API GUIDE

8.7.1.X ARUBA IOT INTERFACE

Aruba WLAN IoT Interface Documentation

aruba
a Hewlett Packard Enterprise company

CHANGELIST

Version	Date	Notes
1.6	03/26/2021	Updated for AOS 8.7.1.3
1.5	09/25/2020	Up version for AOS 8.7.1.0. Also updating serial data information section
1.4	09/15/2020	Fix formatting in preparation for document release
1.4	09/12/2020	Added gattIndication documentation for Southbound API
1.3	07/28/2020	Added Zigbee Socket Data section with examples
1.2	06/11/2020	Added WiFi telemetry, WiFi RTLS and Serial data sections with examples
1.1		Revamp the document in preparation for the AOS 8.7.0.0 release
1.0	06/01/2020	Official release for AOS 8.6.0.0 on asp.arubanetworks.com
0.61	01/21/2020	• Updated the Southbound API fields for Service UUID and Characteristic UUID to be bytes, not strings
0.6	12/04/2019	• Added information about the Token expiration mechanism • Added some information about Northbound Reporting • Added more detailed information about specific fields in the IoT transport-profile
0.5	11/15/2019	• Changed Document to overall IoT Telemetry Interface Documentation • Updated with final 8.6 release information
0.3	07/01/2019	• Changed definition of responses to gattRead • Added information about characteristic discovery • Added clarification about gattNotification subscription
0.2	04/01/2019	Formatting changes, and addition of example responses for every action
0.1	03/27/2019	Initial Document Revision

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett Packard Enterprise Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett Packard Enterprise Company 6280 America Center Drive
San Jose, CA 95002
USA

CONTENTS

1. Introduction

Aruba WLAN provides a public Internet of Things (IoT) interface to partner applications like location services, IoT device management applications, etc. Customers can configure the IoT interface by creating a transport profile on the Controller or Instant AP. The "IoT transport profile" is highly customizable to serve a diverse set of applications. The IoT interface sets up a permanent connection between the Aruba WLAN and the partner application. This connection can then be used to transport IoT related data, such as telemetry reports, commands or IoT data packets.

a. Solution Overview

The following depicts an example of a system configuration that could be setup using the Aruba Telemetry feed.

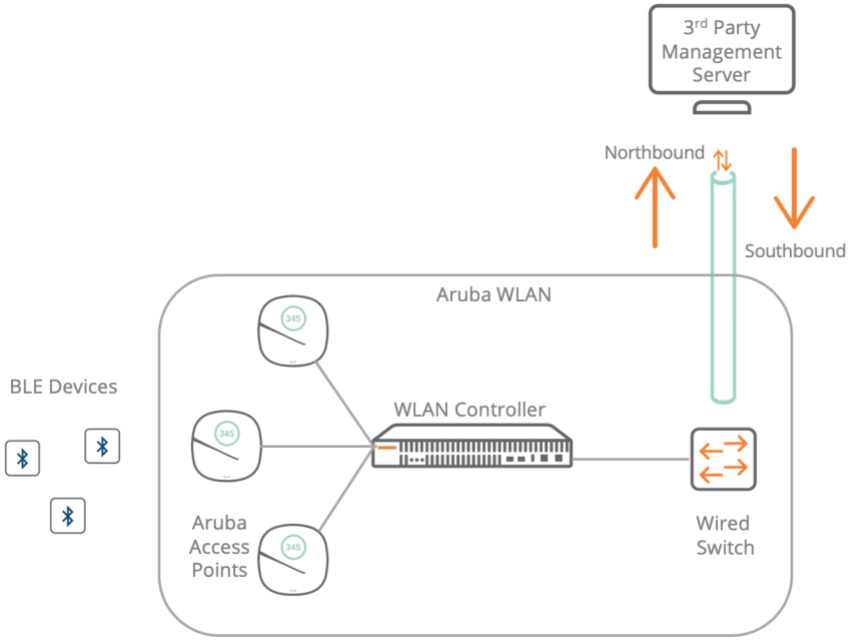

Figure 1: Typical Aruba WLAN Deployment with IoT Interface using the Telemetry WebSocket server type.

To simplify terminology going forward, we first need to define some phrases that will be used often in this document:

- Access Point (AP) – The Aruba AP contains the IoT radio. The communication with BLE devices will be referred to as communication between the remote device and the AP.
- ArubaOS (AOS) – This is the software that runs on Aruba WLAN infrastructure (mobility controllers, APs, Instant APs), containing the Aruba code for the IoT transport profiles
- Northbound – This is the direction from Aruba WLAN infrastructure, to the partner application.
- Southbound – This is the direction from the partner application, to the Aruba WLAN infrastructure.
- Transport Profile – This is the name for the profile that the user will define on the WLAN management console to configure the transport between Aruba WLAN and the partner application.

b. Transport Services

The transport-profile allows configuration for different transport methods. However, this document focuses mainly on one method, the "Aruba Telemetry WebSocket" (ATW) method. As suggested by the name, data is transported over a WebSocket connection. The payload inside the WebSocket is formatted using Google Protocol Buffers (Protobuf). Proto2 is used.

Once the IoT interface connection is established it can transport the data for various IoT services. The services are listed in Table 1. Each service is described in more details in subsequent chapters.

Service	Direction	Purpose
BLE Telemetry	Northbound	Periodic telemetry reports with structured data from each device
BLE Data	Northbound	BLE advertisement frames and Scan-Response frames. Entire frames plus metadata
BLE Connections	Bi-directional	Commands and responses to use connection-based services over BLE.
Wi-Fi Telemetry	Northbound	Periodic reports about nearby Wi-Fi clients
Wi-Fi RTLS Data	Northbound	Wi-Fi RTLS packet forwarding
Zigbee Sockets Data	Bi-directional	Zigbee data frames to/from Zigbee socket devices
Serial Data	Bi-directional	Data frames to/from USB devices plugged into the AP

Table 1: IoT Services

2. Configuration

The IoT interface and the IoT services are configured by setting up an IoT transport profile. Here are the fields in a transport profile:

Category	Name	Description
Interface Type	Server Type	The type of server that is consuming the IoT Interface
Frequency	Reporting Interval	Reporting interval in seconds
Authentication	Authentication URL	Server URL for authentication
	Username	Username for authentication
	Password	Password for authentication
	Client ID	This ID identifies the sender to the server
	Access Token	Access token. Configure this only if you want to bypass authentication
Destination	Server URL	Server URL for sending telemetry
	Proxy	Proxy server for sending telemetry
Device Filters	Device Class Filter	A list of device class tags to filter the devices included in the reports
	UUID Filter	A list of UUIDs to filter the devices included in the reports. Applies only to iBeacon devices
	UID Namespace Filter	A list of UID namespaces to filter devices included in the reports. Applies only Eddystone-UID devices
	URL Filter	A list of URL strings to filter devices included in the reports. Applies only Eddystone-URL devices. The string listed here can be partial URL strings
	Cell Size Filter	A proximity filter. Devices outside the cell will not be reported. Size is specified in meters. Setting to 0 disables the cell size filter
	Movement Filter	Filters devices that do not change distance. Specified in meters. Applicable only if a cell size is set. Setting to 0 disables the movement filter
	Age Filter	Age filter. Devices without recent activity will not be reported
	Vendor Filter	A list of Vendor IDs or Vendor Names which are used to filter reporting
	ZSD filter	A set of Zigbee Socket Devices to filter. This applies only to devices that conform to the ZSD device class
	RTLS Dest. MAC	Sets the destination MAC address filter for RTLS tags device class.
Content specifiers	RSSI reporting format	Set the preferred format for RSSI reporting
	Environment type	The type of environment that the APs are deployed in. The environment determines the RF fading factor that is used for the translation from RSSI to distance
	Custom Fading Factor	For manually setting the fading factor. Applies only when Environment Type is set to Custom
	Device Count Only	For those interested in a count of devices seen, but not the actual content of those devices
	Data Filter	This is a mechanism to suppress particular fields in the telemetry reports, that are not required by the receiver

Table 2: IoT Transport Profile Configuration Parameters

While most of the above fields are self-explanatory, some require more explanation than what is describe above. Here is a deeper explanation of some of the fields above.

Cell Size Filter

A proximity-based filter that will only report devices that are found to be within an "x" meter radius around the access point. This distance is calculated with an algorithm based off the RSSI value. The default value for this field is "0", which translated to the cell size filter being disabled. This field accepts integer values from 2 to 100, and the units are meters.

Movement Filter

This filter is only active when the cell size filter is also configured. When this filter is enabled, devices will only be reported if they have been calculated to have moved more than the configured value for this filter in meters. For example, if the movement filter

is configured to be 2 meters, a device that is calculated to have moved 1 meter will not be reported, while a device that moves 5 meters will be reported. The default value for this field is "0", which corresponds to the movement filter being disabled. This field accepts integer values from 2 to 30, and the units are meters.

Age Filter

The Age Filter is used to only report devices in which we have received an update (either BLE advertisement or scan response) in the configured time. For instance, if the age filter is set to 30 seconds, only devices which have been heard in the last 30 seconds will be reported. If there is a device that received an update 45 seconds before, this device will not be reported. The default value for this field is "0", which corresponds to the age filter being disabled. This field accepts integer values from 30 to 3600, and the units are seconds.

Vendor Filter

The Vendor Filter allows a subscriber to input either Bluetooth SIG Vendor IDs, or freeform Vendor Name strings, which will be used to filter the devices reported. If this is configured, the only devices that will be reported are the devices that match the configured Vendor ID or Vendor Name.

RSSI Reporting Format

We currently support five different RSSI reporting formats when sending reports to subscribers. The four reporting formats are:

- Last:
  - Only the last RSSI value that was seen by the device will be reported.
- Average:
  - The average RSSI over the reporting interval will be reported.
- Max:
  - The max RSSI value that was seen over the reporting interval will be reported only. This max value resets each telemetry reporting interval, and will be updated accordingly
- Bulk:
  - The last 20 RSSI values that were seen by the device since the previous telemetry report will be reported in an array format
- Smooth:
  - A single smoothed out RSSI value will be reported for each telemetry report. This is done by attempting to remove outliers from the RSSI values received by the AP

Environment Type

We currently support five different pre-defined environment types to help adjust RSSI based distance values to better fit the environment in which the BLE devices are operating in. For best results, you should choose the value that closest corresponds to the environment in which BLE is operating.

- Auditorium:
- Office:
- Outdoor:
- Shipboard:
- Warehouse:

Custom Fading Factor

If the above environment type offsets do not properly fit your environment, a custom fading factor can be configured which is a custom environment type. This value will only be considered if "Environment Type" is configured to custom. This field accepts integer values in the range of 10 to 40.

Data Filter

This is a list of fields to suppress in the telemetry reports. The data filter is a string that is a comma separated list of index-paths. Each index path refers to the protobuf field numbers. For example, the value “3.3, 3.12” would suppress the ‘reported.model’ field and the ‘reported.beacons’ field in the telemetry reports.

3. Authentication and Authorization

For the IoT transport profile, authentication is optional. If authentication is desired the user needs to configure an authentication URL, username and password in the IoT transport profile. If Authentication is not required, the authentication URL, username and password should be left empty, and the user needs to provide an access token.

Authorization is expressed using an access token that is present in every message sent to the server. In an authenticated connection, the access token is obtained during authentication. In an unauthenticated connection, the access token must be configured manually in the IoT transport profile.

a. Authentication Handshake

Authentication is always done using HTTPS, subsequent API calls are done using secure WebSockets.

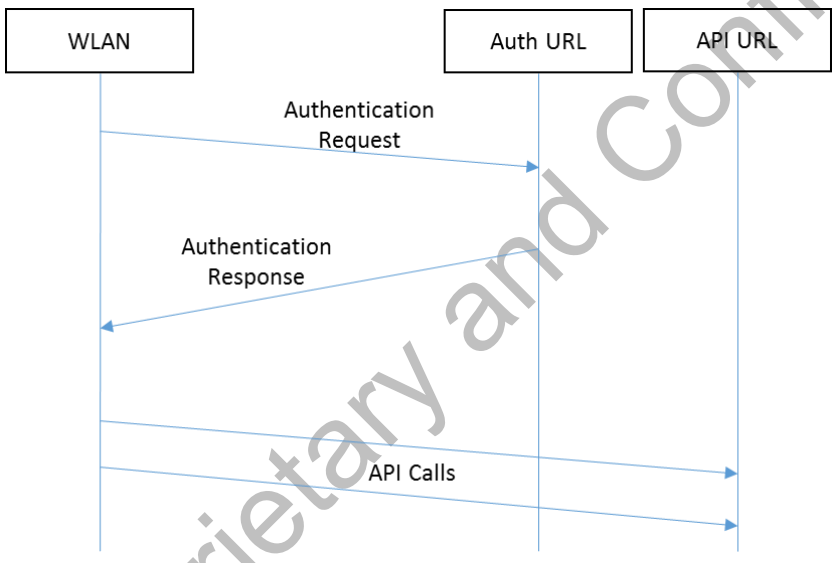

Figure 1: Authentication Sequence

b. Authentication Request

This is an HTTPS POST operation. The body contains the following JSON content:

```
{
    "grant_type": "password",
    "username": <username>,
    "password": <password>,
    "client_id": <ClientID>,
}
```

- "grant_type" is always set to "password"
- The "user_name", "password", and "client_id" fields are taken verbatim from the IoT transport profile
- If there is no client ID configured in the IoT transport profile, then the "client_id" field will be omitted from the JSON in the POST body.

c. Authentication Response

For successful authentication, we look for the following content in the response body:

```
{
    "access_token": <access_token>,
    "token_type": "bearer",
    "refresh_token": <refresh_token>,
    "expires_in": <time>,
    "api url": <API URL>,
}
```

- "access_token" is mandatory
- "token_type" is optional, but if it is present, the value must be set to "bearer"
- "refresh_token" is optional, and will be used for token refresh if present instead of a full re-authentication
- "expires_in" is optional, but if it is not present, we assume that the token is valid until we receive an error
- "api_url" is optional. If it is present, we will use this value rather than the serverURL specified in the IoT transport profile

For Aruba Telemetry HTTPS (ATH), we will use HTTPS POST exclusively. In every call, the access token is included in the authentication header as follows:

```
"Authorization: Bearer <ACCESS_TOKEN>"
```

We currently look for the following errors for authentication:

Error Code	Error Reason	Action
401	Unauthorized	Upon reception of this error, the system will try to authenticate again

Table 2: HTTPS Error Codes Used

For Aruba Telemetry WebSocket (ATW), the access token will be included in the payload of every message. Please see the telemetry proto files for the exact definition. In the return messages, the server can include a status message to indicate if the

token was invalid.

d. Token Expiration and Token Refresh

During the authentication handshake, if we receive the “expires_in” field along with the access token, we will support token expiration. As soon as the authentication handshake happens, we will store the “expires_in” time, and after that amount of time, we will consider the access token invalid. We currently support a minimum “expires_in” time for 300s, and a maximum time of 1 month. At this point we have two options, either a full re-authentication where we redo an authentication handshake and restart the connection, or if the server provided a refresh token, we will attempt to refresh the access token using the refresh token. If a refresh token is provided, instead of immediately tearing down the connection, we will first attempt to get a fresh access token from the server. This refresh will use a grant type of “refresh_token” instead of a grant type of “password”. If we are unable to get a new access token, or the server returns a 401 Error Code, we will fall back to tearing down the connection and redoing the full authentication handshake.

4. AP Health Information

Once the WebSocket connection is set up, each AP will send an AP Health message to the server every 120s. The message contains information about the reporter AP, the onboard radio, and any other supported external USB serial devices.

Note: AP Health messages were introduced in AOS 8.7.1.3. Prior versions of AOS do not send this message.

Sample AP Health Update Message


```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "apHealthUpdate"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.7.1.3-8.7.1.3",
    "swBuild": "79649",
    "time": "1617055073"
  },
  "apHealth": {
    "apStatus": "healthy",
    "radio": [
      {
        "mac": "204c0339e22c",
        "hardware": "gen2",
        "firmware": "arubaDefault",
        "health": "healthy",
        "external": false
      }
    ],
    "usb": [
      {
        "identifier": "ENOCEAN_USB",
        "health": "healthy"
      }
    ]
  }
}
```

5. BLE Telemetry

The BLE telemetry service sends periodic reports about all the BLE devices that are discovered by an AP. The AP will continually listen for advertisements and scan responses. The AP will parse/decode these packets to the best of its abilities and update the telemetry in its internal table. Periodically, the contents of this table will be reported as BLE telemetry data. Once an IoT transport profile is properly configured on the AP/Controller, northbound telemetry messages will start to be sent. This will continue indefinitely until the profile is removed. These telemetry reports contain a summary of all the BLE devices that are seen by a particular AP. For each individual BLE device, we only populate the information that we have for the device. An example of these reports can be seen in the referenced JSON example file.

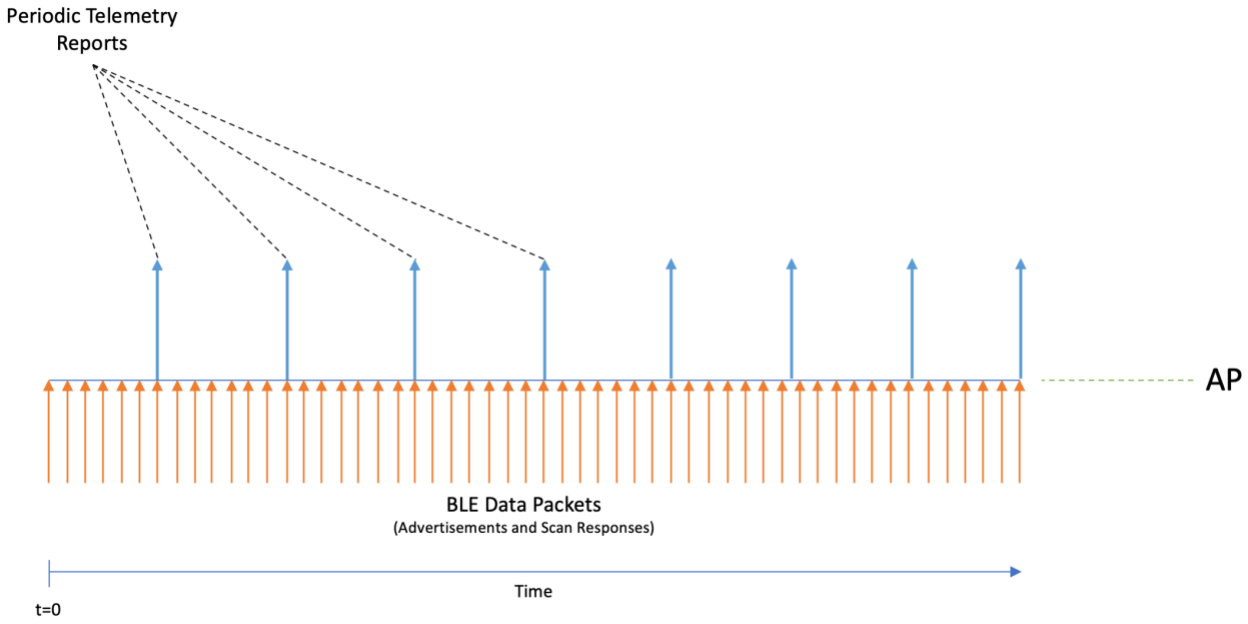

Figure 3: Example of an IoT transport profile with periodic telemetry reports

Sample Telemetry Message


```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "telemetry"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.7.1.3-8.7.1.3",
    "swBuild": "79649",
    "time": "1617067469"
  },
  "reported": [
    {
      "mac": "6ceceb403a93",
      "deviceClass": [
```

```
        "iBeacon",
        "arubaBeacon"
      ],
      "model": "LS-BT1",
      "firmware": {
        "bankA": "0.0-0",
        "bankB": "1.1-4"
      },
      "lastSeen": "1617067466",
      "bevent": {
        "event": "update"
      },
      "rssi": {
        "avg": -52
      },
      "beacons": [
        {
          "ibeacon": {
            "uuid": "4152554ef99b4a3b86d0947070693a78",
            "major": 0,
            "minor": 0,
            "power": -61
          }
        }
      ],
      "txpower": 14,
      "sensors": {
        "battery": 100
      },
      "stats": {
        "uptime": "1729470",
        "frame_cnt": 19
      },
      "vendorName": "Aruba"
    },
    {
      "mac": "cc78aba45183",
      "deviceClass": [
        "arubaTag"
      ],
      "firmware": {
        "bankA": "1.2-15"
      },
      "assetId": "0000-0000-0000",
      "lastSeen": "1617067466",
      "bevent": {
        "event": "update"
      },
      "rssi": {
        "avg": -43
      },
      "sensors": {
        "battery": 79
      },
      "stats": {
        "uptime": "63990",
        "frame_cnt": 3
      },
      "vendorName": "Aruba"
    }
  ]
}
```

6. BLE Data Forwarding

The BLE data service forwards all BLE advertisement frames and scan response frames from certain BLE devices. No special configuration is needed to enable BLE data forwarding. The BLE data service is automatically enabled for selected device classes, namely, mysphera, abilitySmartSensor, sBeacon, exposureNotification and wiliot.

BLE Data forwarding works by forwarding the raw BLE data packets to the subscriber immediately when they are received by the AP. These messages increase the traffic over the WebSocket, as you will receive a message for every BLE advertisement and scan response eligible devices send.

It is also important to remember that BLE data forwarding happens in addition to the periodic telemetry reporting. The two happen in parallel. If BLE data forwarding is the main method for which a subscriber would like to receive data, a high "reporting interval" can be configured in the IoT transport profile.

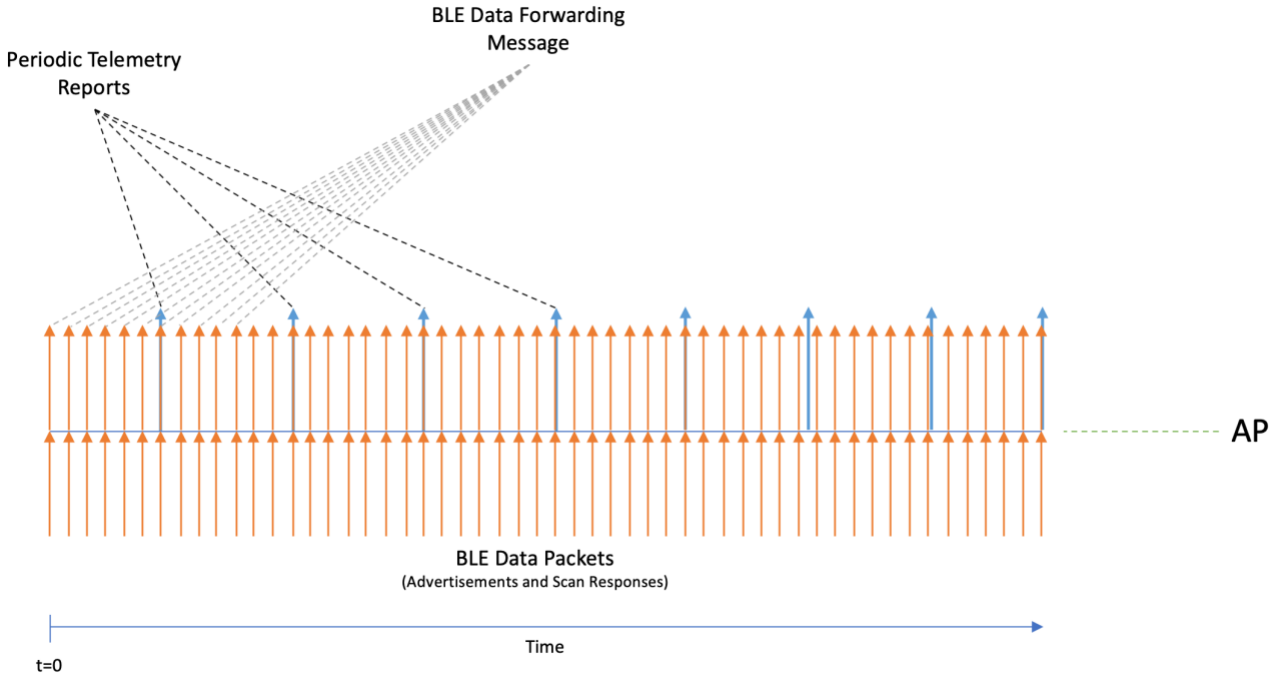

Figure 4: Example of an IoT transport profile with BLE data forwarding

7. BLE Connections

The BLE connections service provides primitives to connect and interact with BLE devices remotely via the IoT interface. This allows our partners to reach out and manage their devices via the Aruba WLAN infrastructure. This service is generic to all BLE devices. The operations map closely to the BLE GATT protocol.

Primitive	Description
Connect	Scan for a BLE device, set up a connection and discover characteristics
Disconnect	Disconnect an active connection
Read	Read from a BLE device using the GATT protocol
Write	Write to a BLE device using the GATT protocol
Notifications	Subscribe for notifications from a BLE device
Indications	Subscribe for indications from a BLE device

a. Encoding

The commands and responses for the BLE connections service are all messages going up and down the IoT Interface WebSocket. The messages are encoded using the Google Protocol Buffer method. We use the proto2 version of the protocol. The definitions can be found in the IoT Interface Protobuf Specification on the ASP portal.

When it comes to the .proto definitions, you will notice all fields are listed as optional. This is by design to increase the forward compatibility of the API definitions in the .proto file. While this is the case, each API call has some fields that must be supplied to properly process the specified operation. These will be defined for each operation.

Southbound Action Message Fields Explained

An overview of the fields in a Southbound Action message are in the chart below. This is a superset of all the fields that might be present in a southbound action. Not all the fields should be present for every command.

Field	Value	Description
meta		
version	uint64	Version of .proto definition. Currently only supported version is "1"
sbTopic	SbTopic Enum	Enum value as defined in aruba-iot-types.proto file
receiver		
apMac	MAC Address	MAC Address of AP which will process the action
actions		
actionId	string	3rd Party Server defined string to correlate responses to actions
type	ActionType Enum	Enum value as defined in aruba-iot-types.proto file
deviceMac	MAC Address	MAC Address of remote BLE device
serviceUuid	bytes	String containing either 16bit or 128bit UUID for GATT Service
characteristicUuid	byes	String containing either 16bit or 128bit UUID for GATT Characteristic

timeOut	uint32	Timeout value in seconds for the action to be completed
value	bytes	Data in a byte format to be written to characteristic in write commands
status		
connectCode	ConnectCode Enum	Enum value defined in aruba-iot-sb-status.proto
connectDescription	string	The server response description for the connection code

Northbound Action Status Message Fields Explained

After specific actions have been completed, status messages will be returned to the 3rd party server. These messages will differ based on the type of action that was completed. For each action type, the expected response will be described. An overview of the different fields that can be present in a response follow.

Note: Only fields that pertain specifically to BLE connections are explained here.

Field	Value	Description
meta		
version	uint64	Version of .proto definition. Currently only supported version is "1"
accessToken	string	Access Token present in all northbound frames which the server must verify
nbTopic	NbTopic Enum	Enum value as defined in aruba-iot-types.proto file
reporter		
name, mac, etc.	Varies	Information about the AP that processed the action will be listed here
actionResults		
actionId	string	3rd Party Server defined string to correlate responses to actions
type	ActionType Enum	Enum value as defined in aruba-iot-types.proto file
deviceMac	MAC Address	MAC Address of remote BLE device
status	ActionStatus Enum	Enum value as defined in aruba-iot-nb-action-results.proto file
statusString	string	Optional additional freeform information
characteristics		
deviceMac	MAC Address	MAC Address of remote BLE device
serviceUuid	bytes	String containing either 16bit or 128bit UUID for GATT Service
characteristicUuid	bytes	String containing either 16bit or 128bit UUID for GATT Characteristic
value	bytes	Value populated after read commands or from notifications
description	string	GATT Characteristic description
Properties	CharProperty Enum	Enum value as defined in aruba-iot-nb-characteristics.proto file

status		
deviceMac	MAC Address	MAC Address of remote BLE Device
status	StatusValue Enum	Emun value as defined in aruba-iot-nb-status.proto
statusString	string	Additional freeform information string

b. Command Overview

i. bleConnect

The bleConnect command will send an operation for a specific AP to attempt to connect to a specified BLE device. There are no prerequisites to this command.

Request Required Parameters	• ActionId • Type • Device MAC
Request Optional Parameters	• Timeout
Possible Responses	• success • connectionTimeout • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0001",
            "type": "bleConnect",
            "deviceMac": "11:22:33:44:55:66",
            "timeOut": 30
        }
    ]
}
```

Example Response

```
{
    "meta":
        {
```

```
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0001",
            "type": "bleConnect",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "success",
            "statusString": "Connection Established!"
        }
    ]
}
```

Service and Characteristic Discovery

In addition to connecting to a device, the bleConnect command also triggers a full GATT service and characteristic discovery. This will happen without user intervention. Once this action is completed, the full list of characteristics will be forwarded back to the partner application in the "characteristics" northbound topic. This is the indication that the full service and characteristic discovery has been completed. An example of this response with two characteristics is provided below.

Example Response

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "characteristics"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "characteristics": [
        {
            "deviceMac": "ff:e1:68:69:8e:57",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF"
        },
        {
            "deviceMac": "ff:e1:68:69:8e:57",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE152-6C6C-4718-A3D4-6DE8A3735CFF"
        },
    ]
}
```

ii. bleDisconnect

The bleDisconnect command will terminate the connection between the specified AP, and a specific remote BLE device.

Request Required Parameters	• ActionId • Type • Device MAC
Request Optional Parameters	• Timeout
Possible Responses	• success • notCurrentlyInConnection • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0002",
            "type": "bleDisconnect",
            "deviceMac": "11:22:33:44:55:66",
            "timeOut": 30
        }
    ]
}
```

Example Response

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
```

```
            "actionId": "0002",
            "type": "bleDisconnect",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "success",
            "statusString": "Device Disconnected"
        }
    ]
}
```

iii. gattRead

The gattRead command must only be called after a connection between an AP and remote BLE device has been established. This command will read the value of the specified GATT characteristic on the remote device.

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID
Request Optional Parameters	• Timeout
Possible Responses	• success (in form of characteristics value – will be explained in response section) • notCurrentlyInConnection • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0003",
            "type": "gattRead",
            "deviceMac": "11:22:33:44:55:66",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "timeOut": 30
        }
    ]
}
```

gattRead Responses

The response to gattRead differs slightly from the pervious commands, as it can come in two different forms. The 3rd party server will always get a response in the form of the northbound topic "actionResults", stating either if the operation was successful, or if an error occurred. If the operation was successful, you will also get a message in the "characteristics" topic,

with an updated characteristic value, showing the value received from the gattRead. An example of the “characteristics” and “actionResults” topic is below.

gattRead Response #1: Success

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0003",
            "type": "gattRead",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": “success”,
        }
    ]
}


{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "characteristics"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "characteristics": [
        {
            "deviceMac": "ff:e1:68:69:8e:57",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "value": "0102"
        }
    ]
}
```

gattRead Response #2: Failure

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
```

```
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0003",
            "type": "gattRead",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "notInConnection",
            "statusString": "Currently not connected to this device."
        }
    ]
}
```

iv. gattWrite

The gattWrite command must only be called after a connection between an AP and remote BLE device has been established. This command will perform a GATT write *without* response to the specified characteristic, with the specified value. It is the 3rd party server's responsibility to know the capabilities of the characteristic.

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID • Value
Request Optional Parameters	• Timeout
Possible Responses	• success • notCurrentlyInConnection • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
```

```
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0004",
            "type": "gattWrite",
            "deviceMac": "11:22:33:44:55:66",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "value": 0F 0F,
            "timeOut": 30
        }
    ]
}
```

Example Response

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0004",
            "type": "gattWrite",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "success",
            "statusString": "Value written successfully."
        }
    ]
}
```

v. gattWriteWithResponse

The gattWriteWithResponse command must only be called after a connection between an AP and remote BLE device has been established. This command will perform a GATT write *with* response to the specified characteristic, with the specified value. It is the 3rd party server's responsibility to know the capabilities of the characteristic.

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID • Value

Request Optional Parameters	• Timeout
Possible Responses	• success • notCurrentlyInConnection • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0005",
            "type": "gattWriteWithResponse",
            "deviceMac": "11:22:33:44:55:66",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "value": 0F 0F,
            "timeOut": 30
        }
    ]
}
```

Example Response

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0005",
            "type": "gattWriteWithResponse",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "success",
            "statusString": "Value written successfully."
        }
    ]
}
```

vi. gattNotification

The gattNotification command must only be called after a connection between an AP and remote BLE device has been established. This command will attempt to subscribe or unsubscribe to notifications for the specified characteristic. In order to subscribe to notifications, you must send a value of "1", and to unsubscribe from notifications, you must send a value of "0".

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID • Value
Request Optional Parameters	• Timeout
Possible Responses	• success • notCurrentlyInConnection • apNotFound • deviceNotFound

Example Request

```
{
    "meta":
        {
            "version": 1,
            "sbTopic": "actions"
        },
    "receiver": {
        "apMac": "aa:bb:cc:dd:ee:ff"
    },
    "actions": [
        {
            "actionId": "0006",
            "type": "gattNotification",
            "deviceMac": "11:22:33:44:55:66",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "value": 1,
            "timeOut": 30
        }
    ]
}
```

gattNotification Responses

For gattNotification actions, the response structure different from previous actions. When you subscribe or unsubscribe to notifications on a GATT characteristic, if the AP was successfully able to perform the operation, you will get a success status returned in the "actionResults" northbound message topic. Following a successful subscription to notifications, any notifications on the GATT characteristics that the 3rd party server has subscribed to will be forwarded back asynchronously. An example of a successful subscription, and an example of a notification value being forwarded, are both provided.

gattNotification Response – Successful subscription

```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "actionResults"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "actionResults": [
        {
            "actionId": "0006",
            "type": "gattNotification",
            "deviceMac": "ff:e1:68:69:8e:57",
            "status": "success",
            "statusString": "Successfully subscribed to notifications"
        }
    ]
}
```


gattNotification Response – Forwarded Notification Value


```
{
    "meta":
        {
            "version": 1,
            "token": "fz36TpBY6KLw1wknMGPr",
            "nbTopic": "characteristics"
        },
    "reporter": {
        "name": "AP-367",
        "mac": "11:22:33:44:55:66",
        "ipv4": "192.168.22.33",
        "ipv6": "fe80::aca5:60ff:fe64:949e",
        "hwType": "BT-AP360",
        "swVersion": "8.6.0.0",
        "swBuild": "72348",
        "time": 15210889010
    },
    "characteristics": [
        {
            "deviceMac": "ff:e1:68:69:8e:57",
            "serviceUuid": "272FE150-6C6C-4718-A3D4-6DE8A3735CFF",
            "characteristicUuid": "272FE151-6C6C-4718-A3D4-6DE8A3735CFF",
            "value": "01"
        }
    ]
}
```


vii. gattIndication

The gattIndiciations actions are very similar to gattNotification actions. All of the actions and responses are exactly the same, just the action type will be "gattIndication" as opposed to "gattNotification". The same subscription response messages are sent to the third-party server. The only difference between the two, is gattNotification and gattIndication work different at the BLE level, and some peripheral BLE devices require gattIndication.

8. Wi-Fi telemetry

The Wi-Fi telemetry service sends periodic reports about all the Wi-Fi devices that are discovered by an AP. The AP sees over the air wireless frames from devices that are in the vicinity of the AP. The AP classifies these devices into (a) associated stations: devices for which we observe bi-directional frames, i.e., going from AP to station and from station to AP, and (b) unassociated stations: devices for which we observe frames either going to the devices or from the device to its associated AP. At every reporting interval, in the periodic report for each station, we will send the tuple of station MAC address, received signal strength (RSSI), and device class.

To enable the WiFi telemetry service in the IoT transport profile, the user will need to include the wifi-assoc-sta and wifi-unassoc-sta classes in the device class filter. The configured server will receive one or more Google Protocol buffer encoded messages, depending upon the number of observed stations, at every reporting interval. Note that the WiFi telemetry is only available when the server type is set to Telemetry-WebSocket.

Example Message

```
{
   "meta": {
      "version": "1",
      "access_token": "any",
      "nbTopic": "telemetry"
   },
   "reporter": {
      "name": "Aruba_AP1",
      "mac": "004e35c76a08",
      "ipv4": "10.5.0.120",
      "ipv6": "fe80::24e:35ff:fec7:6a08",
      "hwType": "AP-365",
      "swVersion": "8.6.0.4",
      "swBuild": "74969",
      "time": "1587663421"
   },
   "wifiData": [
      {
         "mac": "9a5da1e7a59d",
         "deviceClass": [
            "wifiUnassocSta"
         ],
         "rssi": -87
      },
      {
         "mac": "205415caed1a",
         "deviceClass": [
            "wifiAssocSta"
         ],
         "rssi": -75
      }
   ]
}
```

9. WiFi RTLS Data

The WiFi RTLS data telemetry service forwards the wireless data frames that originate from unassociated Wi-Fi tags to the configured server. Wireless packets from unassociated Wi-Fi tags are distinguished from other frames based on the wireless packet type, and the values of the toDS and fromDS flags in the frame control field. When the incoming packet is a data frame with either toDS = 1 and fromDS = 1, or toDS = 0 and fromDS = 0, then the AP tries to match the MAC address from the Address 1 field to the destination MAC address configured in the transport profile. If it is a match then the AP generates a report with the device MAC address, received signal strength (RSSI), device class (set to "wifiTag") and the payload of the wireless frame.

To enable the WiFi RTLS data telemetry service in the IoT transport profile, the user will need to include the wifi-tags class in the device class filter. Whenever the AP sees a frame that matches the MAC address configured in the rtlsDestMAC field in the IoT transport profile, it will immediately send a report to the configured server as a Google Protocol buffer encoded message. Note that the WiFi telemetry is only available when the server type is set to Telemetry-Websocket.

Example Message


```
{
   "meta": {
      "version": "1",
      "access_token": "test",
      "nbTopic": "telemetry"
   },
   "reporter": {
      "name": "Aruba_AP1",
      "mac": "004e35c76a08",
      "ipv4": "10.5.0.120",
      "ipv6": "fe80::24e:35ff:fec7:6a08",
      "hwType": "AP-365",
      "swVersion": "8.6.0.4",
      "swBuild": "74969",
      "time": "1590518273"
   },
   "wifiData": [
      {
         "mac": "000ccc48c5a1",
         "deviceClass": [
            "wifiTag"
         ],
         "rssi": -51,
         "rtls_payload": "00130b06020002003302072 2bc5a000006770407000ccc00001200"
      }
   ]
}
```

10. Zigbee Sockets Data

Aruba defined a new concept for Zigbee data communication, Zigbee Socket Device (ZSD), which can simplify the usage for sending/receiving data over Zigbee. ZSD specifies two parts:

- inbound sockets: Inbound socket is used for receiving data from peer device.
- outbound sockets: Outbound socket is used for sending data to peer device.

Socket consists of 4 members: source-endpoint, destination-endpoint, profile ID and cluster ID. These four parameters are defined into a message 'ZbE2PC' (e2pc) in the API. When a ZSD is bound to an ATW transport profile by configuration, all data related to the e2pc can be transmitted over the ZSD. In fact, e2pc specifies a data tunnel between server and clients. Different services have different e2pc. Sometimes, e2pc for sending is also different from the one for receiving. Similarly, e2pc is like the port of TCP/UDP. Different ports can indicate different services. In the Zigbee world, for each connected device, it has a short network address which is allocated by coordinator/router. This short network address can be treated as the IP address. In Aruba implementation, we use the IEEE address of client device to send data, which is more generic. In the Zigbee stack, we will convert the IEEE address to short address if we have it.

NbZbMsg – Northbound message from inbound socket

Request Required Parameters	• radio_mac IEEE MAC of radio where data is received from
Request Optional Parameters	• report Send data to server. • ack This is used for acknowledgement for the SbZbMsg when 'reqid' is specified. The ack includes 'result' (SUCCEEDED, FAILED) and 'code' which gives the exact failure reason. • response This is used for the 'read' request from SbZbMsg.
Possible Responses	So far, we have no acknowledgement or response for the 'report'.

Example Northbound Message:

```
{
    "meta":{
        "version":1,
        "nbTopic":"zbNbData"
    },
    "reporter":{
        "mac":"80:8d:b7:c0:0d:95"                                   //AP MAC
    },
    "zigbee":{
        "radioMac":"20:4c:03:ff:fe:13:8c:84",               //AP Zigbee radio MAC
        "report":{
            "mac":"00:13:a2:00:41:58:3a:7c",          //EndDevice MAC
            "e2pc":{
                "destination":{
                    "endpoint":2,
```

```
                    "profileId":28674,
                    "clusterId":64514
                },
                "sourceEndpoint":52
            },
            "payload":"000F000A001122334455"
        }
    }
}
```

SbZbMsg – Southbound message to outbound socket

Request Required Parameters	• radio_mac IEEE MAC of radio used for sending data
Request Optional Parameters	• send Send data to peer device. It includes 4 required parameters: reqid, mac, e2pc and payload. • request Not implemented in AOS. it includes these operations: 'read', 'write', 'action'(for other actions).
Possible Responses	• SUCCEEDED • FAILED

Example Southbound Message:

```
{
    "meta":{
        "version":1,
        "sbTopic":3
    },
    "receiver":{
        "apMac":"80:8d:b7:c0:0d:95"                    //AP MAC
    },
    "zigbee":{
        "radioMac":"20:4c:03:ff:fe:13:8c:84",          //AP Zigbee radio MAC
        "send":{
            "mac":"00:13:a2:00:41:58:3a:ce",           //EndDevice MAC
            "e2pc":{
                "destination":{
                    "endpoint":101,
                    "clusterId":28673,
                    "profileId":61441
                },
                "sourceEndpoint":151
            },
            "payload":"7001000A001122334455",
            "reqid":"1"
        }
    }
}
```

11. Serial data

Starting AOS 8.7, Aruba APs support data forwarding service for 3rd party IoT radios that are connected to the AP via the USB port. Every 3rd party radio requires custom integration involving bundling the device driver, port configuration and message parsing sub routines into the AP's software image. When the 3rd party IoT radio is plugged into the USB port, it presents itself as a serial over USB device to the AP after the appropriate driver installation.

The serial data sent by the 3rd party radio to the AP is encoded into a Google Protocol Buffer formatted message and forwarded to the server configured in the IoT transport profile. The server can also send a Google Protocol Buffer formatted message to the AP, which will be forwarded to the 3rd party device (i.e., the serial data bytes will be written to the serial port). The serial data forwarding service is only available when the server type is Telemetry-Websocket. Note that in AOS 8.7, a new parameter setting for device class filter called serial-data was added to enable forwarding messages from the attached 3rd party radio/device. In every SB communication, the server needs to populate the correct USB serial device identifier to ensure that the message is forwarded to the correct device. The device identifier is part of each AP Health Update message and NB serial data messages. In future AOS releases, the device identifier could be unique for a given device connected to an AP, hence the server needs to keep track of it individually on a per AP basis.

Note: nbDeviceId field was introduced in AOS 8.7.1.3. Prior versions of AOS do not send this field. Use of device identifier is mandatory going forward.

Example Message


```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "serialDataNb"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.7.1.3-8.7.1.3",
    "swBuild": "79649",
    "time": "1617055554"
  },
  "nbSData": [
    {
      "nbSerialPayload": "55000f07012bd2a5530158a3e4f757c00412f2948001ffffffff5000c7",
      "nbDeviceId": "ENOCEAN_USB"
    }
  ]
}
```


Example Southbound Message


```
{
    "meta": {
        "access_token": "0123456789",
        "version": 1,
        "sbTopic": "serialDataSb"
    },
    "sbSData": [{
        "sbDeviceId": "ENOCEAN_USB",
        "sbSerialPayload": "5500010005700309"
    }],
    "receiver": {
        "apMac": "904c81cf3886",
```

```
        "all": false
    }
}
```

Example Response to Southbound Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "serialDataNb"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.7.1.3-8.7.1.3",
    "swBuild": "79649",
    "time": "1617055554"
  },
  "nbSData": [
    {
      "nbSerialPayload":
"55002100022600020b000802060100050e3365454f01035245434f4d474154455741590000000009f",
      "nbDeviceId": "ENOCEAN_USB"
    }
  ]
}
```