AWS IoT CoAP/DTLS Service Requirements (Beta)

Acronyms

Column 1	Column 2
ACK	Acknowledgement
Alt	Alternative
CoAP	Constrained Application Protocol
CON	Confirmable
devID	Device identifier
DTLS	Datagram Transport Layer Security
NON	Non-confirmable
NoSec	No security
Opt	Optional
PSK	Pre-shared Key
PUB	Publish
req	Request
resp	Response

Key Features

The AWS CoAP/DTLS service is

- a CoAP client
- a CoAP server
- transported over UDP with DTLS v1.2

The AWS CoAP/UDP service supports

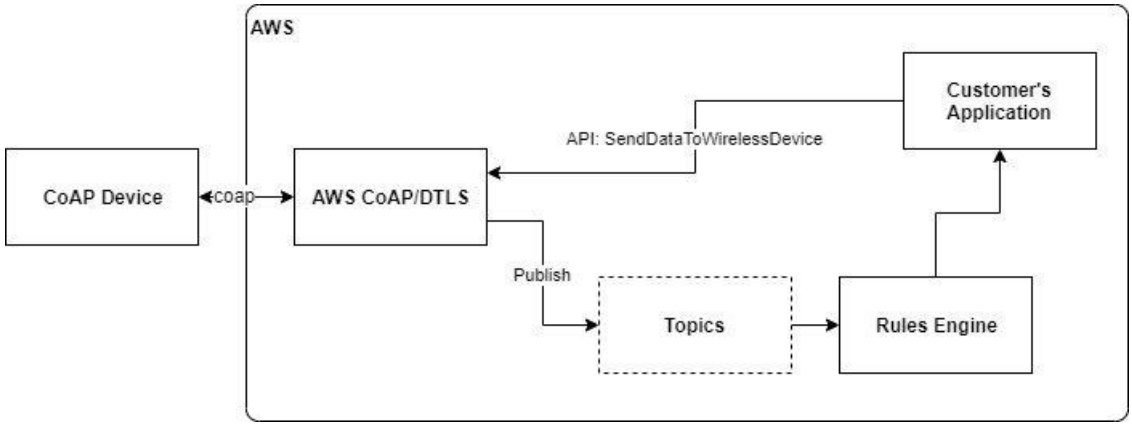
- GET/PUT/POST/DELETE CoAP methods on the uplink and downlink
- downlink messages queues (mode 0: pass through)
- downlink messages queues (mode 1: Storing in Downlink queue)
- [De facto features]
    - passing uplink payload through AWS IoT Core Rules and/or message broker
    - CON (confirmable) and NON (non-confirmable) messages
    - RST (Reset) message
    - ACK (Acknowledgement) message
    - piggybacked response (uplink: DELETE/PUT, downlink: GET/PUT/DELETE/POST)
    - separate response (uplink: GET/POST, downlink: GET/PUT/DELETE/POST)
- DTLS with pre-shared key (PSK) over port 5684
- DTLS session resumption using session ID
- DTLS Connection ID
- PSK with cipher suite AES128_CCM_8 as recommended
    - The list of supported Cipher Suites -
      TLS_ECDHE_PSK_WITH_AES_128_CCM_8_SHA256,
      TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256,
      TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256,
      TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA378,
      TLS_PSK_WITH_AES_128_CCM,
      TLS_PSK_WITH_AES_128_CCM_8,
      TLS_PSK_WITH_AES_128_GCM_SHA256,
      TLS_PSK_WITH_AES_256_CCM,
      TLS_PSK_WITH_AES_256_CCM_8,
      TLS_PSK_WITH_AES_256_GCM_SHA378

The AWS CoAP/UDP server does not support

- Service Discovery - AWS service endpoints are well-defined

- Resource Discovery
- Multicast
- NoSec (no encryption) over port 5683
- proxy
- caching
- protocol translation

Architecture



CoAP methods sequence charts (simplified)

SendDataToWirelessDevice

The Downlink message from Cloud to device is sent using SendDataToWirelessDevice API. It can be used to send both a CoAP Request and a CoAP response. It has the following parameters (only CoAP relevant parameters shown and explained . The full list can be found in API documentation)

```
{
  "PayloadData": "string",
  "WirelessMetadata": {
     "Cellular": {
        # Empty for now
     },
     "WiFi": {
        # Empty for now
     }
     "CoAP": {
        "MessageType": string, # CONFIRMABLE/NON_CONFIRMABLE
        "RequestMethod": string, # GET/PUT/POST/DELETE
        "ResponseStatus":  string, # e.g. CONTENT(2.05)
        "ResponseToken":  string, # To correlate the request
        "UriPaths": List<string>, # Parts of the URI as a list
        "UriQuery":  List<QueryParameter>,
        "LocationPath": List<string>,
        "ContentFormat": string, # e.g "Text/Plain"
        "Accept": string,
        "QueueMode": number, # 0, 1
     }
  }
}
```

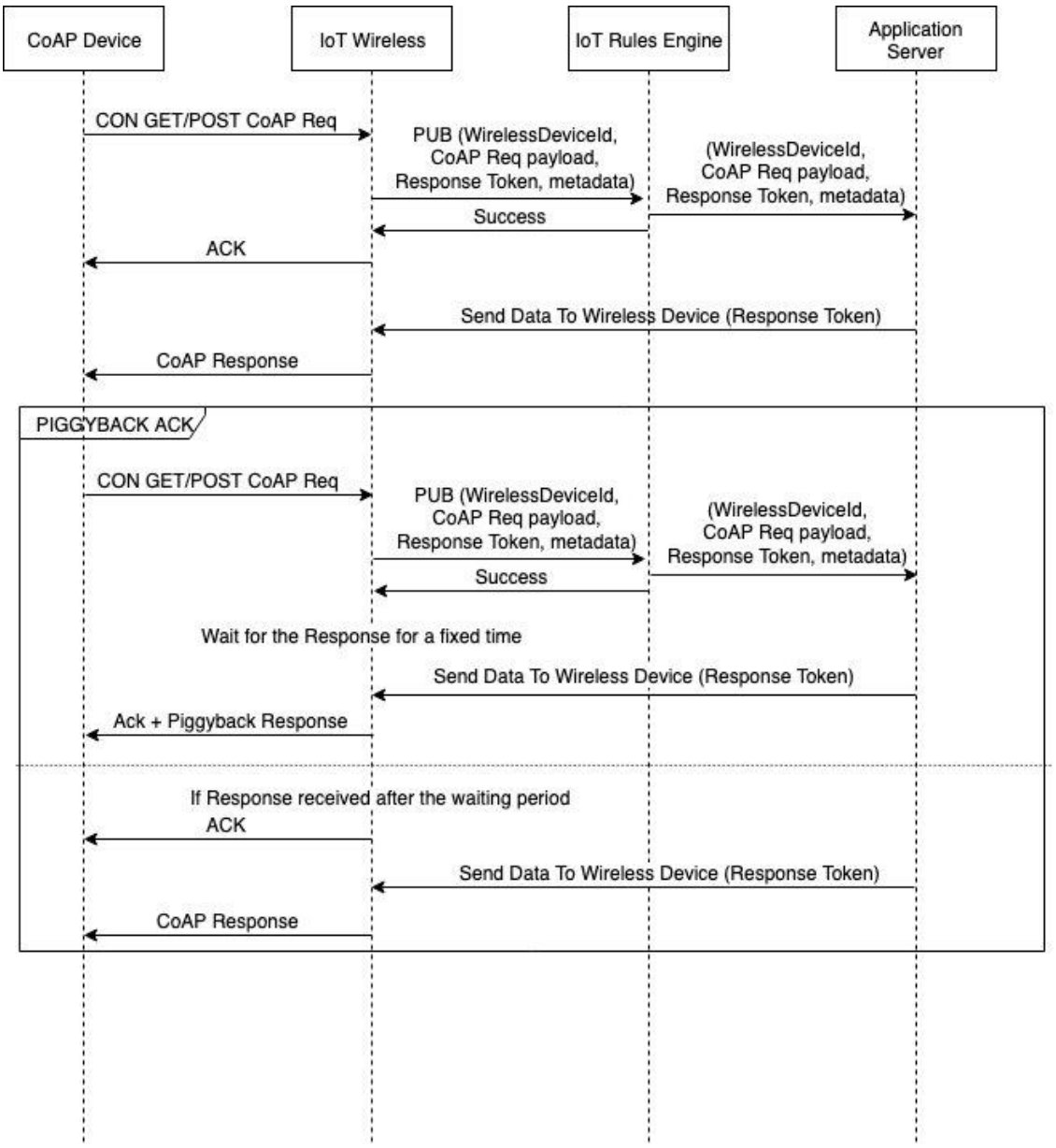
The API uses “WirelessDeviceID” which is an identifier assigned to the device by IoT Wireless and used for its API calls. The “PayloadData“ contains the payload in Base64 format. In the ”CoAP“ wrapper, for a COAP **Request,** the ”RequestMethod“ is

mandatory while for a CoAP Response, ”ResponseStatus“ and ”ResponseToken“ are mandatory and represent a CoAP Response. ResponseToken is a token value different from the CoAP Token in the CoAP header. This ResponseToken is used to correlate an earlier Request to the Response being sent using SendDataToWirelessDevice API in IoT Wireless. The ”QueueMode” informs whether the message has to be queued or not. 0 means no queuing while 1 means the message will be queued till the next uplink.

The successful response to the above API is a message ID which identifies this message within IoT Wireless.

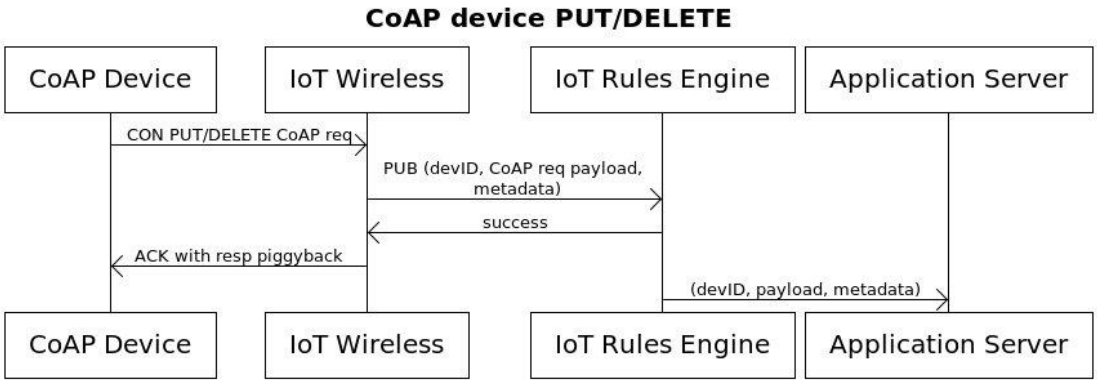
Device GET/POST

When receiving a GET and POST CoAP request from the device, AWS CoAP/DTLS will send the request to the Application Server using Rules Engine. AWS CoAP/DTLS does not generate any response and only sends out an ACK if this a Confirmable message. A token is provided by AWS CoAP/DTLS service and that token has to be included in the response by the Application Server. Once the response is received, AWS CoAP/DTLS service will send a CoAP response to the device. Currently ACK + GET/POST Response piggyback is not supported but in near future it will be supported .



Device PUT/DELETE

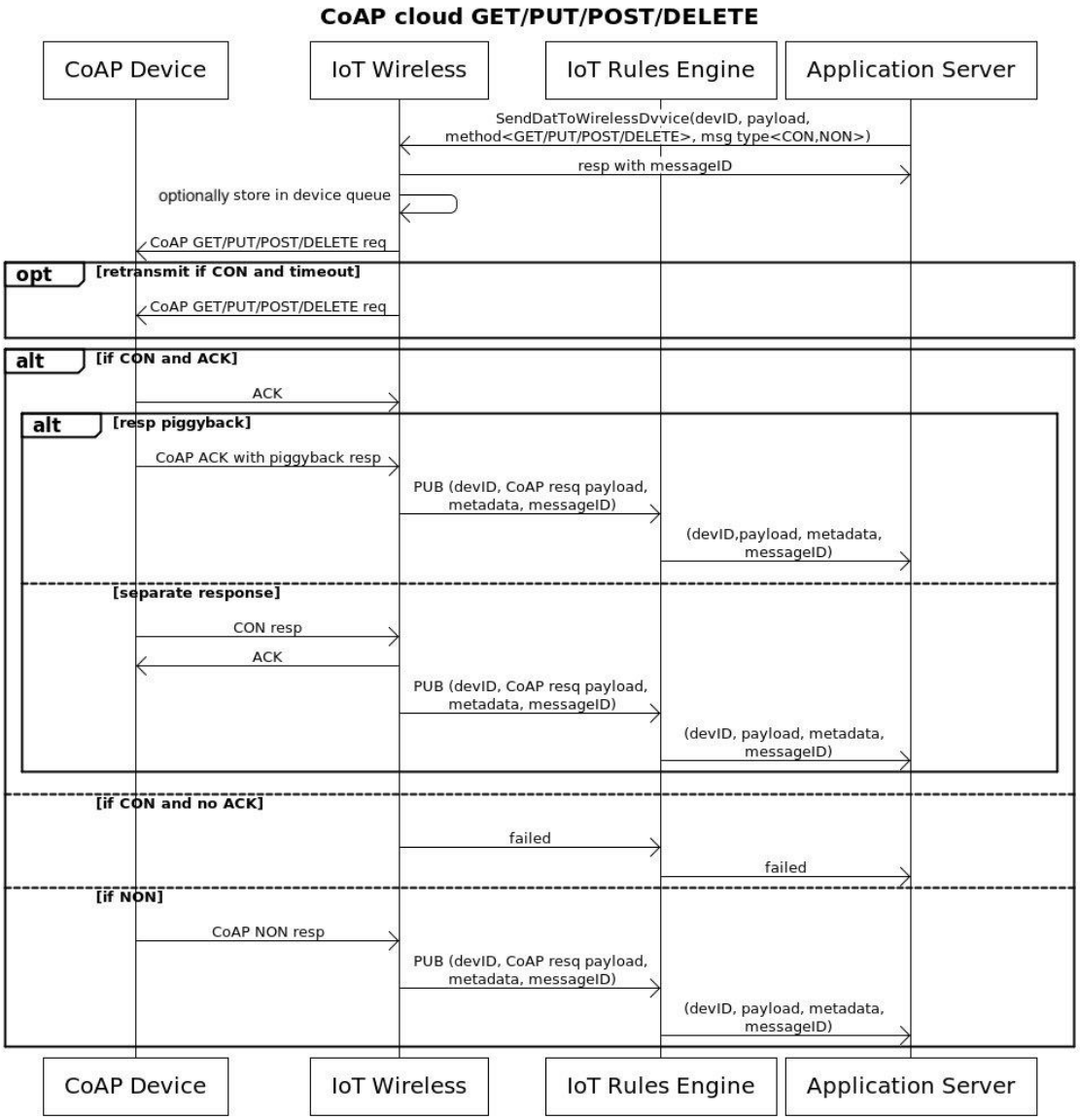
When receiving a PUT and DELETE CoAP request from the device, AWS CoAP/DLTS will send the request to the Application Server using Rules Engine. Once Rules Engine confirms the publish is a success, AWS CoAP/DTLS service will return a 2.xx success response to the device. No further response is expected from the Application Server. This process provides a fast response to the PUT/DELETE request. The response is most likely sent using piggyback if message is confirmable and a separate NON (non-confirmable) message if request was sent with NON.



This figure shows device PUT/DELETE sent using confirmable message

Cloud GET/PUT/POST/DELETE

On the downlink direction, AWS CoAP/DTLS offers the option of message queuing. Application Servers can deliver their downlink messages to AWS CoAP/DTLS by calling the SendDataToWirelssDevice API . If the “queueMode” is set to 0 in the SendDataToWirelssDevice API the message is sent directly to the device otherwise they are sent to the Dowb expect those to be delivered the next time the uplink messages are received from the devices and the UDP session is open.



Device provisioning

Device provisioning is achieved by calling our CreateWirelessDevice API with the CoAP related information and the pre-shared key.

CreateWirelessDevice

Following are the fields in the API request which are relevant to CoAP.

```
{
   "DestinationName": "string",
    "Cellular" {
         "Imei": string,
         "DtlsV1_2": {
            "PreSharedKeyIdentity": string,
            "PreSharedKeySecret": string
         }
    },
    "WiFi" {
         "MacAddress" : string,
         "DtlsV1_2": {
```

```
            "PreSharedKeyIdentity": string,
            "PreSharedKeySecret": string
        }
    },
  "Type": "CoAP"
}
```

The type should be “CoAP”. We provide option to provision parameters for multiple radios for the device. Currently we have “Cellular“ and “Wifi“. The ”IMEI“ in ”Cellular“ and ”MacAddress” in ”Wifi” are different identifiers for the device. Under DTLSv1_2 the PreSharedKeyIdentity and Secret are present. PSKIdentity is of minimum 32 and maximum 256 characters and it can be alphanumeric while PSKSecret is of minimum 32 and maximum 512 characters. PSKSecret can be binary but has to be provided in hex format i.e. values starting with “0x“ will be interpreted as hex and converted to binary(for ex - 0x1234567890ABCDEF1234567890ABCDEF)

Open Questions:

1. Would the priority queue be useful? how many different levels of QoS we should offer?
2. Should customers also provision their device resources during the device creation?
3. In the case of a separate response needs to be sent to the device on the downlink, what is the typical time out value before the device goes back to sleep?
4. Which ones of the CoAP options are important for us to support?
5. When SendDataToWirlessDevice API is invoked, it returns a MessageId. Do we need an API allowing customers to retrieve status of the message using this MessageId, in particular if it was delivered and possibly ACKed by the device?
6. Do Customers see a need, in any scenario, that the Cloud has to initiate a DTLS connection by sending a ClientHello?
7. What is a good default value of NStart (number of CONfirmable messages received waiting for an ACK) for the devices?
8. An ACK for a downlink CONfirmable message is currently not sent as an uplink through the rules engine to the Application. Adding it will increase the number of uplink messages sent from IoT Wireless to IOT Rules engine. Is the ACK needed only for debugging or is there a Business logic built around it?

Tentative Schedule

Number	Item	Details	Scope	Release Date
1	Feature: Queue mode 0 for downlink	Queue mode 0 refers to directly bypassing AWS IoT CoAP, messages will not be buffered	P0	
2	Feature: confirmable and Non-confirmable message type support	Non-confirmable messages means AWS IoT CoAP will not wait for the acknowledgement from the device for downlink and will not send an acknowledgement for uplink	P0	
3	Feature: DTLS with pre-shared key		P0	
4	Feature: DTLS session ID resumption		P0	
5	Feature: DTLS connection ID resumption		P0	
6	Feature: Support GET, PUT, POST, DELETE CoAP methods	Both uplink and downlink	P0	
7	GDPR compliance	Support erasing all customer data when their account is terminated.	P0	
8	Monitoring: Customer logging	AWS IoT CoAP sends a log message to customer's CloudWatch every time a 4xx or 5xx error occurs	P0	
9	IAM	Support IAM operations for all APIs changed by CoAP.	P0	
10	Tagging	Support tagging for all resources created by CoAP.	P0	
11	Service Quota	Support Service Quota features for all APIs changed by CoAP.	P0	
12	PrivateLink (api.iotwireless endpoint only)	Support PrivateLink for management plane public endpoint.	P0	December
13	Feature: Queue mode 1 for downlink	Queue messages if the device does not currently have a DTLS session established with AWS IoT CoAP. Once a session is established, queued messages will be sent	P1	
14	Feature: Downlink queue management	Customers can call list and delete on messages sent using SendDataToWirelessDevice	P1	End of Jan
15	Feature: Block transfer downlink (no S3)		P1	
16	Feature: Block transfer uplink		P1	End of Feb
17	Enable piggyback mechanism		P0	
18	Feature: Block transfer downlink (with S3)		P1	
19	Feature: Create/Get/Delete/List ApplicationProfile	Customizable parameters are: NStart (uplink)	P1	End of March
20	Console	Support all API changes in AWS IoT CoAP console	P1	
21	Feature: AWS IoT CoAP initiated DTLS downlink	Supporting AWS IoT CoAP initiated downlink would mean that AWS IoT CoAP is the client. AWS IoT CoAP would need a way to authenticate itself with a pre-shared key	P1	
22	Feature: Network Analyzer		P1	
23	Data plane AuthZ	Offer data plane AuthZ similar to IoT Policy with CoAP device as the resource	P1	6/30/2022
24	Feature: X509 support		P1	TBD
25	Feature: NoSec	No DTLS security layer, will probably need to use a site-to-site VPN for some sort of security	P1	TBD
26	Event Notification	AWS IoT CoAP publishes an event each time a device changes state or something goes wrong	P1	TBD