Trama de datos LoRaWAN

El siguiente documento aborda los distintos tipos de mensajes transmitidos entre el Gateway LoRaWAN y el servidor UDP.

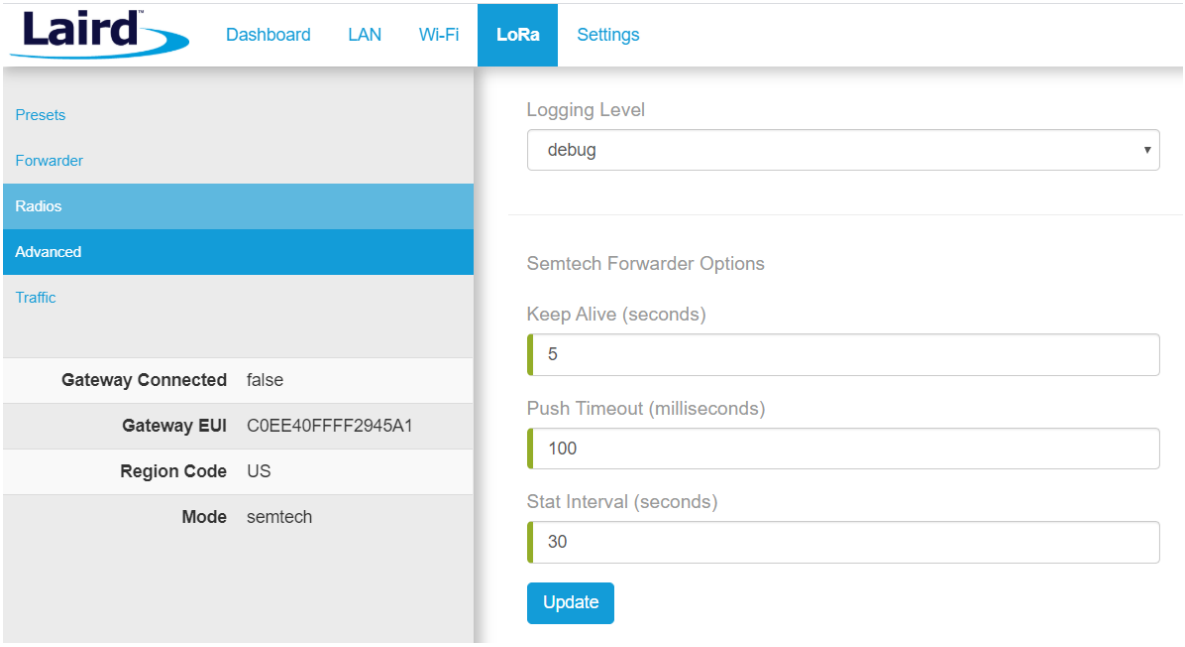
Contenido

1. Trama de subida 2
   1.1. PULL_DATA 2
   1.2. PUSH_DATA 3
      1.2.1. JSON stat, información de estado 3
      1.2.2. JSON rxpk, información de nodo 4
2. Trama de bajada 7
   2.1. PULL_ACK 7
   2.2. PUSH_ACK 7
   2.3. PULL_RESP 7

1. Trama de subida

1.1. PULL_DATA

Es un pequeño mensaje de 12 bytes, es enviado cada cierto tiempo al servidor para indicar que el Gateway está conectado. El tiempo entre envíos de este mensaje se puede modificar en el Gateway, en el campo llamado *Keep Alive*.



Si el mensaje se lee como ASCII, este debe convertirse a hexadecimal:

ASCII	Hexadecimal
⍰)═ ᴸ⁻@)Eí	02 29 25 02 C0 EE 40 FF FF 29 45 A1

El mensaje tiene el siguiente formato:

Byte	Ejemplo	Descripción
[0]	**02**	Versión de firmware, siempre es **02**.
[1-2]	**29 45**	Token, numero aleatorio generado por el Gateway.
[3]	**02**	Indica el tipo de mensaje, para PULL_DATA es **02**.
[4-11]	**C0 EE 40 FF FF 29 45 A1**	Es el Gateway EUI, la dirección única del Gateway.

Este mensaje debe ser respondido por un PULL_ACK desde el servidor.

1.2. PUSH_DATA

Se trata de un mensaje con un JSON incluido, previo a este JSON se tiene un mensaje hexadecimal de 12 bytes, compuesto por los siguientes campos.

02 A9 28 00 C0 EE 40 FF FF 29 45 A1

Byte	Ejemplo	Descripción
[0]	**02**	Versión de firmware, siempre es **02**.
[1-2]	**A9 28**	Token, numero aleatorio generado por el Gateway.
[3]	**00**	Indica el tipo de mensaje, para PUSH_DATA es **00**.
[4-11]	**C0 EE 40 FF FF 29 45 A1**	Es el Gateway EUI, la dirección única del Gateway.

Este mensaje debe ser respondido por un PUSH_ACK desde el servidor.

Existen dos tipos de mensajes PUSH_DATA, se diferencian por el tipo de JSON recibido.

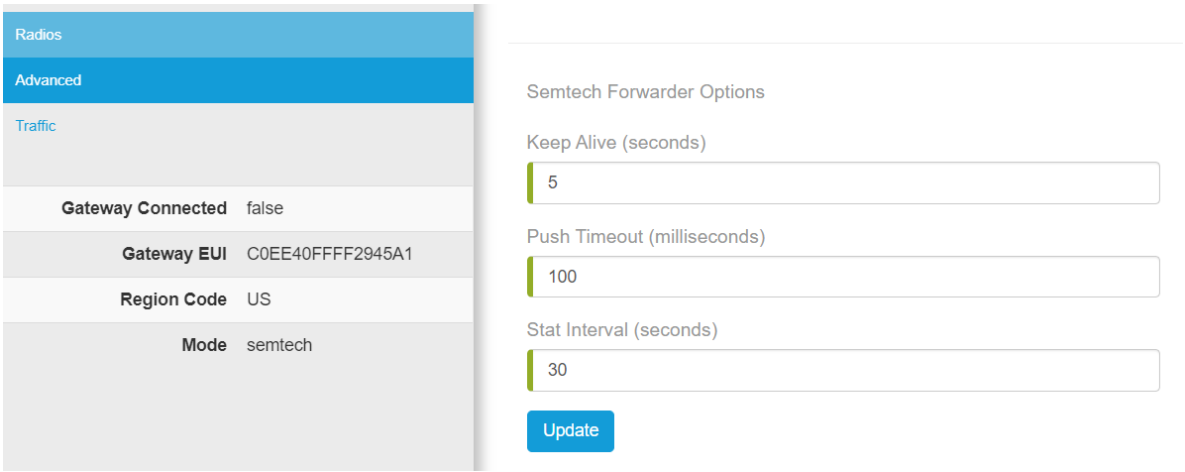
1.2.1. JSON stat, información de estado

El paquete JSON recibido es el siguiente:

```
{
        "stat": {
                "time": "2020-03-18 20:39:10 GMT",
                "rxnb": 0,
                "rxok": 0,
                "rxfw": 0,
                "ackr": 0.0,
                "dwnb": 1,
                "txnb": 0
        }
}
```

La información aquí recibida solo habla del estado del Gateway, por el momento no es necesario leer ni guardar ninguno de los datos aquí mostrados.

El tiempo entre envíos de este mensaje se puede modificar en el Gateway, en el campo llamado *Stat interval*.



1.2.2. JSON rxpk, información de nodo

Los paquetes recibidos del Gateway tienen estructura json como está:


```
{
        "rxpk": [{
                "tmst": 20809572,
                "chan": 2,
                "rfch": 0,
                "freq": 904.300000,
                "stat": 1,
                "modu": "LORA",
                "datr": "SF10BW125",
                "codr": "4/5",
                "lsnr": 9.0,
                "rssi": -5,
                "size": 24,
                "data": "QLsCAACAAQBj2At4l/efuOnEQyBtGTnH"
        }]
}
```


En la sección “data” viene la información relevante al nodo. Esta información está en formato HEX64, por lo que se tendrá que decodificar a formato hexadecimal.

Información en HEX64:

QLsCAACAAQBj2At4l/efuOnEQyBtGTnH

Información en Hexadecimal:

40 BB 02 00 00 80 01 00 63 D8 0B 78 97 F7 9F B8 E9 C4 43 20 6D 19 39 C7

Dirección del dispositivo, bytes [1-4]

De la trama de datos hexadecimal se obtiene la dirección del nodo que está compuesta por los bytes 1 al 4 invertidos. Es decir, en este caso serían:

40 **BB 02 00 00** 80 01 00 63 D8 0B 78 97 F7 9F B8 E9 C4 43 20 6D 19 39 C7

Una vez seleccionados los 4 bytes de la dirección, es necesario invertirlos y convertirlos a decimal.

Original	Invertido	Convertido a decimal
BB 02 00 00	00 00 02 BB	699

Numero de mensaje, bytes [6-7]

De la trama de datos hexadecimal se obtiene el número de mensaje que está compuesto por los bytes 6 al 7 invertidos. Es decir, en este caso serían:

40 BB 02 00 00 80 **01 00** 63 D8 0B 78 97 F7 9F B8 E9 C4 43 20 6D 19 39 C7

Original	Invertido	Convertido a decimal
01 00	00 01	1

Información de consumo, bytes [9-19]

Selección de la trama

De la trama de datos hexadecimal se obtiene la información de consumo que está compuesta por los bytes 9 al 19. Es decir, en este caso serían:

40 BB 02 00 00 80 01 00 63 **D8 0B 78 97 F7 9F B8 E9 C4 43 20** 6D 19 39 C7

Desencriptación

Una vez teniendo la trama encriptada, esta debe pasar por la función de desencriptación que se muestra a continuación[1]. Se adjunta la DLL con esta función. (AES128.dll).

```
/*!
 * Computes the LoRaMAC payload encryption
 *
 * \param [IN]  buffer          - Data buffer
 * \param [IN]  size            - Data buffer size
 * \param [IN]  key             - AES key to be used
 * \param [IN]  address         - Frame address
 * \param [IN]  dir             - Frame direction [0: uplink, 1: downlink]
 * \param [IN]  sequenceCounter - Frame sequence counter
 * \param [OUT] encBuffer       - Encrypted buffer
 */
void LoRaMacPayloadEncrypt( const uint8_t *buffer, uint16_t size, const uint8_t *key, uint32_t address, uint8_t dir, uint32_t sequenceCounter, uint8_t *encBuffer );
```

La llave AES *key* es la siguiente: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

La dirección o *address* es el número de dispositivo, en este caso: 699

La dirección de envío o *dir* es *uplink,* por lo que el valor es 0.

El *sequenceCounter* es el número de mensaje, en este caso: 1

Después de ejecutar esta función, el arreglo *encBuffer* tendrá la información desencriptada, que debe verse así:

Información encriptada	Información desencriptada
D8 0B 78 97 F7 9F B8 E9 C4 43 20	04 65 01 79 05 65 00 00 06 00 1B

Tipos de información de consumo

Existen tres casos diferentes para leer la información de consumo.

1. Caso M0, para todas las direcciones entre 1 y 700, también direcciones entre 1001 y 1020.

La potencia consumida en **Watts-hora** está conformada por los bytes 2 y 3.

Trama	Valor seleccionado	Valor en Wh
04 65 **01 79** 05 65 00 00 06 00 1B	**01 79**	377

La potencia instantánea dada en **Watts** está conformada por los bytes 6 y 7.

Trama	Valor seleccionado	Valor en W
04 65 01 79 05 65 **00 00** 06 00 1B	**00 00**	0

[1] La función para encriptar y desencriptar es la misma.

La temperatura del dispositivo en **grados centígrados** está conformada por el byte 10.

Trama	Valor seleccionado	Valor en °C
04 65 01 79 05 65 00 00 06 00 **1B**	**1B**	27

2. Caso M1, para todas las direcciones entre 701 y 850, también direcciones entre 2001 y 2020.

La potencia consumida en **kilowatts-hora** está conformada por los bytes 2 y 3.

Trama	Valor seleccionado	Valor en KWh
04 65 **DD 94** 05 65 0A A7 06 00 1B	**DD 94**	56724

La potencia instantánea dada en **Watts** está conformada por los bytes 6 y 7.

Trama	Valor seleccionado	Valor en W
04 65 DD 94 05 65 **0A A7** 06 00 1B	**0A A7**	2727

La temperatura del dispositivo en **grados centígrados** está conformada por el byte 10.

Trama	Valor seleccionado	Valor en °C
04 65 DD 94 05 65 0A A7 06 00 **1B**	**1B**	27

3. Caso H2O, para todas las direcciones entre 851 y 1000, también direcciones entre 3001 y 3020.

Los **litros** de agua consumida están conformados por los bytes 2 y 3.

Trama	Valor seleccionado	Valor en litros
04 65 **00 00** 05 00 64 06 67 01 0E	**00 00**	0

El nivel de la batería en **porcentaje** está dado por el byte 6.

Trama	Valor seleccionado	Valor en porcentaje
04 65 00 00 05 00 **64** 06 67 01 0E	**64**	100

La temperatura del dispositivo en **grados centígrados** está conformada por los bytes 9 y 10 divido entre 10.

Trama	Valor seleccionado	Valor en °C
04 65 00 00 05 00 64 06 67 **01 0E**	**01 0E**	270/10=27

2. Trama de bajada

2.1. PULL_ACK

Este mensaje se usa para responder a cada PULL_DATA recibido en el servidor. Está conformado por 12 bytes. Por ejemplo:

02 29 25 04 C0 EE 40 FF FF 29 45 A1

Byte	Ejemplo	Descripción
[0]	**02**	Versión de firmware, siempre es **02**.
[1-2]	**29 45**	Token, numero aleatorio generado por el Gateway.
[3]	**04**	Indica el tipo de mensaje, para PULL_ACK es **04**.
[4-11]	**C0 EE 40 FF FF 29 45 A1**	Es el Gateway EUI, la dirección única del Gateway.

Este mensaje debe ser enviado después de recibir un PULL_DATA, y el Token debe ser igual al recibido en el último PULL_DATA.

Si se requiere enviar un JSON a un nodo especifico, se debe enviar un PULL_RESP.

2.2. PUSH_ACK

Este mensaje se usa para responder a cada PUSH_DATA recibido en el servidor. Está conformado por 4 bytes. Por ejemplo:

02 29 25 04

Byte	Ejemplo	Descripción
[0]	**02**	Versión de firmware, siempre es **02**.
[1-2]	**A9 28**	Token, numero aleatorio generado por el Gateway.
[3]	**01**	Indica el tipo de mensaje, para PULL_ACK es **01**.

Este mensaje debe ser enviado después de recibir un PUSH_DATA, y el Token debe ser igual al recibido en el último PUSH_DATA.

2.3. PULL_RESP

Antes del JSON a enviar se debe agregar un mensaje de 4 bytes con la siguiente información:

02 29 25 04

Byte	Ejemplo	Descripción
[0]	**02**	Versión de firmware, siempre es **02**.
[1-2]	**29 45**	Token, numero aleatorio generado por el Gateway.
[3]	**03**	Indica el tipo de mensaje, para PULL_ACK es **04**.

Este mensaje debe ser enviado después de recibir un PULL_DATA, y el Token debe ser igual al recibido en el último PULL_DATA.

Si no se requiere enviar información con un JSON adjunto, entonces el servidor debe enviar un PULL_ACK en su lugar.

Los paquetes enviados al Gateway con JSON tienen el siguiente formato:


```
{
        "txpk": {
                "imme": false,
                "tmst": 22809572,
                "freq": 923.3,
                "rfch": 0,
                "powe": 17,
                "modu": "LORA",
                "datr": "SF12BW500",
                "codr": "4/5",
                "ipol": true,
                "size": 14,
                "ncrc": true,
                "data": "YLsCAAAAAgAEgt1MwHc="
        }
}
```


Los únicos dos campos a modificar son tmst (es la marca de tiempo del Gateway) y data (la información que será enviada al dispositivo). Este ejemplo concuerda con la trama de subida.

Para calcular el tmst se requiere sumar 2000000 al tmst recibido en la trama de subida.

$$22809572 = 20809572 + 2000000$$

Para crear el campo data se requiere hacer lo siguiente:

Información en HEX64:

YLsCAAAAAgAEgt1MwHc=

Información en Hexadecimal:

60 BB 02 00 00 00 02 00 04 82 DD 4C C0 77

Byte	Ejemplo	Descripción
[0]	**60**	Tipo de mensaje, siempre es 60
[1-4]	**BB 02 00 00**	Dirección del dispositivo invertida, este caso es 02 BB (hex) = 699 (dec)
[5]	**00**	Byte de control, siempre es 00
[6-7]	**02 00**	Numero de mensaje invertido, en este caso es 02 00 (hex) = 2 (dec), su valor debe irse incrementando por cada envío.
[8]	**04**	Numero de puerto para recibir mensaje, siempre es 4
[9]	**82**	Información a enviar, se encuentra encriptada
[10-13]	**DD 4C C0 77**	MIC (*message integrity code*), código generado a partir de todos los 10 bytes previamente formados.

Información a enviar, byte [9]

Este byte puede tener dos valores 00 (para desconectar la carga) o FF (para conectar la carga). Pero antes de ser agregado a la trama hexadecimal, debe ser encriptado.

Encriptación

```
/*!
 * Computes the LoRaMAC payload encryption
 *
 * \param [IN]  buffer          - Data buffer
 * \param [IN]  size            - Data buffer size
 * \param [IN]  key             - AES key to be used
 * \param [IN]  address         - Frame address
 * \param [IN]  dir             - Frame direction [0: uplink, 1: downlink]
 * \param [IN]  sequenceCounter - Frame sequence counter
 * \param [OUT] encBuffer       - Encrypted buffer
 */
void LoRaMacPayloadEncrypt( const uint8_t *buffer, uint16_t size, const uint8_t *key, uint32_t address, uint8_t dir, uint32_t sequenceCounter, uint8_t *encBuffer );
```

La llave AES *key* es la siguiente: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

La dirección o *address* es el número de dispositivo, en este caso: 699

La dirección de envío o *dir* es *downlink,* por lo que el valor es 1.

El *sequenceCounter* es el número de mensaje, en este caso: 2

El valor obtenido de aplicar el proceso anterior se obtiene **82.** Ya que este byte depende del número de dispositivo y el número de mensaje, el valor siempre va a cambiar.

MIC (Message Integrity Code), bytes [10-13]

Para calcular este código deben tenerse los primeros 10 bytes ya formados.

```
/*!
 * Computes the LoRaMAC frame MIC field
 *
 * \param [IN]  buffer          - Data buffer
 * \param [IN]  size            - Data buffer size
 * \param [IN]  key             - AES key to be used
 * \param [IN]  address         - Frame address
 * \param [IN]  dir             - Frame direction [0: uplink, 1: downlink]
 * \param [IN]  sequenceCounter - Frame sequence counter
 * \param [OUT] mic             - Computed MIC field
 */
void LoRaMacComputeMic( const uint8_t *buffer, uint16_t size, const uint8_t *key, uint32_t address, uint8_t dir, uint32_t sequenceCounter, uint32_t *mic );
```

El *buffer* de entrada son los 10 bytes previamente formados, en este caso:

60 BB 02 00 00 00 02 00 04 82

La llave AES *key* es la siguiente: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

La dirección o *address* es el número de dispositivo, en este caso: 699

La dirección de envío o *dir* es *downlink,* por lo que el valor es 1.

El *sequenceCounter* es el número de mensaje, en este caso: 2

El arreglo *mic* tendrá el código generado, que en este caso es: DD 4C C0 77

Finalmente se unen los primeros diez bytes con el MIC y se obtiene la trama completa:

60 BB 02 00 00 00 02 00 04 82 DD 4C C0 77