ToA - App

Bruno Mendes
Electronic and Informatics Engineering Dpt.
University of Algarve
Gambelas, Portugal
a62181@ualg.pt

***Abstract*—**

I. Introduction

– intro about chirpstack - toa - optimization - lora –

II. How it works

O Application Server do Chirpstack oferece uma opção de integração (fig 1) com uma aplicação criada pelo o utilizador, ou seja, irá criar post requests para o tipo de evento configurado pelo end-point (fig. 2) sendo que o parametro do url irá definir o tipo de evento. Neste caso apenas foi configurado para receber uplinks.

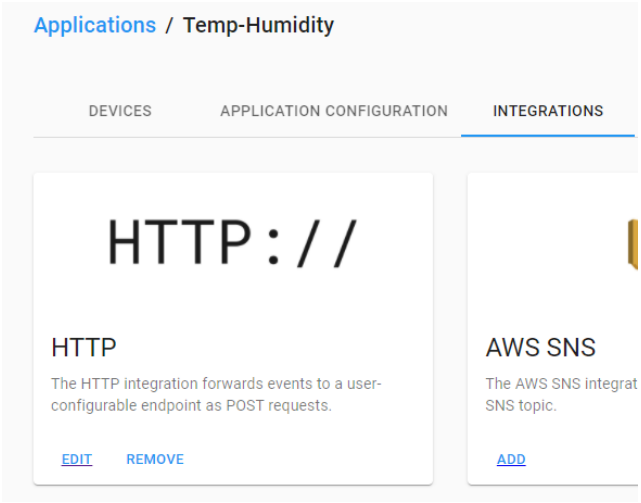

Figure 1.

A aplicação que irá ser criada, denominaremos de TOA(ToA Optimization App) para facilitar a diferenciação entre Chirpstack App Server e esta aplicação.

A ideia da TOA é de funcionar como ponte entre a API do Chirpstack e qualquer algoritmo de optimização de Time on Air. Esta aplicação vai ser constituída por duas partes: 1) um servidor que esteja sempre a receber a informação das frames dos dispositivos da API do Chirpstack e redirecciona-los para um MQTT broker com a informação filtrada dessas frames, 2) um cliente mqtt que esteja subscrito ao mqtt broker e que leia frames do algoritmo de optimização e converta para um formato aceitável de downlink (fig 3).

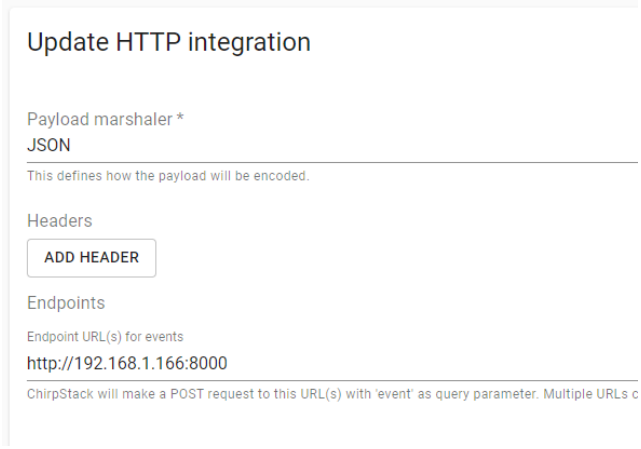

Figure 2.

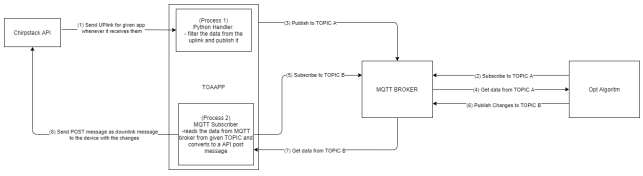

Figure 3.

A. Handler

O Handler é a parte responsável por receber informação da API do Chirpstack, filtrar a informação e enviar para o mqtt broker. Para a construção desta parte foi utilizada como base o código fornecido em [1]. Após receber o uplink (fig. 5), apenas reteve-se a informação necessária para a optimização de Time on Air e para criar um downlink.

```
192.168.1.9 - - [07/Apr/2021 03:32:51] "POST /?event=up HTTP/1.1" 200 -
Uplink received from: fdaffdfaadbfadfa with payload: 0167017f018806765ff2960a0003e8
```

Figure 4. uplink recebido no Handler

```
192.168.1.9 - - [07/Apr/2021 03:32:51] "POST /?event=up HTTP/1.1" 200 -
Uplink received from: fdaffdfaadbfadfa with payload: 0167017f018806765ff2960a0003e8
```

Figure 5. uplink recebido no Handler

Manteve-se então dev_eui, fPort, Spreading Factor, Bandwidth, Signal-to-Noise Ratio, Coding Rate. Dada esta

informação colocou-se num objecto JSON, codificou-se em base 64 e publicou-se no Tópico A (fig. 6).

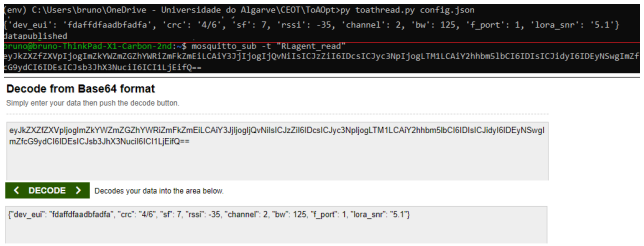

Figure 6. Messagem publicada no MQTT broker com tópico RLagent_read

(talvez explicar com mais detalhe como é que isto tudo é construído?)

B. MQTT sub + API post

Nesta parte é necessário criar um cliente que fica subscrito ao MQTT broker indefinidamente. Quando recebe uma mensagem do tipo Tópico B, este MQTT irá converter a mensagem do Tipo B para uma mensagem do tipo POST para a API que representa um downlink para o dispositivo definido. Para tal utilizou-se a biblioteca do pythong ”requests” em que se define um header com um token e uma payload que representa a mensagem de downlink. A estrutura da payload está definida na fig 8

apiEnqueueDeviceQueueItemRequest {
deviceQueueItem (apiDeviceQueueItem, *optional*): Queue-item object to enqueue.
}
apiDeviceQueueItem {
confirmed (boolean, *optional*): Set this to true when an acknowledgement from the device is required. Please note that this must not be used to guarantee a delivery.,
data (string, *optional*): Base64 encoded data. Or use the json_object field when an application codec has been configured.,
devEUI (string, *optional*): Device EUI (HEX encoded).,
fCnt (integer, *optional*): Downlink frame-counter. This will be automatically set on enqueue.,
fPort (integer, *optional*): FPort used (must be > 0),
jsonObject (string, *optional*): JSON object (string). Only use this when an application codec has been configured that can convert this object into binary form.
}

Figure 7.

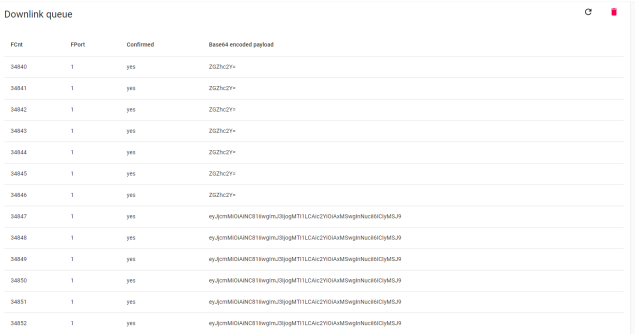

Figure 8. Data que vai ser enviada por Downlink. Nota: só enviado 1 pacote por Uplink porque o dispositivo é de class A, logo só está disponivel a receber downlinks num espaço de intervalo curto depois de enviar o uplink