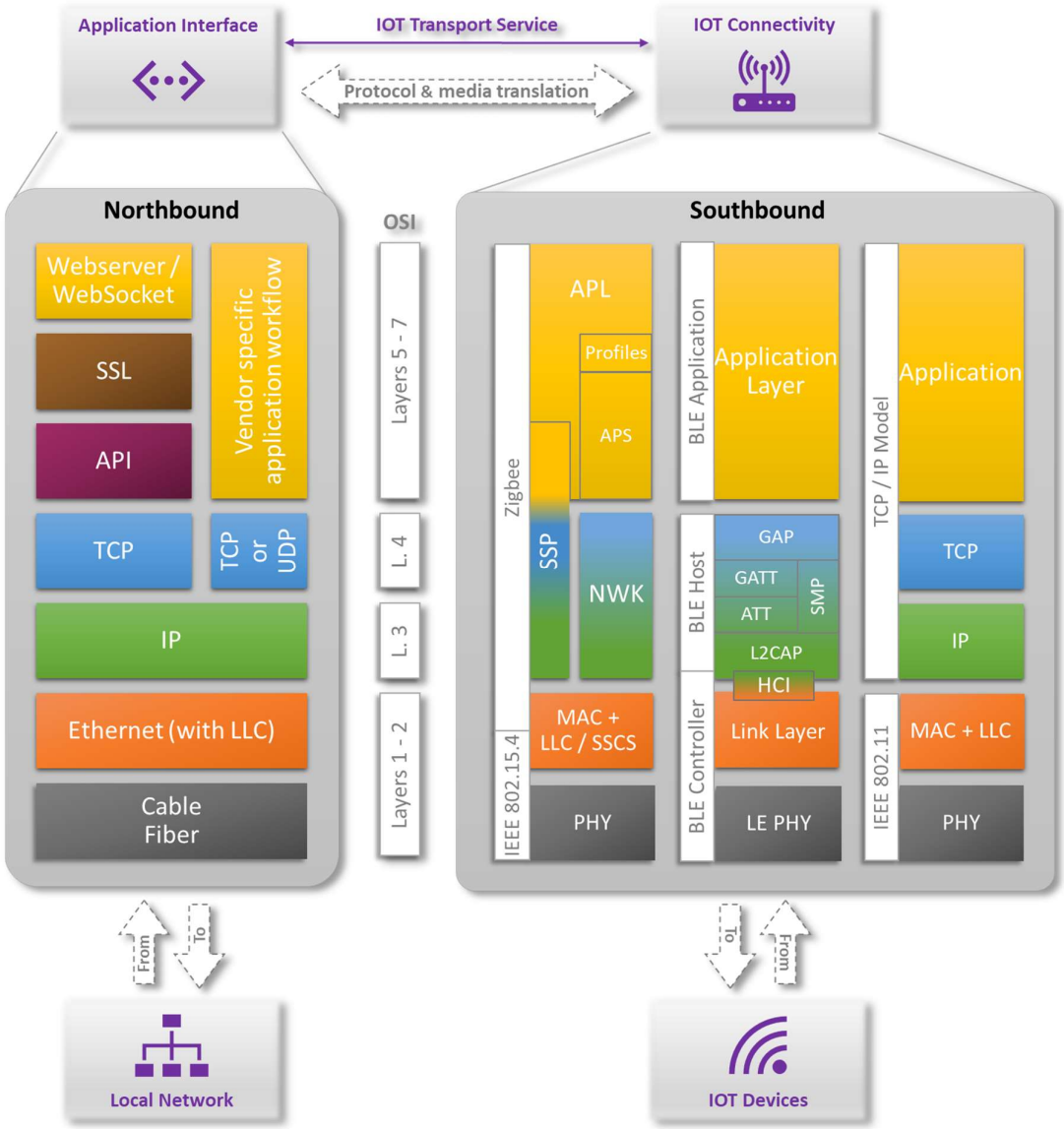
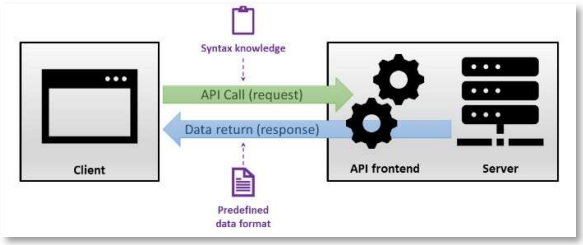


DIAGRAM 2: GATEWAY CONNECTIVITY PROTOCOL STACKS AND THE OSI MODEL

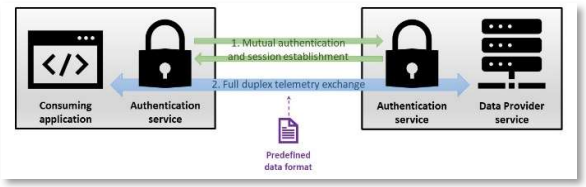
Users and control stations exchange with the northbound processes over open or proprietary protocols based on IP connectivity, while the gateway translates to the southbound processes which interact with IoT devices using different supported protocols. The end-to-end dialog abstracts the whole data flow and different exchanges from user to IoT device or reverse. The gateway does the media and protocol translation as required.

Application interface:
Different types, different usages.

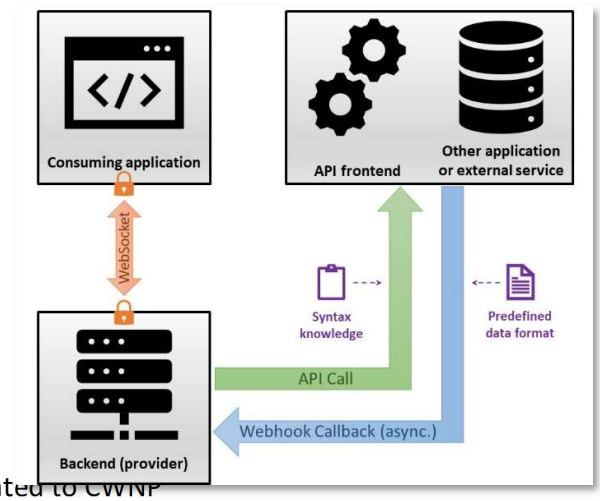
Each application or infrastructure needs a reliable interface to transfer data flows over its different components and to the final consuming users, services, or applications. This could be a static interface with its own commands you must know and use to make requests. Does a machine, a service, or a user acting in real-time over a user interface, ask a question (make a query with the defined commands)? The application interface will respond with a consistently formatted response.



This is what is called an Application Programming Interface (API), usually presented over HTTP/S servers and mostly using formats like XML or JSON for data exchanges. The workflow is static and monolithic: Commands must be known before requesting and you cannot claim something not in the responses of available commands. You'll have to be aware of, and follow, the API manifest. Secondly, flow is unidirectional: Client query the application server, and get a response back. The server can't push or stream data on its own in a request/response model, it only responds to a query. The interface is like a distributed program and treats all requests in a predefined manner.



When message push ability or event response handling is needed, to avoid periodically polling an API, telemetry flows must be considered. The client subscribes to a specific data flow and the server continuously pushes information as it is available. Then you can detect changes and make immediate decisions on the client side. This kind of telemetry interface is called WebSocket. It establishes a full duplex channel between the flow provider and its consumer(s) with a prior mutual authentication exchange. APIs are static and monolithic, WebSocket is more evolutionary by design: the server side simply provides a stream of raw information, processing is done on the client side. Each consuming program can be modified to take different actions based on the same information received. Does the client need to evolve? It can do it autonomously without changing centralized infrastructure operations. This kind of interface presents an additional advantage: bidirectionality. When the provider to subscriber relationship is established, clients receive server dataflows, but the reverse is also true. WebSocket telemetry can be like a live dialog after an initial handshake.



WebSocket operates real-time or in a synchronized manner. If the goal is not only subscribing to a telemetry data flow and acting or reacting on it, but data must be processed asynchronously in the back side, Webhook is a

better option. It provides WebSocket “front channel” with a consuming system, and acts in its backend like an API with other systems. Global functionality is like a reverse API, redirecting WebSocket data flow to another system and acting like a proxy or a “system in the middle”. Webhook scrutinize some specific incoming flow to relay backend or can be designed to react on specific event. In the front side flow continues to be synchronous, with some additional processed data added in the flow coming from another system, treating data asynchronously.

Note: WebSocket should not be confused with non-real-time pub/sub models like those provided by MQTT. With MQTT, subscribers subscribe to topics and receive them close to real-time, but not necessarily in real-time.

Application interface:
Different languages too!

Each application needs an interface to exchange data with other applications or systems. Like it is with humans, different languages with specific data formats (like human language grammars) can be used between peers and they must also share or agree on a common one to be able to understand each other. Interaction can be based on a common language (and by extension its data format) or restricted to the one data format when each system has its own language and only exchanges information with a common data standard.

Flow can be proprietary end-to-end, meaning the IoT ecosystem will establish a dialog with the management / reporting systems, using a language / data format proposed exclusively by the vendor for its physical assets and software suite. Vendor specific ecosystems can open the system to other identified vendors through associations and interest groups or monetized contributions, but it stays proprietary inside the group and isn’t intended to be easily interoperable with outside vendors and applications. In such groups, end-to-end products can also be achieved by associations where a vendor provides the exchange standard, another brings the software part, and another one contributes the hardware, as an example.

On the other hand, language can be open and publicly documented, largely usable without further commercial relationship, easing interfacing between peers or sub-component providers. A widespread data format standard example is JSON[3] (JavaScript Object Notation) used with many different programming languages and systems. JSON, like YAML or XML, is a lightweight plain text format that is human readable. Sharing the same data format makes exchanges easy between systems, relying on the same global language or not. Documentation publicly available, provides necessary knowledge to developers, enabling them to provide tailored interfaces for each application as it is needed.

Note: To be clear, JSON, as a data-interchange format does not define key/value pairs within the JSON specification. JSON provides the "structure" of the document. Just like XML, with JSON, some other shared standards would define key/value pairs.

An example block of JSON follows:

```
{
    "TempSensor103":{
        "Temp":30,
        "Hum":63
    }
```

[3] Présentation de JSON at json.org.

For example, a shared standard may state that temperature values shall be represented as "Temp":30 and the value is transmitted in Celsius.

There are also binary formats like Google's Protocol Buffers[4] (Protobuf), a language-neutral, platform-neutral extensible mechanism for serializing structured data, and BSON or message pack, which provides structured data exchanges too. Binary formats support compression and are preferred for data storage application, inter-application, or internal application service exchanges. It also support more complex data structures than plain text format mostly used for web applications, usually lightweight or clientless. A major difference between JSON and Protobuf excluding the flow human readability, is message delivery using a fixed schema for JSON and user predefined schema for Protobuf where data is also sent along with information about structure and rules defining message exchange. For the second, it's mandatory for both systems to know the data scheme, then being able to serialize / deserialize it correctly, rather than JSON which is fixed and predictable.

Both are designed for data serialization, transfer of complex data structures in a serialized and language neutral byte flow between systems, in a way of direct treatment or data storage. Flow is serialized at the transmitter following a specific scheme defined by the format, and then deserialized at the receiver in a reverse, to reconstruct the original information. The transmitter and receiver don't need to use the same programming language, but only implementing same data format to understand each other.

Profiles: For logical functionnalities and components configuration.

If you know Aruba Network's mobility products philosophy, you are aware of profiles! Each configuration line regarding a logical functionality, a hardware configuration, component fine-tuning, and the global relationship between all, is done through dedicated profiles. The IoT part of the AP is not an exception and comes with its profiles: IoT transport profile, IoT radio profile, Zigbee service and socket device profiles, etc. Details about the profiles will be delivered later in the document.

Putting puzzle pieces together: Sum up the beginning of this document.

An Aruba Access Point can act as an embedded and converged IoT gateway.

It connects the IoT world with "things" that do their intended purpose, roughly, sensing or acting, on the local network, and by possible extension the entire IT world with the Internet. This is done through "polarized" interfaces (Northbound = IT, Southbound = IoT), and provides:

- Media translation → Connections between different physical media and protocol stacks inside converged gateway hardware.

[4] What are protocol buffers? A language guide and more, at Google developers.

- Application interface → Interconnection of IT and IoT, through a unique application interface.
- Language translation → Seamless data exchanges between different protocols and languages.
- Different solution flows → Providing proprietary functionality for specific vendor ecosystems, and open telemetry export or end-to-end bidirectional management based on standardized data formats (like JSON and Protobuf).

Configuration of each different platform sub-component is done through functional profiles instructing, for example, how northbound connectivity will be done with the management server, which type of service will be supported by the platform, and how north will interact with south or how different southbound radios will operate on their domain, and more.

What's inside, how it works?

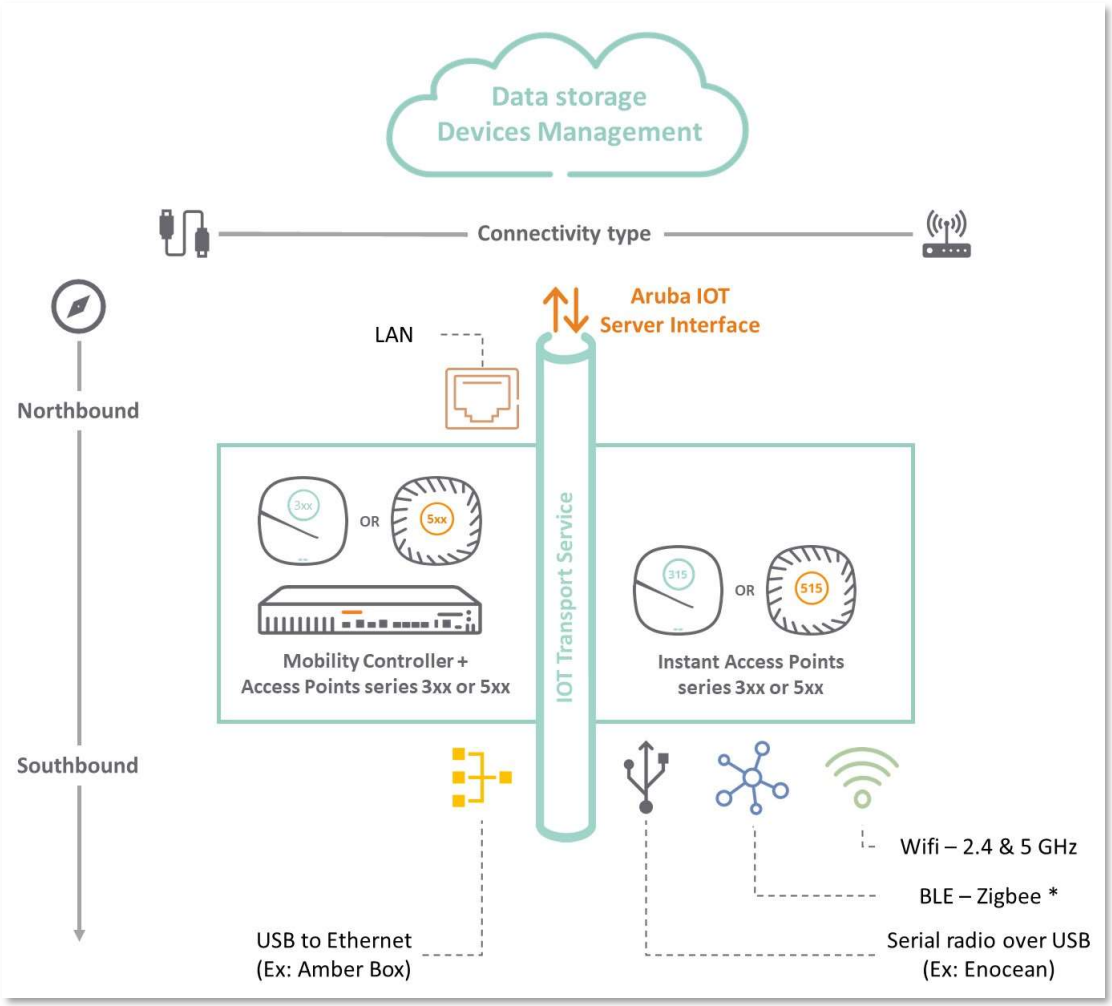


DIAGRAM 3 : IOT GATEWAY COMPONENTS

** Note: AP 3xx series needs a separate USB dongle for Zigbee connectivity.*

Platform hardware, southbound supported IoT technologies.

The supported hardware platforms include Instant APs (IAPs), or Campus APs (CAPs) connected to a mobility controller (MC). Access Point (AP) hardware must be 3xx series- or 5xx series-based.
For standalone deployments, each AP can act as an IoT gateway and then will present a data interface. Using a mobility controller unified solution centralizes the data / application gateway at the controller central point, relying on the dispersed Access Points to provide proximity network connection to IoT devices. Same concepts than are used for Aruba Wi-Fi solutions, such as the fact that IoT flow uses the same GRE tunnel between an AP and its controller, including the same overall security, encryption, and flow isolation.

3xx series runs a dedicated IoT generation 1 radio, implementing Bluetooth Low Energy (BLE) version 4, where generation 2 presents inside the 5xx series, a BLE version 5 and/or Zigbee. A USB dongle can be added to provide generation 2 radio capabilities, completing the generation 1 connectivity (3xx series), or dedicating technologies on each gen 2 radio (5xx series). When a unique gen 2 radio operates dual mode BLE + Zigbee, it can only transmit BLE but with Zigbee it is bi-directional. The dongle can be added to achieve both technologies bidirectionally.

A specific IoT radio profile configures the dedicated hardware: It defines the radio to be used (internal or external) and which technology it should support (BLE, Zigbee or both). Then for each selected IoT protocol, it also defines its operational parameters like operation mode (beaconing, scanning or both) or console activation for BLE, operational mode (coordinator only) and operating channel selection for Zigbee. Multiple profiles could be defined, in the case of internal and external radio usage.

The USB connector can also be used to provide serial radio connectivity over USB, enabling third party dongles and protocols to be directly pushed or encapsulated through the gateway, like EnOcean or Piera. SES Imagotag is another solution relying on a USB dongle, with its main difference being direct IP tunnel attachment to the software solution, without further treatment by the IoT gateway.

Besides the dedicated IoT hardware, the usual AP radio hardware remains accessible to the IoT gateway. The main ethernet port is used northbound to interconnect with a local network to the back-office management and reporting ecosystem. The auxiliary ethernet port can be configured to provide southbound USB to ethernet connectivity dedicated to specific systems like Hanshow or SoluM Electronic Shelf Labelling (ESL) solutions and AmberBox gunshot detectors. The Wi-Fi embedded radio can provide data telemetry about the wireless environment, Real Time Location Services (RTLS) feed to asset tracking solutions, and of course connectivity to Wi-Fi IoT devices.

A word about wireless radios cohabitations: BLE, Zigbee and Wi-Fi may operate on the same 2.4 GHz frequency band resulting in potential interferences between radios. In fact, each wireless transmitter can only detect neighbors it understands, when all nodes run the same radio protocol,

then implementing protocol dependent media access rules to avoid simultaneous transmission and collision causing frame retransmission. Without running the same protocol, each transmitter will, at least, sense its radio channel and consider others it can't understand as noise or unknown transmissions, then applying deferring techniques and waiting for channel clearance before transmitting, resulting in delays and poor overall performance. To avoid those interference issues due to protocol mismatching between different devices, The AP hardware, which as a translational gateway, is connected to each different network / protocol supported, can help in the arbitration task. The feature is called BLE/Wi-Fi coexistence, is supported with Gen2 radios, and enabled by default. It improves overall transmission performance by coordinating the different radios avoiding simultaneous media access, and preventing inter-modulation thus increasing receiver performance.

"Datasheet style" hardware connectivity summary:

- Wi-Fi[5]: "Inherited radios" operating in 2.4 GHz and 5 GHz frequency bands (country specific channelization), supporting 802.11 a/b/g/n/ac and 802.11 ax only for AP 5xx series.
- Bluetooth[6]: Dedicated BLE only class 1 radio, formerly IEEE 802.15.1 and now Bluetooth SIG standard. No support for "classic" Bluetooth (BR/EDR – Basic Rate & Enhanced Data Rate).
    - Gen 1 radio – AP 3xx series: Bluetooth LE stack version 4.
    - Gen 2 radio – AP 5xx series: Bluetooth LE stack version 5.
- Zigbee[7]: Dedicated IEEE 802.15.4 standard Gen 2 radio (AP 5xx series or dedicated dongle) operating in the worldwide 2,4 GHz band only.
- RJ45 - Ethernet:
    - "Classical" IEEE 802.3[8] main ethernet connection, up to 1000Base-T for 3xx series, and 2500Base-T (802.3bz) for 5xx series. Both are Power over Ethernet (POE) compatible POE-PD: 802.3af / 802.3at.
    - Second Ethernet interface is 802.3 up to 1000Base-t only without POE support. In the IoT use case, this interface is intended for dedicated USB over Ethernet connectivity.
- USB host port: Type A connector, powering up to 1A / 5W. Serial over USB applications dedicated with compatible driver / dongle (EnoCean or Piera only nowadays) in IoT use cases.

Northbound application interface:
Connectivity protocols and data exchange standards.

Facing the back-office equipment is the Aruba IoT server interface. This is the IoT gateway northbound server side exposing a proprietary application interface that handles management and data flows going in or out. It supports different connection transport protocols and data formats, open or vendor specific, but dependent on the configured service under the IoT Transport Profile.

Aruba provides detailed documentation about its IoT platform, available on the ASP Support Portal:
Note: An ASP active account is required to access documentation.

- Aruba IoT Interface Documentation (API Guide).
- IoT Basic Setup Guide (Solution Guide).

If you're not locked to a specific vendor solution, you would likely turn out to open connection protocols and then do your own data analysis locally or cloud-based, such as using a management console suitable to your needs. As discussed earlier in this document, the application interface

[5] IEEE 802.11-2016 standard from the Institute of Electrical and Electronics Engineers (IEEE).
[6] More about Bluetooth standard at Bluetooth Special Interest Group (SIG).
[7] More about Zigbee standard at Zigbee Alliance.
[8] IEEE 802.3-2018 standard from the Institute of Electrical and Electronics Engineers (IEEE).

covers two main types of open connectivity protocols and a third "mixed" one for the cloud use case:

- Telemetry over webserver: Data is exported on the fly from the northbound network interface and sent to any distant HTTP / HTTPS server specified by its IP address and an optional operating port. The data format used is JSON (plaintext format), a schema for telemetry applications is provided for information by Aruba on ASP. Prefer using the "S option", as HTTP sends data in clear text, where HTTPS uses encryption on the payload data, then hardening the overall flow privacy. This protocol is not advised for high volume messaging (many per second) and doesn't support low latency well.
- Telemetry over WebSocket: Provides bidirectional exchanges, exporting data or transferring incoming commands from a management station. The data format is Protobuf (binary), and is preferable for low latency usages and direct data consumption, like system to system or system to data storage. Aruba provides on ASP the corresponding Protobuf specification file to serialize / deserialize data used at both northbound and southbound, and decoding of transiting payloads. The distant endpoint is identified by an IP address and a service port. The flow can be ws:// - WebSocket or wss:// - WebSocket over Secure Socket Layer (SSL). The latter is strongly recommended as the data flow is encrypted with the controller or Instant AP local management certificate, meaning better end-to-end privacy. This is the best protocol choice, as it enables many different usages with various requirements (binary data, high operations per second, low latency) and also supports the full set of IoT gateway functionalities.
- Microsoft Azure IoT Hub[9]: Covers previous telemetry real-time and bidirectional functions but does it with WebSocket over AMQP[10] directly to Microsoft Azure[11] cloud with protocol translation at the gateway level. You need to provision the IoT Hub in Azure Cloud with an active subscription before sending data to the service bus or event hub. Flow remains bidirectional with command execution and data export, but all IoT background management moves into the cloud, where any further treatment would be defined and achieved, and applications hosted.

Regardless of the telemetry flow method, the transport protocol used, shares the same authentication methods:

- OAuth2 authentication via a dedicated server URL using a username and password or a client secret with return of an authorization access token,
- No authentication using a predefined static access token for authorization.

Azure IoT Hub connectivity relies on symmetric group key usage, defined in the Device Provisioning Service (DPS[12]) access strategy for flow authentication.

[9] Aruba IoT transport for Microsoft Azure from Aruba.
[10] AMQP - Advanced Message Queuing Protocol: ISO International Standard ISO/IEC 19464.
[11] Azure IoT Hub documentation from Microsoft.
[12] Azure IoT Hub Device Provisioning Service documentation from Microsoft.

The IoT platform supports commercial end-to-end solutions offered by specific vendors or Aruba partners. In this case, the configuration parameters are usually documented by the solution provider. The third-party solution list is as follows, using the Aruba software version 8.8:

- Meridian beacon management,
- Meridian asset tracking,
- ZF Openmatics (also usable via open WebSocket telemetry),
- Assa Abloy.

All third-party server connectivity with proprietary data format is over secure WebSocket protocol, except the last one using HTTPS webserver instead.

Software versions note: Even though IoT support was introduced with the 8.4 version, Aruba added major IoT improvements starting with the 8.7 release, and a little more with 8.8 and now 8.9.

Connecting north to south:
Transport profile and services.

Recall: The IoT application server, with its connectivity protocol and inherited data format, is attached northbound. IoT connectivity and supported protocols are provided by the platform's southbound radios or wired ports. The logical link between two poles is defined through the IoT transport profile. This configuration entity creates a relationship between IoT technologies supporting devices / final use cases, and different application server-side protocol / data format used to interact from outside of the IoT world. This point of configuration is also the place where IoT service type or device class-based filtering can be applied, reporting interval timer is configured, limiting the transiting flow to only required needs between north and south, and at the end to external servers.

For example: BLE telemetry data exported to a webserver. BLE is the southbound IoT radio technology, telemetry is the use case, or the service offered to end users who consume data, and telemetry over HTTPS, the northbound application server connectivity type, implicitly introducing JSON as its data format. These three components are directly interdependent and together form the IoT transport profile, defining its characteristics that will be entered in the platform configuration.

Roughly, transport is the relationship between radio connectivity and server connection type, while service is globally what use case, what functionality, will be delivered to end-users. Both are interdependent and intimately linked. If telemetry over WebSocket fits for all services or end users' usages, all to all association isn't true, as there's some technical limitations. Check the next diagram for details about possible associations between the server interface, radio and intended service.

Continuing the previous example: It wouldn't be very efficient to report all BLE clients in a facility if the application is only focusing on a unique type of beacon tag. That's where filtering placed at the transport service level makes sense. It would also be fruitless trying to configure BLE connection

from an IoT management server to a managed device with a telemetry service used for unidirectional data reporting. That's where evoked technical limitations will disallow irrational relations implementation.

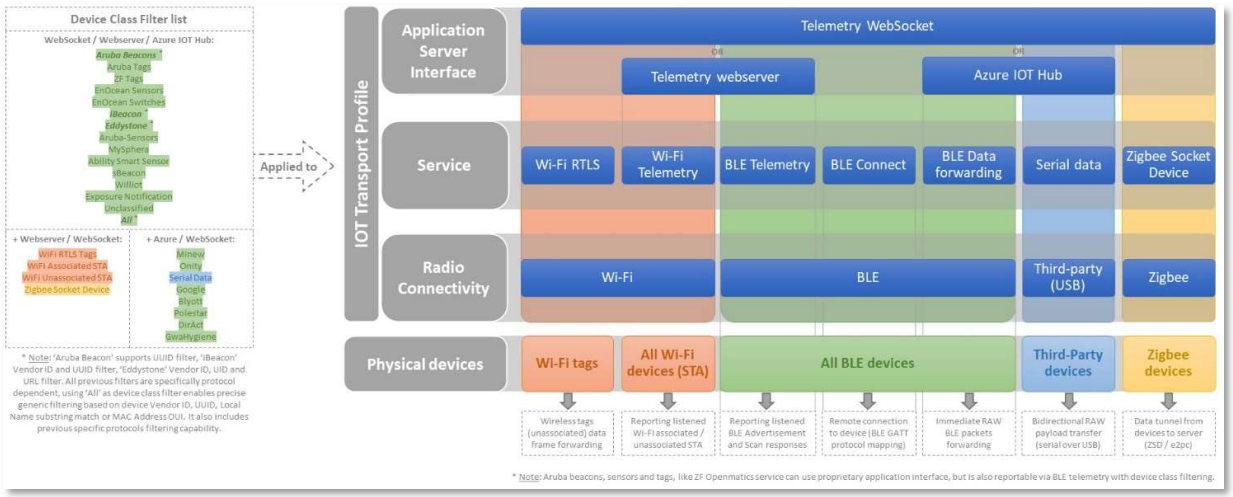


DIAGRAM 4: IOT TRANSPORT PROFILE AND SERVICES (OPEN BASED CONNECTIVITY)

IoT Transport Services details (end-to-end use case) based on open connectivity options:

- **Wi-Fi RTLS data**: Forwards data frames originating from a Wi-Fi Real Time Location Service (RTLS) device to the profile configured RTLS management server. As those devices are unassociated Wi-Fi stations (STA) by design and to avoid confusion with "regular Wi-Fi devices" falling in the telemetry use case, forwarding decision is based on the RTLS destination MAC address parameter configured in the IoT Transport Profile. Running the RTLS use-case, frames are immediately pushed to the remote server when detected, ignoring the profile's configured reporting delay. *Data forward* per device: Received Signal Strength Indicator (RSSI), device class type, and frame payload. *Classification* for Wi-Fi RLTS usage or device class based filtering is 'Wi-Fi RTLS Tags'. That type of global filter common with all other following use cases, defines which selected service will be enabled to cross the IoT application interface.

- **Wi-Fi Telemetry**: Sends a periodic configurable delay reporting, with all STAs in the AP vicinity, or for all managed APs attached to a mobility controller (remember IoT Gateway scale is platform dependent between IAP and MC managing CAP). *Report contains* for each device: STA MAC Address, RSSI and device class type. *Classification* for Wi-Fi devices is data devices association status based: 'Wi-Fi Associated STA', 'Wi-Fi Unassociated STA', or RTLS Tags status based: 'Wi-Fi RTLS Tags'. If last device class can both be found in telemetry or RTLS use case, selected service type will specify the data content that will finally be transferred to the server.

- **BLE Telemetry**: Like it's done for Wi-Fi, this service reports all BLE devices in the standalone AP or aggregated MC managed AP's area. All transmitters advertisements and scan response packets are continuously monitored, parsed, and decoded when possible, then reported at the configured delay interval. An AP can store a maximum of 512 devices' information in its memory, then flushing the oldest entry when reaching the maximum capacity limit. BLE Telemetry is enabled by default for all BLE device class based services and is always active even if other BLE services are in use, like BLE Connect or BLE Data forwarding. *Report contains* for each device: Detailed protocol Information for supported assets (ex: Aruba beacons, iBeacon, ZF Tags) meaning those that could had been parsed, BLE MAC Address and RSSI. With unsupported protocols, at least BLE MAC Address and RSSI is reported.
  Recall: Wi-Fi or BLE Telemetry in addition to WebSocket usage, is reportable on webservers with JSON data format which implies a static predefined data scheme. To help understand possible report content, Aruba provides through its support portal a telemetry example file.

  Classification common for all BLE devices is devices class list predefined, based on vendor identification from Bluetooth SIG member list ID[13]. Complete list is omitted here for brevity and clarity but can be found on previous diagram (device class filter list – left block – highlighted in green). There are also two special device classes: 'Unclassified' reporting all BLE vendors not in the previous list, and 'All' reporting… all device classes! But filterable with Bluetooth SIG vendor ID, including those are not present in the predefined list. This means a possibility with 'All' + specific vendor ID to filter each desired BLE device vendor even if not present in Aruba's predefined list…

  Generic filtering: Company Identifier filter is said generic (user definable), other available exclusively with BLE usages are 'Service UUID[14] filter' a two bytes hexadecimal number also maintained by Bluetooth SIG identifying service type provided by Bluetooth device. Since software version 8.9 there is also 'Local Name filter' matching string value found in the local name advertised in BLE advertisement and scan packets, and 'MAC OUI Filter' to find desired vendor based on its public MAC Address OUI[15] first 3 bytes value. Each generic filter type can be configured with a maximum of ten different entries.
  Recall note: Classification and generic filtering is common for all services based on BLE.

- **BLE Connect:** This service adds the two-way communication mode missing to data reporting. Thus, it only works with a WebSocket application server interface type. This service is provided for any BLE device regardless predefined device classes and enables remote connection to a managed BLE device by interaction with its BLE Generic Attribute Profile (GATT profile[16]). Direct link is made from northbound attached management server throughout the IoT gateway to southbound internal only BLE radio. Starting software

[13] Bluetooth SIG member Companies Identifiers list.
[14] Bluetooth SIG assigned numbers lists (16-bits UUIDs).
[15] Wireshark OUI Lookup Tool.
[16] Video introduction to Bluetooth Generic Attribute Profile (GATT) at Bluetooth SIG.

version 8.8, BLE security encryption is provided with Gen2 radio. Due to its generic operation mode, this service follows predefined BLE devices classification and generic filtering options like other BLE orientated services (more details in previous BLE Telemetry section). Check Aruba IoT WebSocket Interface Guide on ASP, for further information about BLE connection handling.

- **BLE Data forwarding**: Forwards RAW data from BLE devices advertisement and scan response frames. Service immediately pushes content sent from selected predefined device classes to the profile configured destination server. Complete device class list is omitted here for brevity and clarity but can be found on previous diagram (device class filter list – left block – highlighted in green). This service works in close conjunction with telemetry, main difference is the immediate data push rather than a configurable reporting interval. Be careful, RAW data frame forwarding is more bandwidth intensive than a periodic information reporting, and both are transmitted. Like the BLE Telemetry service, Data Forwarding service is enabled by default for all device classes excepted 'Unclassified' since software version 8.8. Classification and filtering follow the same concepts and options previously explained for BLE Telemetry.

- **Serial data**: Or Serial data forward over USB, is based on 3rd party radios with dedicated and supported driver connected on AP's USB interface, then encapsulating vendor specific radio protocol directly to northbound attached management server via the IoT gateway. End-to-end data flow between devices and server, through AP(s) and USB dongle radio(s) are bidirectional, implying a northbound WebSocket connectivity type only. This service literally permits to extend the IoT platform capacities to different vendor specific protocols and ecosystems, with two prerequisite all the same: Dongle should support serial data transfer over USB, partnership between Aruba and the external vendor has to be formalized since specific USB driver is required and should be embedded on the AP software side. Good example here is partnership between Aruba and EnoCean Alliance, and EnoCean USB sticks support by APs. *Classification* for Serial over USB devices is 'Serial data' with no more options.

- **Zigbee**: Provides Zigbee radio connectivity to remote applications through IoT gateway and internal or external Gen2 radio (remember Gen1 is BLE only). IoT application interface data exchange is only supported through WebSocket type for Zigbee use cases. Data flows over inbound or outbound socket handled by the Zigbee Socket Device (ZSD) service, that can be considered as a logical entity linking Zigbee Application Layer (APL) from southbound Zigbee protocol stack to the northbound IoT application interface where data are exchanged with remote application server. Each socket has a unique direction, inbound data is coming from device through Zigbee radio to APL then going out of the AP by the IoT application interface, outbound is the opposite where packets are coming from the northbound interface to APL then forwarded to device over Zigbee radio. Socket configuration is done in a ZSD profile containing one or more socket definition(s), specifying for each the direction (inbound or outbound), source and destination endpoint, cluster ID and profile ID[17]. Global Zigbee radio network settings defining how the Zigbee Coordinator operates, with parameters like Personal Network Identifier (PAN) ID, permission to join the network or security mode usage are configured under the Zigbee service profile[18]. *Classification* can only be 'Zigbee Socket Device' with as second parameter, the identifying ZSD profile(s) to be used.

[17] Terminology in Zigbee Definitions - §1.4.1.2 from Zigbee Specification (document 05-3474-21) at Zigbee Alliance.
[18] Terminology in Zigbee Definitions - §1.4.1.2 from Zigbee Specification (document 05-3474-21) at Zigbee Alliance.

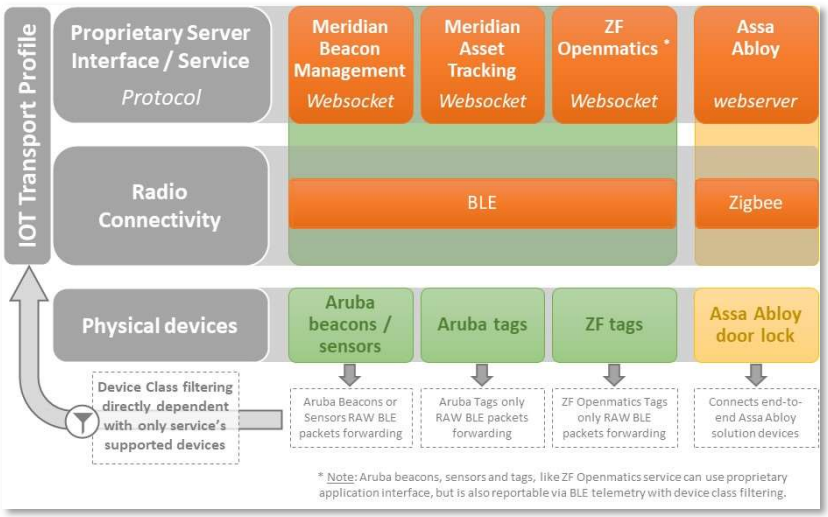


DIAGRAM 5: IOT TRANSPORT PROFILE AND SERVICES (PROPRIETARY BASED CONNECTIVITY)

IoT Transport Services details (end-to-end use case) based on 3rd party vendor solutions:

- **Meridian beacon management**: Forwards bidirectionally RAW BLE data frames to Meridian cloud application originating only from device classification: 'Aruba beacons' or 'Aruba sensors'. Covers indoor wayfinding and geofencing, as well as in store customer experience enrichment use cases. IoT gateway act as a remote gateway for managed products in the Meridian end-to-end solution. More details about the complete solution on Aruba's website.

- **Meridian asset tracking**: Same operating mode as the beacon management solution. Difference with previous proprietary service is restriction to device classification: 'Aruba tags'. Covered use case is asset tracking with active BLE tags. IoT gateway acts again as a remote gateway for managed products in the Meridian end-to-end solution. Also note that each Aruba AP embedding a BLE radio can be configured to act itself as an iBeacon. This can be configured through Meridian application suite, or simply for a "simple and isolated" usage through the AP or MC management interface with 'beaconing mode' set for the BLE operating mode in radio interface profile. When using the IAP beacon broadcasted information is fixed (BLE radio MAC Address) and not customizable. Using an MC based solution permits to edit iBeacon UUID, major number, minor number and TxPower settings. More details about the Meridian complete solution on Aruba's website.

- **ZF Openmatics**: Another RAW BLE data frame forwarding but this time from ZF Openmatics covering an asset tracking use case. Classification for this solution will be: 'ZF tags'. IoT gateway is again in a remote gateway position delivering interconnection to vendor management servers. More details about this partner solution on Aruba's website.

- **Assa Abloy**: Two differences here with listed predecessors: Used radio protocol is Zigbee, and northbound connectivity type is webserver based rather than WebSocket. Operation mode remains the same: Forwarding data from door locks ecosystem, to Assa Abloy Global Solutions Visionline management software. Classification unique value is 'Assa Abloy'. Partner solution details are on Aruba's website.

Two words about Ethernet to serial connectivity and SES Imagotag solution, which were not explicitly detailed as Transport Profile services but were however mentioned as southbound connectivity options in diagram 2 - IoT gateway components and in the supported IoT technologies paragraph.
This is not an oversight! I didn't consider those as services because it does not use the IoT application interface and its components. The first just bridges AP auxiliary Ethernet port and the hosted USB system, providing direct IP connectivity to backend management system through management port without any further processing. The second establish a direct IP tunnel between vendor specific USB dongle hardware and its management software, where data flows directly without any other action.

A few words about security:
Aruba security concepts extended to IoT.

IoT devices are simple devices, simple by their hardware parts, simple for their processing resources, but also simple when talking about cryptographic features which requires computing capacities too. That kind of device is more designed for low energy consumption, constrained selling price also affecting components choices, than to deal natively with strong authentication, security, or privacy. As the gateway is located on standard Aruba access points, IoT integration inherits and benefits from Aruba infrastructure security concepts. This convergence enhances global security from the IoT perspective.

Tunneling: When deployed in campus or remote configuration with a mobility controller, the access point tunnels all its traffic back to the controller. Stronger privacy can be achieved with tunnel encryption activation. Isolation is native as all data goes inside the tunnel, also easing traffic control.

Central enforcement point: It's the controller role in the distributed infrastructure. As it centralizes all traffic, it is the point of choice to operate filtering and segmentation tasks. Aruba mobility controllers onboard a Deep Packet Inspection (DPI) capable firewall that will then inspect and enforce IoT traffic like it does for "regular" network traffic. Micro segmentation can be used to isolate devices between others or restricting network destinations accessibility to the strict needs only. Application Visibility (AppRF) and web content classification (WebCC) will offer to IoT devices, traffic summary and visibility, URL and content filtering, IP reputation and geolocation correlation. A little word about Access Control Lists (ACL): It can be employed to filter traffic at layer 2 or layer 3 level, like it is usually. But it is also possible to create ACL enforcing USB or second ethernet port, then serial over USB or serial over Ethernet usages.

Authentication, protocol breakout and VLAN support: Each IoT transport profile is configurable in the way it directs and authenticates northbound traffic outside the gateway. Authentication done at the infrastructure level uses an onboard Trusted Platform Module (TPM) to securely store any credential, better than what a simple and trivial IoT device can do. Northbound is also responsible of authentication with remote severs. The IoT gateway then becomes a protocol breakout device as it does not directly unconditionally forward traffic coming from the IoT radio, but it acts like a proxy: Terminating the session on one hand and then opening a new authenticated one on the second, achieving protocol breakout. Logical network isolation can also be done in conjunction with DPI firewall by selection of different VLAN identifiers per profile, differentiating usages flow at transport profile level. The capability of bridging IoT traffic at layer 2 makes each one isolated from the other. When indirect Internet access is used to access cloud resources, authenticated proxy server relay can be configured.

Global infrastructure security: Concepts like device fingerprinting, dynamic segmentation, role-based policy decisions per user or per device, central application of security policies, and traffic and anomaly analytics engine are inherited from the platform acting in conjunction with the ClearPass[19] solution. Thus, enhancing the IoT environment at the infrastructure level.

Securing exchanges with "the outside world": If the IoT device can't encrypt its own traffic due to lack of processing resources, the AP or the combination of AP + MC are is to take a part of the job for the rest of the traffic path. When using secure interconnection through the IoT application gateway to interact with backend management servers, an infrastructure SSL certificate is used even if IoT devices can't encrypt anything or doesn't ensure confidentiality in exchanges nor guaranties data integrity. These security concepts perfectly fit with the three principles of the CIA triad[20], a well know security model. SSL employed at the infrastructure level fills the security hole introduced by the IoT devices' resource lack that usually can't run it. These computing capacity transfers then benefits to the whole IoT ecosystem.

Implementation examples

During testing I worked with Azure IoT Hub and an IAP directly connected to the Internet. I also tested how it worked with a mobility controller and managed APs but forget it quickly, not because it was inefficient but as it worked the same way, and preferred IAP alone that was less cumbersome.

Using cloud-based data analytics was a fast and easy way to concentrate on the access point configuration and dependencies with northbound, southbound connectivity and IoT device configuration, avoiding taking a lot of time on the server part and data acquisition coding. Azure with the free account resources proposes almost all that is needed to be quickly operational, a few

[19] Terminology and detail about ClearPass for secure network access at Aruba.
[20] Confidentiality, Integrity and Availability, the CIA triad explained on WhatIs.com.

things remains unfree, but stays very cheap. If you've got good coding talents, documentation is available and mentioned earlier in this document, it's also perfectly possible to do it all by yourself!

Another interesting tool to concentrate on the essentials without skills to operate a data analytic server and its connectivity, and even no knowledge about working with Azure IoT hub, is the IoT Utilities application available for Android devices at the Google play store. This app must be mentioned as its developer did a great job and delivered a really complete solution to test all features delivered by the IoT gateway. It's simple, you connect your Android device via Wi-Fi and use its IP address as the destination endpoint in the IoT transport profile configuration (of course it should be reachable from your access point or mobility controller). That's all: You now have a full "on-prem" IoT server in your hand, which will decode and display all flows it receives on its WebSocket. Terribly efficient, easy to use, and free. Cherry on the cake, the application comes with git documentation including configuration of your mobility controller or instant AP. Really appreciable!

Disclaimer: The following use case examples will not be detailed as a full turnkey procedure, as I want to keep this document short, light, and readable. Primary guidelines will be provided to give direction and should be completed with Aruba and eventually other vendor documentation for full coverage. All configuration related indications are based on controller or IAP software version 8.8.

Configuring telemetry with Azure IoT Hub

If you do not already have a data analytics backend with a telemetry server running WebSocket otherwise a simple webserver, you could probably be interested in this service in Azure cloud.

To do so, you should have an active Azure account and create attached to your Azure subscription, optionally in a dedicated resource group:

- A new 'IoT Hub' resource: This is the IoT application core.
- A new 'IoT Hub Device Provisioning Service' (DPS) linked to the IoT Hub: Used for AP credentials provisioning and authentication with cloud application.
- An 'Enrollment Group' attached to the DPS object: Used for symmetric key distribution to AP.

On the Aruba side create or edit an IoT transport profile, with at least following parameters:

- State: 'Enabled'.
- Type: 'Azure IoTHub'.
- Authentication:
    - ID Scope: Value found on Azure in the DPS object 'Overview' menu, 'Essentials' pane.
    - Group key: 'Primary Key' value found in the Enrollment Group properties, in the DPS object 'Manage Enrollment' menu, 'Group Name' edition, then 'Settings' pane.

Doing so will connect the Aruba IoT application interface (per controller or per IAP) with the Azure IoT Hub object. Even if no data is pushed out from gateway until a device class filter is configured, you should at least receive the 'AP Health' messages, indicating a successful connection with your

cloud based IoT application. Checking in the 'Overview' menu of the 'IoT Hub object' you should see a messages count incrementing in the 'Usage' pane. Entering the 'IoT devices' menu, should bring you the listing of authenticated and "enrolled" AP(s), identified by their MAC addresses.

Server room monitoring with EnoCean sensor

EnoCean[21] is an ecosystem of self-powered energy harvesting devices, providing sensors and actuators for various usages. Controller function is hosted in a USB dongle which can be plugged into Aruba access points, and, with serial over USB connectivity, can pass data to the management server.

Hardware: What you will need to start is an EnoCean device and the controller USB stick, it is also possible to acquire an IoT demonstration kit[22] working with a "server-side" python demo script. In my test use case, I played with EnOcean STM550 multisensor to simulate server room temperature and hygrometry monitoring and the EnOcean USB300 dongle as the radio controller.

Software: Overcoming the EnOcean demo script limitations, you should better use your own telemetry server or Azure IoT Hub. Telemetry over WebSocket flow from the configured access point contains serial over USB encapsulated payload transmitted from the USB dongle. Data first needs to be deserialized as WebSocket uses Protobuf for the data format and the resulting EnOcean ESP3 serial protocol[23] payload must be decoded before being human readable or binary usable. There is also a need to deal with the logical association between the EnOcean device and the dongle. Documentation about protocol buffers serialization or EnOcean ESP3 is available, if you have time and coding skills, it could be an option. If not, EnOcean Alliance provides EnOcean IoT Connector[24] a group of docker containers intended for device onboarding via an API and processing of the EnOcean device flow to decode and output it in JSON format. The tool isn't free for industrial deployments, but a limited trial version license can be requested. It is available via DockerHub, or directly in Microsoft Azure Marketplace to integrate with Azure IoT cloud applications.

Testbed configuration guidelines:

- Deploy EnOcean IoT Connector: On premise, private cloud, or in Azure IoT.
- If needed don't forget to install your own PKI certificate on IAP or mobility controller, as it is used to secure the WebSocket exchanges with servers.
- Create or edit an IoT transport profile, with at least following parameters:
  - State: 'Enabled'.
  - Type: 'Telemetry WebSocket' or 'Azure IoTHub'.
  - Server URL: If using 'Telemetry WebSocket', must provide the complete wss:// URL pointing to the IoT Connector server. Not needed when running it in Azure.

[21] More information about EnOcean at EnOcean Alliance. | Partnership between EnOcean and Aruba at EnOcean Alliance.
[22] EnOcean starter kit at EnOcean Alliance. | EnOcean Aruba IoT demo.
[23] EnOcean ESP3 serial protocol specification from EnOcean Alliance.
[24] EnOcean IoT Connector presentation at EnOcean Alliance.

  - Device Class: ‘Serial Data’.
  - Authentication: WebSocket Server dependent. 'Group symmetric key’ for Azure.
- Configure IoT Connector to redirect JSON data flow to your final data acquisition server.
- Use the IoT Connector API to onboard your device(s).

Bluetooth survey with BLE Telemetry and devices RAW data forwarding

The AP embedded BLE radio is used to report operating BLE devices in the monitored environment.

Hardware: AP embedded BLE radio reports discovered end devices, no more minimal requirement. You should have one or more BLE devices in the test vicinity to gather data, if you do not have physical beaconing devices, it’s possible to emulate iBeacon or Eddystone via software.

Software: A telemetry receiver, that could be in Azure IoT cloud, or your own WebSocket server. It’s also possible to rely on a simple webserver but BLE Connect and BLE RAW data forwarding will not be accessible, thus limiting usage to telemetry reports only. A simple and efficient choice, that is also easy and fast to run, if you’re not easy with Azure cloud solution and doesn’t already have an operating WebSocket data acquiring server, is to use IoT Utilities application. You will be able to test all BLE services functionalities with an Android device running the WebSocket server and the application interface to view reported data or access remote devices (device and profile dependent). The application can also emulate iBeacon devices with configurable parameters (UUID, Minor, Major, and Signal power). You can use any other beacon emulator with as many different physical devices you need, like beacon simulator available for Android platforms, which can emulate iBeacon but also Eddystone with its 4 different frame types, all configurable (UID, signal, URL, TLM values, EID key…).

Testbed configuration guidelines:

- Install and configure IoT Utilities application on an Android device and connect it via Wi-Fi to your infrastructure. It should be layer 3 reachable from the AP or MC management interface.
- If you’re using a private PKI, deploy certificates on both AP / MC and IoT Utilities app server. If not, IoT Utilities and AP / MC each have a self-signed certificate preinstalled. You’ll need to export server’s app public certificate and place it in the trusted CA store infrastructure side.
- Create an IoT radio profile to activate internal radio for BLE usage, with at least:
  - State: ‘Enabled’.
  - Radio mode: ‘BLE’.
  - Operational mode: ‘Analysis’ or ‘Both’ (if also using AP as a beacon transmitter).
  - Console: ‘Auto’.
- Create or edit an IoT transport profile, with at least following parameters:
  - State: ‘Enabled’.
  - Type: ‘Telemetry WebSocket’ or ‘Azure IoTHub’ for full functionalities, ‘Telemetry Https’ for plain text reporting only.

- Server URL: If using 'Telemetry WebSocket', provide the complete wss:// URL pointing to IoT Utilities server (ex: wss://<server IP>:5443/wss). It is not needed with Azure.
- Device Class: 'iBeacon' and / or 'Eddystone' for beaconing purposes, any other specific device class corresponding to your own devices, or 'All' (with or without additional filtering) and 'Unclassified' to catch all devices. Note: Eddystone will only report RAW data, iBeacon will be viewable in telemetry flow and RAW data. If 'All' isn't selected, only RAW data will be viewable as no telemetry will be reported.
- BLE data forwarding: If server type is not 'Telemetry Https', activate this slider for immediate BLE RAW data forwarding in addition to telemetry periodic reports.
- Authentication: WebSocket server dependent. 'Group symmetric key' for Azure. Example configuration for use with IoT Utilities authentication API:
  - Authentication: Client credentials
  - URL: https://<server IP>:5443/auth
  - Client secret: Configured when installing the application, then manageable afterwards in application settings, 'IoT Server' menu, 'Server security / authentication' server group and finally 'Authentication username' and Authentication password' menu items.
  - ID client: What you want, which will identify IAP or MC in reporting data.

Results will be viewable in the IoT Utilities application main dashboard under 'BLE telemetry data records' and 'BLE data records' with advanced details by clicking each item. Analytics are also accessible via the main menu and 'IoT Data' group. Main menu 'BLE Connect' option in 'IoT Server' group is the place to go for devices recognized as connectable. 'BLE Testing' option at same place, is where you can configure the app to act as in iBeacon device.

Wi-Fi devices reporting with Wi-Fi Telemetry

AP Wi-Fi can be used to report analytics about Wi-Fi devices in the area.

Hardware: AP embedded Wi-Fi radio reports discovered end devices, no more minimal requirement.

Software: As it was with BLE telemetry, you need to use a data acquiring server to receive telemetry flow. Wi-Fi telemetry is mainly reporting, excepted for specific RTLS functions if you're already equipped with a dedicated management and tracking solution, so an HTTPS webserver will mostly be sufficient. Reporting will also be available with WebSocket, that kind of enabled server will be mandatory is using advanced RTLS functions. Like it was advised in the former section, the IoT Utilities application is a good and quick solution to easily view reported data, if you do not have already have a server available on premise, in Azure or in your private cloud.

Testbed configuration guidelines:

- Install and configure IoT Utilities application on an Android device and connect it via Wi-Fi to your infrastructure. It should be layer 3 reachable from the AP or MC management interface.
- If you're using a private PKI, deploy certificates on both AP / MC and IoT Utilities app server. If not, IoT Utilities and AP / MC each have a self-signed certificate preinstalled. You'll need to export server's app public certificate and place it in the trusted CA store infrastructure side.
- Create or edit an IoT transport profile, with at least following parameters:
    - State: 'Enabled'.
    - Type: 'Telemetry WebSocket' or 'Azure IoTHub' for full RTLS functionalities, 'Telemetry Https' for plain text reporting only (including RTLS devices).
    - Server URL: If using 'Telemetry WebSocket', provide the complete wss:// URL pointing to IoT Utilities server (ex: wss://<server IP>:5443/wss). It is not needed with Azure. If using 'Telemetry Https', enter webserver service full URL starting with https.
    - Device Class: 'WiFi RTLS Tags' (if used), 'WiFi Associated Stations' and 'WiFi Unassociated Stations'. All other device classes excepted 'Serial Data' and 'Zigbee Socket Devices' are for BLE purposes. Note: Previous BLE telemetry use case is pretty similar in its configuration and can be configured concurrently with actual one.
    - BLE data forwarding: Not needed for Wi-Fi only use case, could be if merged with BLE.
    - Authentication: WebSocket server or webserver dependent. 'Group symmetric key' for Azure. Example configuration for use with IoT Utilities authentication API:
        - Authentication: Client credentials
        - URL: https://<server IP>:5443/auth
        - Client secret: Configured when installing the application, then manageable afterwards in application settings, 'IoT Server' menu, 'Server security / authentication' server group and finally 'Authentication username' and Authentication password' menu items.
        - ID client: What you want, which will identify IAP or MC in reporting data.

Results will be accessible in the IoT Utilities application main dashboard under 'Wi-Fi telemetry data records' with advanced details by clicking on item. Analytics are also accessible via the main menu and 'IoT Data' group. If used in conjunction with BLE telemetry refer to previous section for report details.

AP as a BLE beacon

Embedded BLE radio can be configured for environment analysis, but also or only for beaconing. This can be done via dedicated solutions like Meridian or ZF Openmatics which will interact with BLE console, or simply by command line or GUI to only enable the function without further interaction. Standalone configuration can seem a useless idea, but can present some advantages if you're running a dedicated solution like home automation system where an end device will react

and do some action when detecting the beacon (I'm at home then… or I'm leaving home so do…) or if using an indoor geolocation application of your own, based on devices decisions only and no interaction with central management or location server (due to GDPR regulation constraints as an example).

Hardware: Nothing more than an Gen2 radio inside or outside an AP.

Software: Nothing special to activate the AP for beaconing purpose. Anything else that could be necessary in relation with end usage (ex: Specific geolocation application).

Configuration guidelines:

- No IoT profile configuration required here!
- Just configure BLE radio for beaconing by an IoT radio profile with BLE usage creation, with:
  - State: 'Enabled'.
  - Radio mode: 'BLE'.
  - Operational mode: 'Beaconing' or 'Both' (if also using another BLE analysis function).
  - Console: 'Auto'.
- If using mobility controller managed AP(s), beacon parameters can optionally be configured. Those are static with IAP.
  - Issue the following command line at the controller's privileged prompt (mdconnect from mobility master if concerned): **ap ble-configure ap-name** *<AP Name>* **cfg-ble-mac** *<IoT radio MAC address>* **uuid** *<IBeacon UUID hexadecimal string>* **major** *<16 bits IBeacon major number>* **minor** *<16 bits IBeacon minor number>* **txpower** *<0-15 txpower range>*
  - Dedicated IoT radio MAC address is obtained with this command: **show ap debug ble-table ap-name** *<AP Name>* **all**
    Note: In fact, it will list all BLE devices including at least the AP dedicated radio, but this command is also useful to check beacon additional parameters actually configured.

Zigbee environment sniffer

Zigbee radio scan and relay frames to a remote location server for analysis, using GRE encapsulation.

Hardware: An AP with Gen2 IoT radio, no more minimal requirement.

Software: A packet analyzer recognizing Zigbee protocol on the remote side.

Configuration guidelines:

- Issue the following command line at the IAP or controller privileged prompt (mdconnect from mobility master if concerned): **iot-sniffer radio** *<IoT radio MAC address>* **enable zigbee**
- **Then start frames transfer with: iot-sniffer radio** *<IoT radio MAC address>* **start** *<Remote IP address> <optional: Remote port>* **zigbee-channel** *<channel number between 11 - 26>*

- Dedicated IoT radio MAC address is obtained with this command: **show ap debug ble-table ap-name** <*AP Name*> **all**
  Note: It will list all BLE devices including at least the AP dedicated IoT radio, curious as we are talking about Zigbee now! Remember IoT radio is shared between BLE and Zigbee usages, it's the same physical radio module, with a unique MAC address. A similar command exists for Zigbee radio but requires creating an IoT radio profile enabling Zigbee first, that is not a prerequisite for the sniffing functionality. If the radio profile enabling Zigbee already exists, **show ap debug ZigBee radio-table** will list the Zigbee Coordinator (ZC) configuration including IEEE "Zigbee" Address. Its size is 64 bits, and it's constructed using the 48 bits physical radio MAC address borrowed in its middle by FF:FE, to obtain the IEEE Address used by Zigbee.
  Just remove the 16 bits with fixed pattern in the middle, and you're with the MAC address.