NOTICE OF USE AND DISCLOSURE

LoRa Alliance, Inc.
3855 SW 153rd Drive
Beaverton, OR 97007

LoRaWAN™ 1.0.3 Specification

Authored by the LoRa Alliance Technical Committee

Chairs:
N.SORNIN (Semtech), A.YEGIN(Actility)
Editor:
N.SORNIN(Semtech)
Contributors:
A.BERTOLAUD (Gemalto), J.CATALANO (Kerlink), J.DELCLEF (ST Microelectronics), V.DELPORT (Microchip Technology), P.DUFFY (Cisco), F.DYDUCH (Bouygues Telecom), T.EIRICH (TrackNet), L.FERREIRA (Orange), S.GHAROUT(Orange), O.HERSENT (Actility), A.KASTTET (Homerider Systems), D.KJENDAL (Senet), V.KLEBAN (Everynet), J.KNAPP (TrackNet), T.KRAMP (TrackNet), M.KUYPER (TrackNet), P.KWOK (Objenious), M.LEGOURIEREC (Sagemcom), C.LEVASSEUR (Bouygues Telecom), M.LUIS (Semtech), M.PAULIAC (Gemalto), P.PIETRI (Orbiwise), D.SMITH (MultiTech), R.SOSS(Actility), T.TASHIRO (M2B Communications), P.THOMSEN (Orbiwise), A.YEGIN (Actility)

Version: V1.0.3
Date: March 20, 2018
Status: Final release

Contents

Tables

Figures

1 Introduction

This document describes the LoRaWAN™ network protocol which is optimized for battery-powered end-devices that may be either mobile or mounted at a fixed location.

LoRaWAN networks typically are laid out in a star-of-stars topology in which **gateways**[1] relay messages between **end-devices**[2] and a central **network server** at the backend. Gateways are connected to the network server via standard IP connections while end-devices use single-hop LoRa™ or FSK communication to one or many gateways.[3] All communication is generally bi-directional, although uplink communication from an end-device to the network server is expected to be the predominant traffic.

Communication between end-devices and gateways is spread out on different **frequency channels** and **data rates**. The selection of the data rate is a trade-off between communication range and message duration, communications with different data rates do not interfere with each other. LoRa data rates range from 0.3 kbps to 50 kbps. To maximize both battery life of the end-devices and overall network capacity, the LoRa network infrastructure can manage the data rate and RF output for each end-device individually by means of an **adaptive data rate** (ADR) scheme.

End-devices may transmit on any channel available at any time, using any available data rate, as long as the following rules are respected:

- The end-device changes channel in a pseudo-random fashion for every transmission. The resulting frequency diversity makes the system more robust to interferences.
- The end-device respects the maximum transmit duty cycle relative to the sub-band used and local regulations.
- The end-device respects the maximum transmit duration (or dwell time) relative to the sub-band used and local regulations.

> **Note:** Maximum transmit duty-cycle and dwell time per sub-band are region specific and are defined in the LoRaWAN ¨regional phyLayer definition document

1.1 LoRaWAN Classes

All LoRaWAN devices implement at least the Class A functionality as described in this document. In addition they MAY implement options named Class B, Class C as also described in this document or others to be defined. In all cases, they must remain compatible with Class A.

[1] Gateways are also known as **concentrators** or **base stations**.
[2] End-devices are also known as **motes**.
[3] Support for intermediate elements – repeaters – is not described in the document, however payload restrictions for encapsulation overhead are included in this specification. A repeater is defined as using LoRaWAN as its backhaul mechanism.

1.2 Conventions

MAC commands are written ***LinkCheckReq***, bits and bit fields are written **FRMPayload**, constants are written RECEIVE_DELAY1, variables are written *N*.

In this document,

- The octet order for all multi-octet fields is little endian and
- EUI are 8 bytes multi-octet fields and are transmitted as little endian.
- By default, RFU bits are set to zero

2 Introduction on LoRaWAN options

LoRa™ is a wireless modulation for long-range low-power low-data-rate applications developed by Semtech. Devices implementing more than Class A are generally named "higher Class end-devices" in this document.

2.1 LoRaWAN Classes

A LoRa network distinguishes between a basic LoRaWAN (named Class A) and optional features (Class B, Class C …):

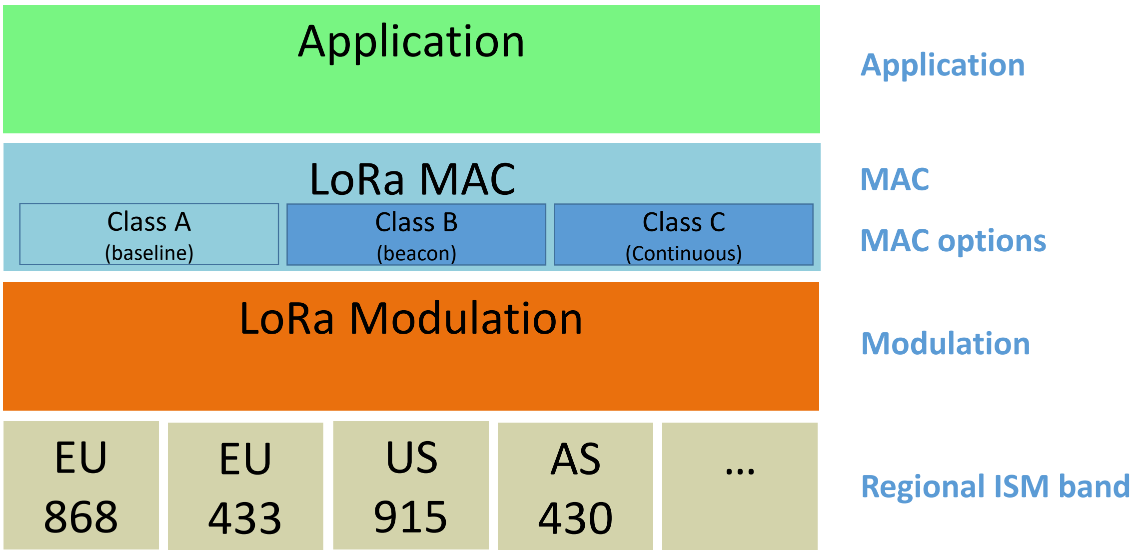


Figure 1: LoRaWAN Classes

- **Bi-directional end-devices (Class A):** End-devices of Class A allow for bi-directional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. The transmission slot scheduled by the end-device is based on its own communication needs with a small variation based on a random time basis (ALOHA-type of protocol). This Class A operation is the lowest power end-device system for applications that only require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time will have to wait until the next scheduled uplink.
- **Bi-directional end-devices with scheduled receive slots (Class B):** End-devices of Class B allow for more receive slots. In addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times. In order for the End-device to open it receive window at the scheduled time it receives a time synchronized Beacon from the gateway. This allows the server to know when the end-device is listening.
- **Bi-directional end-devices with maximal receive slots (Class C):** End-devices of Class C have nearly continuously open receive windows, only closed when transmitting. Class C end-device will use more power to operate than Class A or Class B but they offer the lowest latency for server to end-device communication.

2.2 Specification scope

This LoRaWAN specification describes the additional functions differentiating an end-device higher Class from one of Class A. A higher Class end-device SHALL also implement all the functionality described in the LoRaWAN Class A specification.

> **NOTE:** Physical message format, MAC message format, and other parts of this specification that are common to both end-devices of Class A and higher Classes are described only in the LoRaWAN Class A specification to avoid redundancy.

CLASS A – ALL END-DEVICES

All LoRaWAN end-devices MUST implement Class A features.

3 Physical Message Formats

The LoRa terminology distinguishes between uplink and downlink messages.

3.1 Uplink Messages

Uplink messages are sent by end-devices to the network server relayed by one or many gateways.

Uplink messages use the LoRa radio packet explicit mode in which the LoRa physical header (**PHDR**) plus a header CRC (**PHDR_CRC**) are included.[1] The integrity of the payload is protected by a CRC.

The **PHDR**, **PHDR_CRC** and payload **CRC** fields are inserted by the radio transceiver.

Uplink PHY:

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

Figure 2: Uplink PHY structure

3.2 Downlink Messages

Each **downlink message** is sent by the network server to only one end-device and is relayed by a single gateway.[2]

Downlink messages use the radio packet explicit mode in which the LoRa physical header (**PHDR**) and a header CRC (**PHDR_CRC**) are included.[3]

Downlink PHY:

Preamble	PHDR	PHDR_CRC	PHYPayload

Figure 3: Downlink PHY structure

3.3 Receive Windows

Following each uplink transmission the end-device opens two short receive windows. The receive window start times are defined using the end of the transmission as a reference.

[1] See the LoRa radio transceiver datasheet for a description of LoRa radio packet implicit/explicit modes.
[2] This specification does not describe the transmission of multicast messages from a network server to many end-devices.
[3] No payload integrity check is done at this level to keep messages as short as possible with minimum impact on any duty-cycle limitations of the ISM bands used.

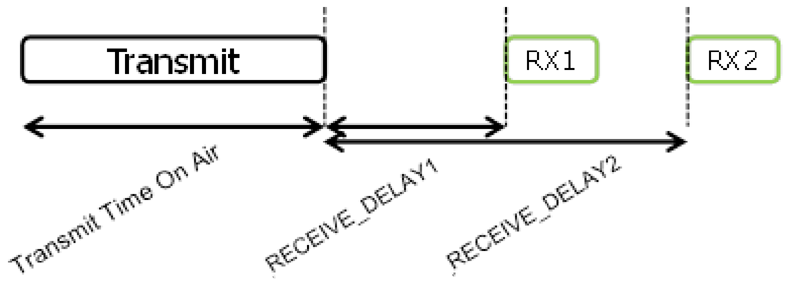


Figure 4: End-device receive slot timing.

3.3.1 First receive window channel, data rate, and start

The first receive window RX1 uses a frequency that is a function of the uplink frequency and a data rate that is a function of the data rate used for the uplink. RX1 opens RECEIVE_DELAY1[1] seconds (+/- 20 microseconds) after the end of the uplink modulation. The relationship between uplink and RX1 slot downlink data rate is region specific and detailed in the “LoRaWAN regional physical layer specification” document. By default the first receive window datarate is identical to the datarate of the last uplink.

3.3.2 Second receive window channel, data rate, and start

The second receive window RX2 uses a fixed configurable frequency and data rate and opens RECEIVE_DELAY2[1] seconds (+/- 20 microseconds) after the end of the uplink modulation. The frequency and data rate used can be modified through MAC commands (see Section 5).The default frequency and data rate to use are region specific and detailed in the “LoRaWAN regional physical layer specification” document

3.3.3 Receive window duration

The length of a receive window MUST be at least the time required by the end-device’s radio transceiver to effectively detect a downlink preamble.

3.3.4 Receiver activity during the receive windows

If a preamble is detected during one of the receive windows, the radio receiver stays active until the downlink frame is demodulated. If a frame was detected and subsequently demodulated during the first receive window and the frame was intended for this end-device after address and MIC (message integrity code) checks, the end-device does not open the second receive window.

3.3.5 Network sending a message to an end-device

If the network intends to transmit a downlink to an end-device, it will always initiate the transmission precisely at the beginning of one of those two receive windows.

[1] RECEIVE_DELAY1 and RECEIVE_DELAY2 are described in Chapter 6.

3.3.6 Important notice on receive windows

An end-device SHALL NOT transmit another uplink message before it either has received a downlink message in the first or second receive window of the previous transmission, or the second receive window of the previous transmission is expired.

3.3.7 Receiving or transmitting other protocols

The end-device may listen or transmit other protocols or do any transactions between the LoRaWAN transmission and reception windows, as long as the end-device remains compatible with the local regulation and compliant with the LoRaWAN specification.

4 MAC Message Formats

All LoRa uplink and downlink messages carry a PHY payload (**Payload**) starting with a single-octet MAC header (**MHDR**), followed by a MAC payload (**MACPayload**)[1], and ending with a 4-octet message integrity code (**MIC**).

Radio PHY layer:

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC*

Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

PHYPayload:

MHDR	MACPayload	MIC

or

MHDR	Join-Request	MIC

or

MHDR	Join-Response	MIC

Figure 6: PHY payload structure

MACPayload:

FHDR	FPort	FRMPayload

Figure 7: MAC payload structure

FHDR:

DevAddr	FCtrl	FCnt	FOpts

Figure 8: Frame header structure

Figure 9: LoRa message format elements

4.1 MAC Layer (PHYPayload)

Size (bytes)	1	1..*M*	4
PHYPayload	MHDR	MACPayload	MIC

Table 1: PHY payload format

The maximum length (*M*) of the **MACPayload** field is region specific and is specified in Chapter 6.

4.2 MAC Header (MHDR field)

Bit#	7..5	4..2	1..0
MHDR bits	MType	RFU	Major

[1] Maximum payload size is detailed in the Chapter 6.

The MAC header specifies the message type (**MType**) and according to which major version (**Major**) of the frame format of the LoRaWAN layer specification the frame has been encoded.

4.2.1 Message type (MType bit field)

The LoRaWAN distinguishes between six different MAC message types: **join request**, **join accept**, **unconfirmed data up/down**, and **confirmed data up/down**.

MType	Description
000	Join Request
001	Join Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	RFU
111	Proprietary

Table 2: MAC message types

4.2.1.1 Join-request and join-accept messages

The join-request and join-accept messages are used by the over-the-air activation procedure described in Chapter 6.2.

4.2.1.2 Data messages

Data messages are used to transfer both MAC commands and application data, which can be combined together in a single message. A **confirmed-data message** has to be acknowledged by the receiver, whereas an **unconfirmed-data message** does not require an acknowledgment.[1] **Proprietary messages** can be used to implement non-standard message formats that are not interoperable with standard messages but must only be used among devices that have a common understanding of the proprietary extensions.

Message integrity is ensured in different ways for different message types and is described per message type below.

4.2.2 Major version of data message (Major bit field)

Major bits	Description
00	LoRaWAN R1
01..11	RFU

Table 3: Major list

> **Note:** The Major version specifies the format of the messages exchanged in the join procedure (see Chapter 6.2) and the first four bytes of the MAC Payload as described in Chapter 4. For each major version, end-devices may implement different minor versions of the frame format. The minor version used by an end-device must be made known to the network server beforehand using out of band messages (e.g., as part of the device personalization information).

[1] A detailed timing diagram of the acknowledge mechanism is given in Section 18.

4.3 MAC Payload of Data Messages (MACPayload)

The MAC payload of the data messages, a so-called “data frame”, contains a frame header (**FHDR**) followed by an optional port field (**FPort**) and an optional frame payload field (**FRMPayload**).

4.3.1 Frame header (FHDR)

The **FHDR** contains the short device address of the end-device (**DevAddr**), a frame control octet (**FCtrl**), a 2-octets frame counter (**FCnt**), and up to 15 octets of frame options (**FOpts**) used to transport MAC commands.

Size (bytes)	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

For downlink frames the FCtrl content of the frame header is:

Bit#	7	6	5	4	[3..0]
FCtrl bits	ADR	RFU	ACK	FPending	FOptsLen

For uplink frames the FCtrl content of the frame header is:

Bit#	7	6	5	4	[3..0]
FCtrl bits	ADR	ADRACKReq	ACK	ClassB	FOptsLen

4.3.1.1 Adaptive data rate control in frame header (ADR, ADRACKReq in FCtrl)

LoRa network allows the end-devices to individually use any of the possible data rates. This feature is used by the LoRaWAN to adapt and optimize the data rate of static end-devices. This is referred to as Adaptive Data Rate (ADR) and when this is enabled the network will be optimized to use the fastest data rate possible.

Adaptive Data Rate control may not be possible when the radio channel attenuation changes fast and constantly. When the network is unable to control the data rate of a device , the device’s application layer should control it. It is recommended to use a variety of different data rates in this case. The application layer should always try to minimize the aggregated air time used given the network conditions.

If the **ADR** bit is set, the network will control the data rate of the end-device through the appropriate MAC commands. If the **ADR** bit is not set, the network will not attempt to control the data rate of the end-device regardless of the received signal quality. The **ADR** bit MAY be set and unset by the end-device or the Network on demand. However, whenever possible, the ADR scheme should be enabled to increase the battery life of the end-device and maximize the network capacity.

> **Note:** Even mobile end-devices are actually immobile most of the time. So depending on its state of mobility, an end-device can request the network to optimize its data rate using ADR.

If an end-device whose data rate is optimized by the network to use a data rate higher than its lowest available data rate, it periodically needs to validate that the network still receives the uplink frames. Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (**ADRACKReq**). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame resets the ADR_ACK_CNT counter. The downlink **ACK** bit does not need to be set as any response during the receive slot of the end-device indicates that the gateway has still received the uplinks from this device. If no reply is received within the next ADR_ACK_DELAY uplinks (i.e., after a total of ADR_ACK_LIMIT + ADR_ACK_DELAY), the end-device MAY try to regain connectivity by switching to the next lower data rate that provides a longer radio range. The end-device will further lower its data rate step by step every time ADR_ACK_DELAY is reached. The **ADRACKReq** SHALL not be set if the device uses its lowest available data rate because in that case no action can be taken to improve the link range.

> **Note:** Not requesting an immediate response to an ADR acknowledgement request provides flexibility to the network to optimally schedule its downlinks.

> **Note:** In uplink transmissions the **ADRACKReq** bit is set if ADR_ACK_CNT >= ADR_ACK_LIMIT and the current data-rate is greater than the device defined minimum data rate, it is cleared in other conditions.

4.3.1.2 Message acknowledge bit and acknowledgement procedure (ACK in FCtrl)

When receiving a *confirmed data* message, the receiver SHALL respond with a data frame that has the acknowledgment bit (**ACK**) set. If the sender is an end-device, the network will send the acknowledgement using one of the receive windows opened by the end-device after the send operation. If the sender is a gateway, the end-device transmits an acknowledgment at its own discretion.

Acknowledgements are only sent in response to the latest message received and are never retransmitted.

> **Note:** To allow the end-devices to be as simple as possible and have as few states as possible it may transmit an explicit (possibly empty) acknowledgement data message immediately after the reception of a data message requiring a confirmation. Alternatively the end-device may defer the transmission of an acknowledgement to piggyback it with its next data message.

4.3.1.3 Retransmission procedure

The number of retransmissions (and their timing) for the same message where an acknowledgment is requested but not received is at the discretion of the end device and may be different for each end-device.

> **Note:** Some example timing diagrams of the acknowledge mechanism are given in Chapter 18.

> **Note:** If an end-device has reached its maximum number of retransmissions without receiving an acknowledgment, it can try to re-gain connectivity by moving to a lower data rate with longer reach. It is up to the end-device to retransmit the message again or to forfeit that message and move on.

> **Note:** If the network server has reached its maximum number of retransmissions without receiving an acknowledgment, it will generally consider the end-device as unreachable until it receives further messages from the end-device. It is up to the network server to retransmit the message once connectivity to the end-device in question is regained or to forfeit that message and move on.

> **Note:** The recommended data rate back-off strategy during re-transmissions is described in Chapter 18.4

4.3.1.4 Frame pending bit (FPending in FCtrl, downlink only)

The frame pending bit (**FPending**) is only used in downlink communication, indicating that the gateway has more data pending to be sent and therefore asking the end-device to open another receive window as soon as possible by sending another uplink message.

The exact use of **FPending** bit is described in Chapter 18.3.

4.3.1.5 Frame counter (FCnt)

Each end-device has two frame counters to keep track of the number of data frames sent uplink to the network server (FCntUp), incremented by the end-device and received by the end-device downlink from the network server (FCntDown), which is incremented by the network server. The network server tracks the uplink frame counter and generates the downlink counter for each end-device. After a JoinReq – JoinAccept message exchange or a reset for a personalized end-device, the frame counters on the end-device and the frame counters on the network server for that end-device are reset to 0. Subsequently FCntUp and FCntDown are incremented at the sender side by 1 for each new data frame sent in the respective direction. At the receiver side, the corresponding counter is kept in sync with the value received provided the value received has incremented compared to the current counter value and is less than the value specified by MAX_FCNT_GAP[1] after considering counter rollovers. If this difference is greater than the value of MAX_FCNT_GAP then too many data frames have been lost then subsequent will be discarded. The FCnt is not incremented in case of multiple transmissions of an unconfirmed frame (see NbTrans parameter), or in the case of a confirmed frame that is not acknowledged.

The LoRaWAN allows the use of either 16-bits or 32-bits frame counters. The network side needs to be informed out-of-band about the width of the frame counter implemented in a given end-device. If a 16-bits frame counter is used, the **FCnt** field can be used directly as the counter value, possibly extended by leading zero octets if required. If a 32-bits frame counter is used, the **FCnt** field corresponds to the least-significant 16 bits of the 32-bits frame counter (i.e., FCntUp for data frames sent uplink and FCntDown for data frames sent downlink).

The end-device SHALL not reuse the same FCntUp value, except for retransmission, with the same application and network session keys.

[1] Actual value for MAX_FCNT_GAP, RECEIVE_DELAY1 and RECEIVE_DELAY2 can be found at **Error! Reference source not found.** for EU863-870 or **Error! Reference source not found.** for US902-928.

> **Note:** Since the **FCnt** field carries only the least-significant 16 bits of the 32-bits frame counter, the server must infer the 16 most-significant bits of the frame counter from the observation of the traffic.

4.3.1.6 Frame options (FOptsLen in FCtrl, FOpts)

The frame-options length field (**FOptsLen**) in **FCtrl** byte denotes the actual length of the frame options field (**FOpts**) included in the frame.

FOpts transport MAC commands of a maximum length of 15 octets that are piggybacked onto data frames; see Chapter 5 for a list of valid MAC commands.

If **FOptsLen** is 0, the **FOpts** field is absent. If **FOptsLen** is different from 0, i.e. if MAC commands are present in the **FOpts** field, the port 0 cannot be used (**FPort** MUST be either not present or different from 0).

MAC commands cannot be simultaneously present in the payload field and the frame options field. Should this occur, the device SHALL ignore the frame.

4.3.2 Port field (FPort)

If the frame payload field is not empty, the port field MUST be present. If present, an **FPort** value of 0 indicates that the **FRMPayload** contains MAC commands only; see Chapter 4.4 for a list of valid MAC commands. **FPort** values 1..223 (0x01..0xDF) are application-specific. FPort value 224 is dedicated to LoRaWAN Mac layer test protocol.

> Note : The purpose of Fport value 224 is to provide a dedicated Fport to run Mac compliance test scenarios over-the-air on final versions of devices, without having to rely on specific test versions of devices for practical aspects. The test is not supposed to be simultaneous with live operations, but the Mac layer implementation of the device shall be exactly the one used for the normal application. The test protocol is normally encrypted using the AppSKey. This ensures that the network cannot enable the device's test mode without involving the device's owner. If the test runs on a live network connected device, the way the test application on the network side learns the AppSkey is outside of the scope of the LoRaWAN specification. If the test runs using OTAA on a dedicated test bench (not a live network), the way the Appkey is communicated to the test bench, for secured JOIN process, is also outside of the scope of the specification.
>
> The test protocol, running at application layer, is defined outside of the LoRaWAN spec, as it is an application layer protocol.

FPort values 225..255 (0xE1..0xFF) are reserved for future standardized application extensions.

Size (bytes)	7..22	0..1	0..*N*
MACPayload	FHDR	FPort	FRMPayload

N is the number of octets of the application payload. The valid range for *N* is region specific and is defined in the "LoRaWAN regional physical layer specification" document.

N should be equal or smaller than:

$$N \leq M - 1 - (\text{length of } \mathbf{FHDR} \text{ in octets})$$

where M is the maximum MAC payload length.

4.3.3 MAC Frame Payload Encryption (FRMPayload)

If a data frame carries a payload, **FRMPayload** MUST be encrypted before the message integrity code (**MIC**) is calculated.

The encryption scheme used is based on the generic algorithm described in IEEE 802.15.4/2006 Annex B [IEEE802154] using AES with a key length of 128 bits.

The key K used depends on the FPort of the data message:

FPort	K
0	NwkSKey
1..255	AppSKey

Table 4: FPort list

The fields encrypted are:

pld = **FRMPayload**

For each data message, the algorithm defines a sequence of Blocks A_i for $i = 1..k$ with $k = \text{ceil}(\text{len}(pld) / 16)$:

Size (bytes)	1	4	1	4	4	1	1
A_i	0x01	4 x 0x00	Dir	DevAddr	FCntUp or FCntDown	0x00	i

The direction field (**Dir**) is 0 for uplink frames and 1 for downlink frames.

The blocks A_i are encrypted to get a sequence S of blocks S_i:

S_i = aes128_encrypt(K, A_i) for $i = 1..k$
$S = S_1 \mid S_2 \mid .. \mid S_k$

Encryption and decryption of the payload is done by truncating

(pld | pad_{16}) xor S

to the first len(pld) octets.

4.4 Message Integrity Code (MIC)

The message integrity code (**MIC**) is calculated over all the fields in the message.

msg = **MHDR** | **FHDR** | **FPort** | **FRMPayload**

whereby len(msg) denotes the length of the message in octets.

The **MIC** is calculated as follows [RFC4493]:

$cmac$ = aes128_cmac(**NwkSKey**, B_0 | msg)
MIC = $cmac$[0..3]

whereby the block B_0 is defined as follows:

Size (bytes)	1	4	1	4	4	1	1
B_0	0x49	4 x 0x00	Dir	DevAddr	FCntUp or FCntDown	0x00	len(msg)

The direction field (**Dir**) is 0 for uplink frames and 1 for downlink frames.

5 MAC Commands

For network administration, a set of MAC commands can be exchanged exclusively between the network server and the MAC layer on an end-device. MAC layer commands are never visible to the application or the application server or the application running on the end-device.

A single data frame can contain any sequence of MAC commands, either piggybacked in the **FOpts** field or, when sent as a separate data frame, in the **FRMPayload** field with the **FPort** field being set to 0. Piggybacked MAC commands are always sent without encryption and MUST NOT exceed 15 octets. MAC commands sent as **FRMPayload** are always encrypted and MUST NOT exceed the maximum **FRMPayload** length.

> **Note:** MAC commands whose content shall not be disclosed to an eavesdropper must be sent in the **FRMPayload** of a separate data message.

A MAC command consists of a command identifier (**CID**) of 1 octet followed by a possibly empty command-specific sequence of octets.

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x02	***LinkCheckReq***	x		Used by an end-device to validate its connectivity to a network.
0x02	***LinkCheckAns***		x	Answer to LinkCheckReq command. Contains the received signal power estimation indicating to the end-device the quality of reception (link margin).
0x03	***LinkADRReq***		x	Requests the end-device to change data rate, transmit power, repetition rate or channel.
0x03	***LinkADRAns***	x		Acknowledges the LinkRateReq.
0x04	***DutyCycleReq***		x	Sets the maximum aggregated transmit duty-cycle of a device
0x04	***DutyCycleAns***	x		Acknowledges a DutyCycleReq command
0x05	***RXParamSetupReq***		x	Sets the reception slots parameters
0x05	***RXParamSetupAns***	x		Acknowledges a RXSetupReq command
0x06	***DevStatusReq***		x	Requests the status of the end-device
0x06	***DevStatusAns***	x		Returns the status of the end-device, namely its battery level and its demodulation margin
0x07	***NewChannelReq***		x	Creates or modifies the definition of a radio channel
0x07	***NewChannelAns***	x		Acknowledges a NewChannelReq command
0x08	***RXTimingSetupReq***		x	Sets the timing of the of the reception slots
0x08	***RXTimingSetupAns***	x		Acknowledges RXTimingSetupReq command
0x09	***TxParamSetupReq***		x	Used by the network server to set the maximum allowed dwell time and Max EIRP of end-device, based on local regulations
0x09	***TxParamSetupAns***	x		Acknowledges TxParamSetupReq command
0x0A	***DlChannelReq***		x	Modifies the definition of a downlink RX1 radio channel by shifting the downlink frequency from the uplink frequencies (i.e. creating an asymmetric channel)
0x0A	***DlChannelAns***	x		Acknowledges DlChannelReq command

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x0B to 0x0C	***RFU***			
0x0D	***DeviceTimeReq***	x		Used by an end-device to request the current date and time
0x0D	***DeviceTimeAns***		x	Sent by the network, answer to the DeviceTimeReq request
0X0E to 0x7F	RFU			
0x80 to 0xFF	Proprietary	x	x	Reserved for proprietary network command extensions

Table 5: MAC commands

> **Note:** The length of a MAC command is not explicitly given and must be implicitly known by the MAC implementation. Therefore unknown MAC commands cannot be skipped and the first unknown MAC command terminates the processing of the MAC command sequence. It is therefore advisable to order MAC commands according to the version of the LoRaWAN specification which has introduced a MAC command for the first time. This way all MAC commands up to the version of the LoRaWAN specification implemented can be processed even in the presence of MAC commands specified only in a version of the LoRaWAN specification newer than that implemented.

> **Note:** Any values adjusted by the network server (e.g., RX2, new or adjusted channels definitions) remain only valid until the next join of the end-device. Therefore after each successful join procedure the end-device uses the default parameters again and it is up to the network server to re-adjust the values again as needed.

5.1 Link Check commands (*LinkCheckReq, LinkCheckAns*)

With the ***LinkCheckReq*** command, an end-device MAY validate its connectivity with the network. The command has no payload.

When a ***LinkCheckReq*** is received by the network server via one or multiple gateways, it responds with a ***LinkCheckAns*** command.

Size (bytes)	1	1
LinkCheckAns Payload	Margin	GwCnt

The demodulation margin (**Margin**) is an 8-bit unsigned integer in the range of 0..254 indicating the link margin in dB of the last successfully received ***LinkCheckReq*** command. A value of “0” means that the frame was received at the demodulation floor (0 dB or no margin) while a value of “20”, for example, means that the frame reached the gateway 20 dB above the demodulation floor. Value “255” is reserved.

The gateway count (**GwCnt**) is the number of gateways that successfully received the last ***LinkCheckReq*** command.

5.2 Link ADR commands (*LinkADRReq, LinkADRAns*)

With the ***LinkADRReq*** command, the network server requests an end-device to perform a rate adaptation.

Size (bytes)	1	2	1
LinkADRReq Payload	DataRate_TXPower	ChMask	Redundancy

Bits	[7:4]	[3:0]
DataRate_TXPower	DataRate	TXPower

The requested date rate (**DataRate**) and TX output power (**TXPower**) are region-specific and are encoded as indicated in the "LoRaWAN regional physical layer specification" document. The TX output power indicated in the command is to be considered the maximum transmit power the device may operate at. An end-device will acknowledge as successful a command which specifies a higher transmit power than it is capable of using and should, in that case, operate at its maximum possible power. The channel mask (**ChMask**) encodes the channels usable for uplink access as follows with bit 0 corresponding to the LSB:

Bit#	**Usable channels**
0	Channel 1
1	Channel 2
..	..
15	Channel 16

Table 6: Channel state table

A bit in the **ChMask** field set to 1 means that the corresponding channel can be used for uplink transmissions if this channel allows the data rate currently used by the end-device. A bit set to 0 means the corresponding channels should be avoided.

Bits	7	[6:4]	[3:0]
Redundancy bits	RFU	ChMaskCntl	NbTrans

In the Redundancy bits the **NbTrans** field is the number of transmissions for each uplink message. This applies only to "unconfirmed" uplink frames. The default value is 1 corresponding to a single transmission of each frame. The valid range is [1:15]. If **NbTrans**==0 is received the end-device should use the default value. This field can be used by the network manager to control the redundancy of the node uplinks to obtain a given Quality of Service. The end-device performs frequency hopping as usual between repeated transmissions, it does wait after each repetition until the receive windows have expired. . Whenever a downlink message is received during the RX1 slot window, it SHALL stop any further retransmission of the same uplink message. For class A devices, a reception in the RX2 slot has the same effect.

The channel mask control (**ChMaskCntl**) field controls the interpretation of the previously defined **ChMask** bit mask. It controls the block of 16 channels to which the **ChMask** applies. It can also be used to globally turn on or off all channels using specific modulation. This field usage is region specific and is defined in the "LoRaWAN regional physical layer specification" document.

The network server MAY include multiple LinkAdrReq commands within a single downlink message. For the purpose of configuring the end-device channel mask, the end-device will process all contiguous LinkAdrReq messages, in the order present in the downlink message, as a single atomic block command. The end-device will accept or reject all Channel Mask controls in the contiguous block, and provide consistent Channel Mask ACK status indications for each command in the contiguous block in each LinkAdrAns message, reflecting the acceptance or rejection of this atomic channel mask setting. The device will only process the DataRate, TXPower and NbTrans from the last message in the contiguous block, as these settings govern the end-device global state for these values. The end-device will provide consistent ACK status in each LinkAdrAns message reflecting the acceptance or rejection of these final settings.

The channel frequencies are region-specific and they are defined in Chapter 6. An end-device answers to a ***LinkADRReq*** with a ***LinkADRAns*** command.

Size (bytes)	1
LinkADRAns Payload	Status

Bits	[7:3]	2	1	0
Status bits	RFU	Power ACK	Data rate ACK	Channel mask ACK

The ***LinkADRAns* Status** bits have the following meaning:

	Bit = 0	**Bit = 1**
Channel mask ACK	The channel mask sent enables a yet undefined channel or the channel mask required all channels to be disabled. The command was discarded and the end-device state was not changed.	The channel mask sent was successfully interpreted. All currently defined channel states were set according to the mask.
Data rate ACK	The data rate requested is unknown to the end-device or is not possible given the channel mask provided (not supported by any of the enabled channels). The command was discarded and the end-device state was not changed.	The data rate was successfully set.
Power ACK	The requested power level is not implemented in the device. The command was discarded and the end-device state was not changed.	The power level was successfully set.

Table 7: LinkADRAns status bits signification

If any of those three bits equals 0, the command did not succeed and the node has kept the previous state.

5.3 End-Device Transmit Duty Cycle (*DutyCycleReq*, *DutyCycleAns*)

The ***DutyCycleReq*** command is used by the network coordinator to limit the maximum aggregated transmit duty cycle of an end-device. The aggregated transmit duty cycle corresponds to the transmit duty cycle over all sub-bands.

Size (bytes)	1
DutyCycleReq Payload	DutyCyclePL

Bits	7:4	3:0
DutyCyclePL	RFU	MaxDCycle

The maximum end-device transmit duty cycle allowed is:

$$aggregated\ duty\ cycle = \frac{1}{2^{\mathrm{MaxDCycle}}}$$

The valid range for **MaxDutyCycle** is [0 : 15]. A value of 0 corresponds to “no duty cycle limitation” except the one set by the regional regulation.

An end-device answers to a ***DutyCycleReq*** with a ***DutyCycleAns*** command. The ***DutyCycleAns*** MAC reply does not contain any payload.

5.4 Receive Windows Parameters (*RXParamSetupReq*, *RXParamSetupAns*)

The ***RXParamSetupReq*** command allows a change to the frequency and the data rate set for the second receive window (RX2) following each uplink. The command also allows to program an offset between the uplink and the RX1 slot downlink data rates.

Size (bytes)	1	3
RXParamSetupReq Payload	DLsettings	Frequency

Bits	7	6:4	3:0
DLsettings	RFU	RX1DRoffset	RX2DataRate

The RX1DRoffset field sets the offset between the uplink data rate and the downlink data rate used to communicate with the end-device on the first reception slot (RX1). As a default this offset is 0. The offset is used to take into account maximum power density constraints for base stations in some regions and to balance the uplink and downlink radio link margins.

The data rate (RX2**DataRate**) field defines the data rate of a downlink using the second receive window following the same convention as the ***LinkADRReq*** command (0 means DR0/125kHz for example). The frequency (**Frequency**) field corresponds to the frequency of the channel used for the second receive window, whereby the frequency is coded following the convention defined in the ***NewChannelReq*** command.

The ***RXParamSetupAns*** command is used by the end-device to acknowledge the reception of ***RXParamSetupReq*** command. The ***RXParamSetupAns*** command should be added in the FOpt field of all uplinks until a class A downlink is received by the end-device. This guarantees that even in presence of uplink packet loss, the network is always aware of the downlink parameters used by the end-device.

The payload contains a single status byte.

Size (bytes)	1
RXParamSetupAns Payload	Status

The status (**Status**) bits have the following meaning.

Bits	7:3	2	1	0
Status bits	RFU	RX1DRoffset ACK	RX2 Data rate ACK	Channel ACK

	Bit = 0	**Bit = 1**
Channel ACK	The frequency requested is not usable by the end-device.	RX2 slot channel was successfully set
RX2 Data rate ACK	The data rate requested is unknown to the end-device.	RX2 slot data rate was successfully set
RX1DRoffset ACK	the uplink/downlink data rate offset for RX1 slot is not in the allowed range	RX1DRoffset was successfully set

Table 8: RX2SetupAns status bits signification

If either of the 3 bits is equal to 0, the command did not succeed and the previous parameters are kept.

5.5 End-Device Status (*DevStatusReq*, *DevStatusAns*)

With the ***DevStatusReq*** command a network server may request status information from an end-device. The command has no payload. If a ***DevStatusReq*** is received by an end-device, it responds with a ***DevStatusAns*** command.

Size (bytes)	1	1
DevStatusAns Payload	Battery	Margin

The battery level (**Battery**) reported is encoded as follows:

Battery	**Description**
0	The end-device is connected to an external power source.
1..254	The battery level, 1 being at minimum and 254 being at maximum
255	The end-device was not able to measure the battery level.

Table 9: Battery level decoding

The margin (**Margin**) is the demodulation signal-to-noise ratio in dB rounded to the nearest integer value for the last successfully received ***DevStatusReq*** command. It is a signed integer of 6 bits with a minimum value of -32 and a maximum value of 31.

Bits	7:6	5:0
Status	RFU	Margin

5.6 Creation / Modification of a Channel (*NewChannelReq*, *NewChannelAns, DlChannelReq, DlChannelAns*)

The ***NewChannelReq*** command can be used to either modify the parameters of an existing bidirectional channel or to create a new one. The command sets the center frequency of the new channel and the range of uplink data rates usable on this channel:

Size (bytes)	1	3	1
NewChannelReq Payload	ChIndex	Freq	DrRange

The channel index (**ChIndex**) is the index of the channel being created or modified. Depending on the region and frequency band used, the LoRaWAN specification imposes default channels which MUST be common to all devices and cannot be modified by the ***NewChannelReq*** command (cf. Chapter 6). If the number of default channels is *N*, the default channels go from 0 to *N*-1, and the acceptable range for **ChIndex** is *N* to 15. A device MUST be able to handle at least 16 different channel definitions. In certain region the device may have to store more than 16 channel definitions.

The frequency (**Freq**) field is a 24 bits unsigned integer. The actual channel frequency in Hz is 100 x **Freq** whereby values representing frequencies below 100 MHz are reserved for future use. This allows setting the frequency of a channel anywhere between 100 MHz to 1.67 GHz in 100 Hz steps. A **Freq** value of 0 disables the channel. The end-device has to check that the frequency is actually allowed by its radio hardware and return an error otherwise.

The data-rate range (**DrRange**) field specifies the uplink data-rate range allowed for this channel. The field is split in two 4-bit indexes:

Bits	7:4	3:0
DrRange	MaxDR	MinDR

Following the convention defined in Section 5.2 the minimum data rate (**MinDR**) subfield designate the lowest uplink data rate allowed on this channel. For example 0 designates DR0 / 125 kHz. Similarly, the maximum data rate (**MaxDR**) designates the highest uplink data rate. For example, DrRange = 0x77 means that only 50 kbps GFSK is allowed on a channel and DrRange = 0x50 means that DR0 / 125 kHz to DR5 / 125 kHz are supported.

The newly defined or modified channel is enabled and can immediately be used for communication. The RX1 downlink frequency is set equal to the uplink frequency.

The end-device acknowledges the reception of a ***NewChannelReq*** by sending back a ***NewChannelAns*** command. The payload of this message contains the following information:

Size (bytes)	1
NewChannelAns Payload	Status

The status (**Status**) bits have the following meaning:

Bits	7:2	1	0
Status	RFU	Data rate range ok	Channel frequency ok

	Bit = 0	**Bit = 1**
Data rate range ok	The designated data rate	The data rate range is

	range exceeds the ones currently defined for this end-device	compatible with the possibilities of the end-device
Channel frequency ok	The device cannot use this frequency	The device is able to use this frequency.

Table 10: NewChannelAns status bits signification

If either of those 2 bits equals 0, the command did not succeed and the new channel has not been created.

The ***DlChannelReq*** command allows the network to associate a different downlink frequency to the RX1 slot. This command is applicable for all the physical layer specifications supporting the ***NewChannelReq*** command (for example EU and China physical layers, but not for US or Australia). In regions where that command is not defined, the device SHALL silently drop it.

The command sets the center frequency used for the downlink RX1 slot, as follows:

Size (bytes)	1	3
DlChannelReq Payload	ChIndex	Freq

The channel index (**ChIndex**) is the index of the channel whose downlink frequency is modified

The frequency (**Freq**) field is a 24 bits unsigned integer. The actual downlink frequency in Hz is 100 x **Freq** whereby values representing frequencies below 100 MHz are reserved for future use. The end-device has to check that the frequency is actually allowed by its radio hardware and return an error otherwise.

If the ***DlChannelReq*** command is defined in the region where the end-device is operating, the end-device acknowledges the reception of a ***DlChannelReq*** by sending back a ***DlChannelAns*** command. The ***DlChannelAns*** command SHALL be added in the FOpt field of all uplinks until a downlink packet is received by the end-device. This guarantees that even in presence of uplink packet loss, the network is always aware of the downlink frequencies used by the end-device.

The payload of this message contains the following information:

Size (bytes)	1
DlChannelAns Payload	Status

The status (**Status**) bits have the following meaning:

Bits	7:2	1	0
Status	RFU	Uplink frequency exists	Channel frequency ok

	Bit = 0	**Bit = 1**
Channel frequency ok	The device cannot use this frequency	The device is able to use this frequency.

Uplink frequency exists	The uplink frequency is not defined for this channel , the downlink frequency can only be set for a channel that already has a valid uplink frequency	The uplink frequency of the channel is valid

Table 11: DlChannelAns status bits signification

If either of those 2 bits equals 0, the command did not succeed and the RX1 slot channel frequency has not been updated.

5.7 Setting delay between TX and RX (*RXTimingSetupReq*, *RXTimingSetupAns*)

The ***RXTimingSetupReq*** command allows configuring the delay between the end of the TX uplink and the opening of the first reception slot. The second reception slot opens one second after the first reception slot.

Size (bytes)	1
RXTimingSetupReq Payload	Settings

The delay (**Delay**) field specifies the delay. The field is split in two 4-bit indexes:

Bits	7:4	3:0
Settings	RFU	Del

The delay is expressed in seconds. **Del** 0 is mapped on 1 s.

Del	**Delay [s]**
0	1
1	1
2	2
3	3
..	..
15	15

Table 12: Del mapping table

An end device answers ***RXTimingSetupReq*** with ***RXTimingSetupAns*** with no payload.

The ***RXTimingSetupAns*** command should be added in the FOpt field of all uplinks until a class A downlink is received by the end-device. This guarantees that even in presence of uplink packet loss, the network is always aware of the downlink parameters used by the end-device.

5.8 End-device transmission parameters (*TxParamSetupReq*, *TxParamSetupAns*)

This MAC command only needs to be implemented for compliance in certain regulatory regions. Please refer to the “LoRaWAN physical layer definitions” document.

The ***TxParamSetupReq*** command can be used to notify the end-device of the maximum allowed dwell time, i.e. the maximum continuous transmission time of a packet over the air, as well as the maximum allowed end-device Effective Isotropic Radiated Power (EIRP).

Size (bytes)	1
TxParamSetup payload	EIRP_DwellTime

The structure of EIRP_DwellTime field is described below:

Bits	7:6	5	4	3:0
MaxDwellTime	RFU	DownlinkDwellTime	UplinkDwellTime	MaxEIRP

Bits [0…3] of ***TxParamSetupReq*** command are used to encode the Max EIRP value, as per the following table. The EIRP values in this table are chosen in a way that covers a wide range of max EIRP limits imposed by the different regional regulations.

Coded Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Max EIRP (dBm)	8	10	12	13	14	16	18	20	21	24	26	27	29	30	33	36

The maximum EIRP corresponds to an upper bound on the device's radio transmit power. The device is not required to transmit at that power , but SHALL never radiate more that this specified EIRP.

Bits 4 and 5 define the maximum Uplink and downlink dwell time respectively, which is encoded as per the following table:

Coded Value	**Dwell Time**
0	No Limit
1	400 ms

When this MAC command is implemented (region specific), the end-device acknowledges the TxParamSetupReq command by sending a ***TxParamSetupAns*** command. This ***TxParamSetupAns*** command doesn't contain any payload.

When this MAC command is used in a region where it is not required, the device does not process it and SHALL NOT transmit an acknowledgement.

5.9 DeviceTime commands (DeviceTimeReq, DeviceTimeAns)

This MAC command is only available if the device is activated on a LoRaWAN1.0.3 or later compatible Network Server. LoRaWAN1.0.2 or earlier servers do not implement this MAC command.

With the ***DeviceTimeReq*** command, an end-device MAY request from the network the current network time. This allows the device to synchronize its internal clock to the network's clock. This is specifically useful to speed-up the acquisition of the class B beacon. The request has no payload.

With the ***DeviceTimeAns*** command, the Network Server provides the network time to the end device. The time provided is the network time captured at the end of the uplink transmission. The command has a 5 bytes payload defined as follows:

Size (bytes)	4	1
DeviceTimeAns Payload	32-bit unsigned integer : Seconds since epoch*	8bits unsigned integer: fractional-second in ½^8 second steps

Figure 10 : DeviceTimeAns payload format

The time provided by the network MUST have a worst case accuracy of +/-100mSec. The ***DeviceTimeAns*** command MUST be sent as a class A downlink (i.e. over the RX1/RX2 receive windows of the Class A mode). (*) The GPS epoch (i.e Sunday January the 6th 1980 at 00:00:00 UTC) is used as origin. The “seconds” field is the number of seconds elapsed since the origin. This field is monotonically increasing by 1 every second. To convert this field to UTC time, the leap seconds must be taken into account.

> Example: Friday 12th of February 2016 at 14:24:31 UTC corresponds to 1139322288 seconds since GPS epoch. As of June 2017, the GPS time is 17seconds ahead of UTC time.

6 End-Device Activation

To participate in a LoRaWAN network, each end-device has to be personalized and activated.

Activation of an end-device can be achieved in two ways, either via **Over-The-Air Activation** (OTAA) when an end-device is deployed or reset, or via **Activation By Personalization** (ABP) in which the two steps of end-device personalization and activation are done as one step.

6.1 Data Stored in the End-device after Activation

After activation, the following information is stored in the end-device: a device address (**DevAddr**), an application identifier (**AppEUI**), a network session key (**NwkSKey**), and an application session key (**AppSKey**).

6.1.1 End-device address (DevAddr)

The **DevAddr** consists of 32 bits identifies the end-device within the current network. Its format is as follows:

Bit#	[31..25]	[24..0]
DevAddr bits	NwkID	NwkAddr

The most significant 7 bits are used as network identifier (**NwkID**) to separate addresses of territorially overlapping networks of different network operators and to remedy roaming issues. The least significant 25 bits, the network address (**NwkAddr**) of the end-device, can be arbitrarily assigned by the network manager.

6.1.2 Application identifier (AppEUI)

The **AppEUI** is a global application ID in IEEE EUI64 address space that uniquely identifies the entity able to process the JoinReq frame.

The **AppEUI** is stored in the end-device before the activation procedure is executed.

6.1.3 Network session key (NwkSKey)

The **NwkSKey** is a **network session key** specific for the end-device. It is used by both the network server and the end-device to calculate and verify the **MIC** (message integrity code) of all data messages to ensure data integrity. It is further used to encrypt and decrypt the payload field of a MAC-only data messages.

6.1.4 Application session key (AppSKey)

The **AppSKey** is an **application session key** specific for the end-device. It is used by both the application server and the end-device to encrypt and decrypt the payload field of application-specific data messages. Application payloads are end-to-end encrypted between the end-device and the application server, but they are not integrity protected. That means, a network server may be able to alter the content of the data messages in transit. Network servers are considered as trusted, but applications wishing to implement end-to-end confidentiality and integrity protection are recommended to use additional end-to-end security solutions, which are beyond the scope of this specification.

6.2 Over-the-Air Activation

For over-the-air activation, end-devices must follow a join procedure prior to participating in data exchanges with the network server. An end-device has to go through a new join procedure every time it has lost the session context information.

The join procedure requires the end-device to be personalized with the following information before its starts the join procedure: a globally unique end-device identifier (**DevEUI**), the application identifier (**AppEUI**), and an AES-128 key (**AppKey**).

The **AppEUI** is described above in 6.1.2.

> **Note:** For over-the-air-activation, end-devices are not personalized with any kind of network key. Instead, whenever an end-device joins a network, a network session key specific for that end-device is derived to encrypt and verify transmissions at the network level. This way, roaming of end-devices between networks of different providers is facilitated. Using both a network session key and an application session key further allows federated network servers in which application data cannot be read or tampered with by the network provider.

6.2.1 End-device identifier (DevEUI)

The **DevEUI** is a global end-device ID in IEEE EUI64 address space that uniquely identifies the end-device.

6.2.2 Application key (AppKey)

The **AppKey** is an AES-128 root key specific to the end-device.[1] Whenever an end-device joins a network via over-the-air activation, the AppKey is used to derive the session keys NwkSKey and AppSKey specific for that end-device to encrypt and verify network communication and application data.

6.2.3 Join procedure

From an end-device's point of view, the join procedure consists of two MAC messages exchanged with the server, namely a **join request** and a **join accept**.

6.2.4 Join-request message

The join procedure is always initiated from the end-device by sending a join-request message.

Size (bytes)	8	8	2
Join Request	AppEUI	DevEUI	DevNonce

The join-request message contains the **AppEUI** and **DevEUI** of the end-device followed by a **nonce** of 2 octets (**DevNonce**).

1. Since all end-devices end up with unrelated application keys specific for each end-device, extracting the AppKey from an end-device only compromises this one end-device.

DevNonce is a random value.[1] For each end-device, the network server keeps track of a certain number of **DevNonce** values used by the end-device in the past, and ignores join requests with any of these **DevNonce** values from that end-device.

> **Note:** This mechanism prevents replay attacks by sending previously recorded join-request messages with the intention of disconnecting the respective end-device from the network. Any time the network server processes a Join-Request and generates a Join-accept frame, it shall maintain both the old security context (keys and counters, if any) and the new one until it receives the first successful uplink frame using the new context, after which the old context can be safely removed. This provides defense against an adversary replaying an earlier Join-request using a DevNonce that falls outside the finite list of values tracked by the network server.

The message integrity code (**MIC**) value (see Chapter 4 for MAC message description) for a join-request message is calculated as follows:[2]

cmac = aes128_cmac(AppKey, MHDR | AppEUI | DevEUI | DevNonce)
MIC = *cmac*[0..3]

The join-request message is not encrypted.

The join-request message can be transmitted using any data rate and following a random frequency hopping sequence across the specified join channels. It is recommended to use a plurality of data rates. The intervals between transmissions of **Join-Requests** SHALL respect the condition described in chapter 7.

6.2.5 Join-accept message

The network server will respond to the **join-request** message with a **join-accept** message if the end-device is permitted to join a network. The join-accept message is sent like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2, respectively). The channel frequency and data rate used for these two receive windows are identical to the one used for the RX1 and RX2 receive windows described in the “receive windows” section of the “LoRaWAN regional physical layer specification” document

No response is given to the end-device if the join request is not accepted.

The join-accept message contains an application nonce (**AppNonce**) of 3 octets, a network identifier (**NetID**), an end-device address (**DevAddr**), a delay between TX and RX (**RxDelay**) and an optional list of channel frequencies (**CFList**) for the network the end-device is joining. The CFList option is region specific and is defined in the “LoRaWAN regional physical layer specification” document.

Size (bytes)	3	3	4	1	1	(16) Optional
Join Accept	AppNonce	NetID	DevAddr	DLSettings	RxDelay	CFList

[1] The DevNonce can be extracted by issuing a sequence of RSSI measurements under the assumption that the quality of randomness fulfills the criteria of true randomness
[2] [RFC4493]

The **AppNonce** is a random value or some form of unique ID provided by the network server and used by the end-device to derive the two session keys **NwkSKey** and **AppSKey** as follows:[1]

NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad_{16})
AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad_{16})

The MIC value for a join-accept message is calculated as follows:[2]

cmac = aes128_cmac(AppKey,
MHDR | AppNonce | NetID | DevAddr | DLSettings | RxDelay | CFList)
MIC = *cmac*[0..3]

The join-accept message itself is encrypted with the **AppKey** as follows:
aes128_decrypt(AppKey, AppNonce | NetID | DevAddr | DLSettings | RxDelay | CFList | MIC)

> **Note:** The network server uses an AES decrypt operation in ECB mode to encrypt the join-accept message so that the end-device can use an AES encrypt operation to decrypt the message. This way an end-device only has to implement AES encrypt but not AES decrypt.

> **Note:** Establishing these two session keys allows for a federated network server infrastructure in which network operators are not able to eavesdrop on application data. In such a setting, the application provider must support the network operator in the process of an end-device actually joining the network and establishing the NwkSKey for the end-device. At the same time the application provider commits to the network operator that it will take the charges for any traffic incurred by the end-device and retains full control over the AppSKey used for protecting its application data.

The format of the **NetID** is as follows: The seven LSB of the **NetID** are called **NwkID** and match the seven MSB of the short address of an end-device as described before. Neighboring or overlapping networks must have different **NwkID**s. The remaining 17 MSB can be freely chosen by the network operator.

The DLsettings field contains the downlink configuration:

Bits	7	6:4	3:0
DLsettings	RFU	RX1DRoffset	RX2 Data rate

The RX1DRoffset field sets the offset between the uplink data rate and the downlink data rate used to communicate with the end-device on the first reception slot (RX1). By default this offset is 0.. The offset is used to take into account maximum power density constraints for base stations in some regions and to balance the uplink and downlink radio link margins.

The actual relationship between the uplink and downlink data rate is region specific and detailed in the “Physical Layer” section

The delay **RxDelay** follows the same convention as the **Delay** field in the ***RXTimingSetupReq*** command.

[1] The pad_{16} function appends zero octets so that the length of the data is a multiple of 16.
[2] [RFC4493]

6.3 Activation by Personalization

Under certain circumstances, end-devices can be activated by personalization. Activation by personalization directly ties an end-device to a specific network by-passing the **join request** - **join accept** procedure.

Activating an end-device by personalization means that the **DevAddr** and the two session keys **NwkSKey** and **AppSKey** are directly stored into the end-device instead of the **DevEUI, AppEUI** and the **AppKey**. The end-device is equipped with the required information for participating in a specific LoRa network when started.

Each device should have a unique set of NwkSKey and AppSKey. Compromising the keys of one device shouldn't compromise the security of the communications of other devices. The process to build those keys should be such that the keys cannot be derived in any way from publicly available information (like the node address for example).

7 Retransmissions back-off

Uplink frames that:

- Require an **acknowledgement or an anwser** by the network or an application server, and are **retransmitted** by the device if the acknowledgement or answer is not received.

and

- can be triggered by an **external** event causing **synchronization** across a large (>100) number of devices (power outage, radio jamming, network outage, earthquake…)

can trigger a catastrophic, self-persisting, radio network overload situation.

> Note: An example of such uplink frame is typically the JoinRequest if the implementation of a group of end-devices decides to reset the MAC layer in the case of a network outage.
>
> The whole group of end-device will start broadcasting JoinRequest uplinks and will only stops when receiving a JoinResponse from the network.

For those frame retransmissions, the interval between the end of the RX2 slot and the next uplink retransmission SHALL be random and follow a different sequence for every device (For example using a pseudo-random generator seeded with the device's address) .The transmission duty-cycle of such message SHALL respect the local regulation and the following limits, whichever is more constraining:

Aggregated during the first hour following power-up or reset	T0<t<T0+1	Transmit time < 36Sec
Aggregated during the next 10 hours	T0+1<t<T0+11	Transmit time < 36Sec
After the first 11 hours , aggregated over 24h	T0+11+N<t<T0+35+N N>=0	Transmit time < 8.7Sec per 24h

CLASS B – BEACON

8 Introduction to Class B

This section describes the LoRaWAN Class B layer which is optimized for battery-powered end-devices that may be either mobile or mounted at a fixed location.

End-devices should implement Class B operation when there is a requirement to open receive windows at fixed time intervals for the purpose of enabling server initiated downlink messages.

LoRaWAN Class B option adds a synchronized reception window on the end-device.

One of the limitations of LoRaWAN Class A is the Aloha method of sending data from the end-device; it does not allow for a known reaction time when the customer application or the server wants to address the end-device. The purpose of Class B is to have an end-device available for reception at a predictable time, in addition to the reception windows that follows the random uplink transmission from the end-device of Class A. Class B is achieved by having the gateway sending a beacon on a regular basis to synchronize all end-devices in the network so that the end-device can open a short additional reception window (called "ping slot") at a predictable time during a periodic time slot.

> **Note:** The decision to switch from Class A to Class B comes from the application layer of the end-device. If this class A to Class B switch needs to be controlled from the network side, the customer application must use one of the end-device's Class A uplinks to send back a downlink to the application layer, and it needs the application layer on the end-device to recognize this request – this process is not managed at the LoRaWAN level.

9 Principle of synchronous network initiated downlink (Class-B option)

For a network to support end-devices of Class B, all gateways MAY synchronously broadcast a beacon providing a timing reference to the end-devices. Based on this timing reference the end-devices can periodically open receive windows, hereafter called "ping slots", which can be used by the network infrastructure to initiate a downlink communication. A network initiated downlink using one of these ping slots is called a "ping". The gateway chosen to initiate this downlink communication is selected by the Network Server.. For this reason, if an end-device moves and detects a change in the identity advertised in the received beacon, it must send an uplink to the Network Server so that the server can update the downlink routing path database.

Before a device can operate in Class B mode, the following informations MUST be made available to the Network Server out-of-band.

- The device's default ping-slot periodicity
- Default Ping-slot data rate
- Default Ping-slot channel

All end-devices start and join the network as end-devices of Class A. The end-device application can then decide to switch to Class B. This is done through the following process:

- The end-device application requests the LoRaWAN layer to switch to Class B mode. The LoRaWAN layer in the end-device searches for a beacon and returns either a BEACON_LOCKED service primitive to the application if a network beacon was found and locked or a BEACON_NOT_FOUND service primitive. To accelerate the beacon discovery the LoRaWAN layer MAY use the “DeviceTimeReq” MAC command.
- Once in Class B mode, the MAC layer sets to 1 the *Class B* bit of the FCTRL field of every uplink frame transmitted. This bit signals to the server that the device has switched to Class B. The MAC layer will autonomously schedule a reception slot for each beacon and each ping slot. When the beacon reception is successful the end-device LoRaWAN layer forwards the beacon content to the application together with the measured radio signal strength. The end-device LoRaWAN layer takes into account the maximum possible clock drift in the scheduling of the beacon reception slot and ping slots. When a downlink is successfully demodulated during a ping slot, it is processed similarly to a downlink as described in the LoRaWAN Class A specification.
- A mobile end-device should periodically inform the Network Server of its location to update the downlink route. This is done by transmitting a normal (possibly empty) “unconfirmed” or “confirmed” uplink. The end-device LoRaWAN layer will appropriately set the *Class B* bit to 1 in the frame’s FCtrl field. Optimally this can be done more efficiently if the application detects that the node is moving by analyzing the beacon content. In that case the end-device MUST apply a random delay (as defined in Section 15.5 between the beacon reception and the uplink transmission to avoid systematic uplink collisions.
- At any time the Network Server MAY change the device’s ping-slot downlink frequency or data rate by sending a PingSlotChannelReq MAC command.
- The device MAY change the periodicity of its ping-slots at any time. To do so, it MUST temporarily stop class B operation (unset class B bit in its uplink frames) and send a PingSlotInfoReq to the Network Server. Once this command is acknowledged the device MAY restart class B operation with the new ping-slot periodicity
- If no beacon has been received for a given period (as defined in Section 12.2), the synchronization with the network is lost. The MAC layer should inform the application layer that it has switched back to Class A. As a consequence the end-device LoRaWAN layer stops setting the *Class B* bit in all uplinks and this informs the Network Server that the end-device is no longer in Class B mode. The end-device application can try to switch back to Class B periodically. This will restart this process starting with a beacon search.

The following diagram illustrates the concept of beacon reception slots and ping slots.

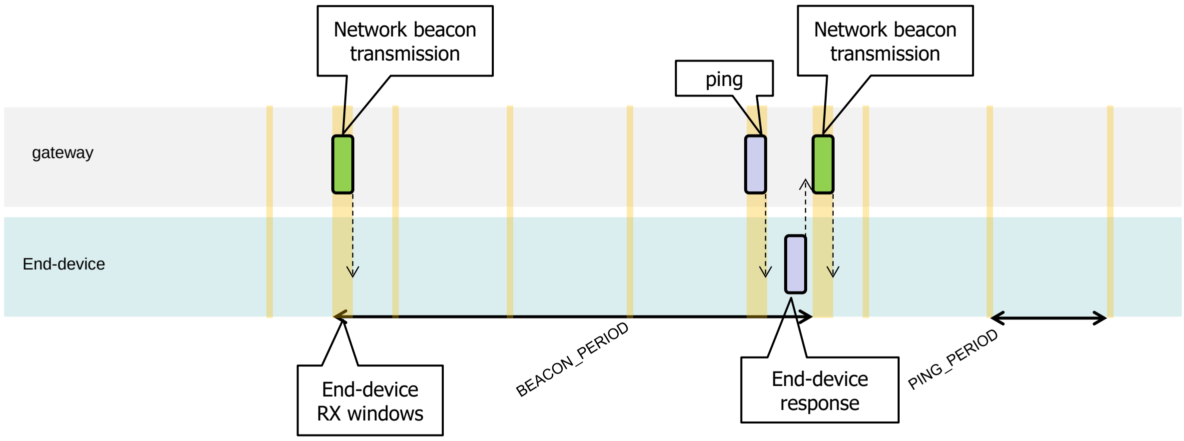


Figure 11: Beacon reception slot and ping slots

In this example, given the beacon period is 128 s, the end-device also opens a ping reception slot every 32 s. Most of the time this ping slot is not used by the server and therefore the end-device reception window is closed as soon as the radio transceiver has assessed that no preamble is present on the radio channel. If a preamble is detected the radio transceiver will stay on until the downlink frame is demodulated. The MAC layer will then process the frame, check that its address field matches the end-device address and that the Message Integrity Check is valid before forwarding it to the application layer.

10 Uplink frame in Class B mode

The uplink frames in Class B mode are same as the Class A uplinks with the exception of the RFU bit in the FCtrl field in the Frame header. In the Class A uplink this bit is unused (RFU). This bit is used for Class B uplinks.

Bit#	7	6	5	4	3..0
FCtrl	ADR	ADRACKReq	ACK	Class B	FOptsLen

Figure 12 : class B FCtrl fields

The *Class B* bit set to 1 in an uplink signals the Network Server that the device as switched to Class B mode and is now ready to receive scheduled downlink pings.

The signification of the FPending bit for downlink is unaltered and still signals that one or more downlink frames are queued for this device in the server.

11 Downlink Ping frame format (Class B option)

11.1 Physical frame format

A downlink Ping uses the same format as a Class A downlink frame but might follow a different channel frequency or data rate plan.

11.2 Unicast & Multicast MAC messages

Messages can be “unicast” or “multicast”. Unicast messages are sent to a single end-device and multicast messages are sent to multiple end-devices. All devices of a multicast group MUST share the same multicast address and associated encryption keys. The LoRaWAN Class B specification does not specify means to remotely setup such a multicast group or securely distribute the required multicast key material. This must either be performed during the node personalization or through the application layer.

> Example: The document [RPD1] describes a possible application layer mechanism for over-the-air multicast key distribution.

11.2.1 Unicast MAC message format

The MAC payload of a unicast downlink **Ping** uses the format defined in the Class A specification. It is processed by the end-device in exactly the same way. The same frame counter is used and incremented whether the downlink uses a Class B ping slot or a Class A downlink slot.

11.2.2 Multicast MAC message format

The Multicast frames share most of the unicast frame format with a few exceptions:

- They are not allowed to carry MAC commands, neither in the **FOpt** field, nor in the payload on port 0 because a multicast downlink does not have the same authentication robustness as a unicast frame.
- The **ACK** and **ADRACKReq** bits MUST be zero. The **MType** field MUST carry the value for Unconfirmed Data Down.
- The **FPending** bit indicates there is more multicast data to be sent. If it is set the next multicast receive slot will carry a data frame. If it is not set the next slot may or may not carry data. This bit can be used by end-devices to evaluate priorities for conflicting reception slots.

12 Beacon acquisition and tracking

Before switching from Class A to Class B, the end-device MUST first receive one of the network beacons to align his internal timing reference with the network.

Once in Class B, the end-device MUST periodically search and receive a network beacon to cancel any drift of its internal clock time base, relative to the network timing.

A Class B device may be temporarily unable to receive beacons (out of range from the network gateways, presence of interference, ..). In this event, the end-device has to gradually widen its beacon and ping slots reception windows to take into account a possible drift of its internal clock.

> **Note:** For example, a device which internal clock is defined with a +/-10ppm precision may drift by +/-1.3mSec every beacon period.

12.1 Minimal beacon-less operation time

In the event of beacon loss, a device SHALL be capable of maintaining Class B operation for 2 hours (120 minutes) after it received the last beacon. This temporary Class B operation without beacon is called "beacon-less" operation. It relies on the end-device's own clock to keep timing.

During beacon-less operation, Class B unicast, multicast and beacon reception slots MUST all be progressively expanded to accommodate the end-device's possible clock drift.

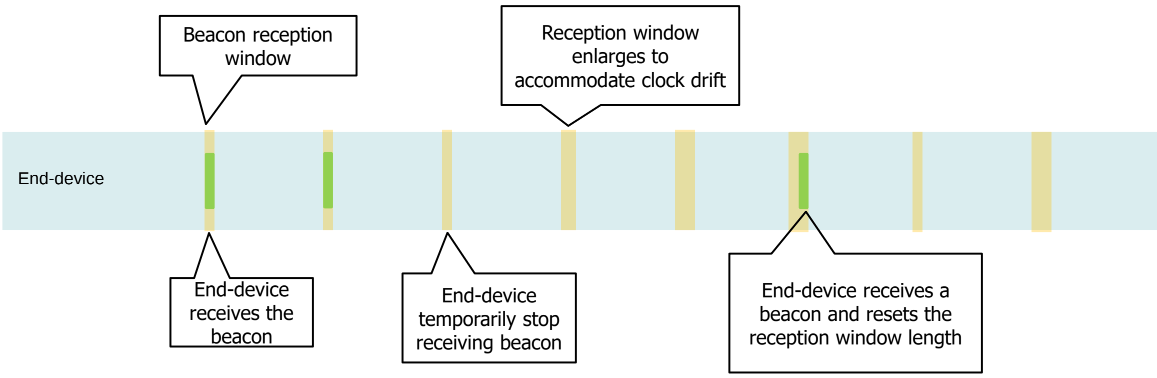


Figure 13 : beacon-less temporary operation

12.2 Extension of beacon-less operation upon reception

During this 120 minutes time interval the reception of any beacon directed to the end-device, should extend the Class B beacon-less operation further by another 120 minutes as it allows to correct any timing drift and reset the receive slots duration.

> Note: Device should also use classB pingSlots downlinks to resynchronize its' internal clock.

12.3 Minimizing timing drift

The end-devices MAY use the beacon's (when available) precise periodicity to calibrate their internal clock and therefore reduce the initial clock frequency imprecision. As the timing oscillator’s exhibit a predictable temperature frequency shift, the use of a temperature sensor could enable further minimization of the timing drift.

13 Class B Downlink slot timing

13.1 Definitions

To operate successfully in Class B the end-device MUST open reception slots at precise instants relative to the infrastructure beacon. This section defines the required timing.

The interval between the start of two successive beacons is called the beacon period. The beacon frame transmission is aligned with the beginning of the BEACON_RESERVED interval. Each beacon is preceded by a guard time interval where no ping slot can be placed. The length of the guard interval corresponds to the time on air of the longest allowed frame. This is to insure that a Class B downlink initiated during a ping slot just before the guard time will always have time to complete without colliding with the beacon transmission. The usable time interval for ping slot therefore spans from the end of the beacon reserved time interval to the beginning of the next beacon guard interval.

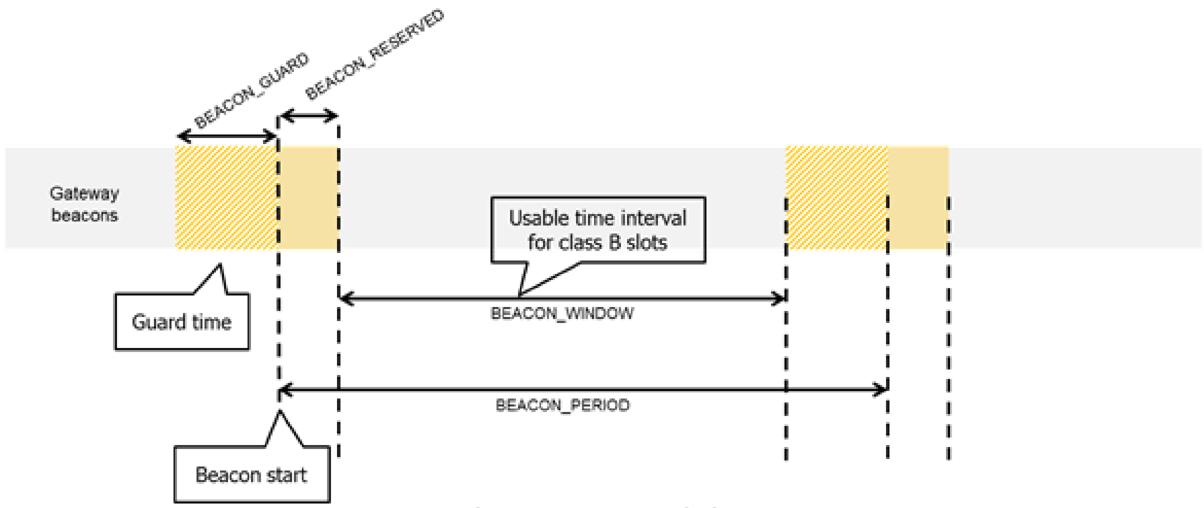


Figure 14: Beacon timing

Beacon_period	128 s
Beacon_reserved	2.120 s
Beacon_guard	3.000 s
Beacon-window	122.880 s

Table 13: Beacon timing

The beacon frame time on air is actually much shorter than the beacon reserved time interval to allow appending network management broadcast frames in the future.

The beacon window interval is divided into 2^{12} = 4096 ping slots of 30 ms each numbered from 0 to 4095.

An end-device using the slot number N MUST have its receiver on *Ton* seconds after the start of the beacon where:

$$Ton = beacon\_reserved + N * 30 \text{ ms}$$

N is called the *slot index.*

The latest ping slot starts at *beacon_reserved* + 4095 * 30 ms = 124 970 ms after the beacon start or 3030 ms before the beginning of the next beacon.

13.2 Slot randomization

To avoid systematic collisions or over-hearing problems the slot index is randomized and changed at every beacon period.

The following parameters are used:

DevAddr	Device 32 bit network unicast or multicast address
pingNb	Number of ping slots per beacon period. This MUST be a power of 2 integer: *pingNb* = 2^k where 0 <= k <=7
pingPeriod	Period of the device receiver wake-up expressed in number of slots: *pingPeriod* = 2^{12} / *pingNb*
pingOffset	Randomized offset computed at each beacon period start. Values can range from 0 to (pingPeriod-1)
beaconTime	The time carried in the field **BCNPayload**.Time of the immediately preceding beacon frame
slotLen	Length of a unit ping slot = 30 ms

Table 14 : Class B slot randomization algorithm parameters

At each beacon period the end-device and the server compute a new pseudo-random offset to align the reception slots. An AES encryption with a fixed key of all zeros is used to randomize:

Key = 16 x 0x00
Rand = aes128_encrypt(Key, beaconTime | DevAddr | pad16)
pingOffset = (*Rand*[0] + *Rand*[1]x 256) modulo *pingPeriod*

The slots used for this beacon period will be:

pingOffset + *N* x *pingPeriod* with *N*=[0:*pingNb*-1]

The node therefore opens receive slots starting at :

First slot	Beacon_reserved + pingOffset x slotLen
Slot 2	Beacon_reserved + (pingOffset + pingPeriod) x slotLen
Slot 3	Beacon_reserved + (pingOffset + 2 x pingPeriod) x slotLen
…	…

If the end-device serves simultaneously a unicast and one or more multicast slots this computation is performed multiple times at the beginning of a new beacon period. Once for the unicast address (the node network address) and once for each multicast group address.

In the case where a multicast ping slot and a unicast ping slot collide and cannot be served by the end-device receiver then the end-device should preferentially listen to the multicast slot. If there is a collision between multicast reception slots the FPending bit of the previous multicast frame can be used to set a preference.

The randomization scheme prevents a systematic collision between unicast and multicast slots. If collisions happen during a beacon period then it is unlikely to occur again during the next beacon period.

14 Class B MAC commands

All commands described in the Class A specification SHALL be implemented in Class B devices. The Class B specification adds the following MAC commands.

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x10	***PingSlotInfoReq***	x		Used by the end-device to communicate the unicast ping slot periodicity to the Network Server
0x10	***PingSlotInfoAns***		x	Used by the network to acknowledge a PingInfoSlotReq command
0x11	***PingSlotChannelReq***		x	Used by the Network Server to set the unicast ping channel frequency and data rate of an end-device
0x11	***PingSlotChannelAns***	x		Used by the end-device to acknowledge a ***PingSlotChannelReq***command
0x12	***BeaconTimingReq***	x		Deprecated
0x12	***BeaconTimingAns***		x	deprecated
0x13	***BeaconFreqReq***		x	Command used by the Network Server to modify the frequency at which the end-device expects to receive beacon broadcast
0x13	***BeaconFreqAns***	x		Used by the end-device to acknowledge a BeaconFreqReq command

Table 15 : Class B MAC command table

14.1 PingSlotInfoReq

With the ***PingSlotInfoReq*** command an end-device informs the server of its unicast ping slot periodicity. This command may only be used to inform the server of the periodicity of a UNICAST ping slot. A multicast slot is entirely defined by the application and should not use this command.

Size (bytes)	1
PingSlotInfoReq Payload	PingSlotParam

Figure 15 : PingSlotInfoReq payload format

Bit#	7:3	[2:0]
PingSlotParam	RFU	Periodicity

The **Periodicity** subfield is an unsigned 3 bits integer encoding the ping slot period currently used by the end-device using the following equation.

$$pingNb = 2^{7-Periodicity} \; and \; pingPeriod = 2^{5+Periodicity}$$

The actual ping slot periodicity will be equal to $0.96 \times 2^{Periodicity}$ in seconds

- **Periodicity** = 0 means that the end-device opens a ping slot approximately every second during the beacon_window interval
- **Periodicity** = 7 , every 128 seconds which is the maximum ping period supported by the LoRaWAN Class B specification.

To change its ping slot periodicity a device SHALL first revert to Class A , send the new periodicity through a ***PingSlotInfoReq*** command and get an acknowledge from the server through a ***PingSlotInfoAns*** . It MAY then switch back to Class B with the new periodicity.

This command MAY be concatenated with any other MAC command in the **FHDRFOpt** field as described in the Class A specification frame format.

14.2 BeaconFreqReq

This command is sent by the server to the end-device to modify the frequency on which this end-device expects the beacon.

Octets	3
BeaconFreqReq payload	Frequency

Figure 16 : BeaconFreqReq payload format

The Frequency coding is identical to the ***NewChannelReq*** MAC command defined in the Class A.

Frequency is a 24bits unsigned integer. The actual beacon channel frequency in Hz is 100 x frequ. This allows defining the beacon channel anywhere between 100 MHz to 1.67 GHz by 100 Hz step. The end-device has to check that the frequency is actually allowed by its radio hardware and return an error otherwise.

A valid non-zero Frequency will force the device to listen to the beacon on a fixed frequency channel even if the default behavior specifies a frequency hopping beacon (i.e US ISM band).

A value of 0 instructs the end-device to use the default beacon frequency plan as defined in the “Beacon physical layer” section. Where applicable the device resumes frequency hopping beacon search.

Upon reception of this command the end-device answers with a ***BeaconFreqAns*** message. The MAC payload of this message contains the following information:

Size (bytes)	1
BeaconFreqAns payload	Status

Figure 17 : BeaconFreqAns payload format

The **Status** bits have the following meaning:

Bits	7:1	0
Status	RFU	Beacon frequency ok

	Bit = 0	Bit = 1
Beacon frequency ok	The device cannot use this frequency, the previous beacon frequency is kept	The beacon frequency has been changed

14.3 PingSlotChannelReq

This command is sent by the server to the end-device to modify the frequency and/or the data rate on which the end-device expects the downlink pings.

This command **can only be sent in a class A receive window** (following an uplink). The command SHALL NOT be sent in a class B ping-slot. If the device receives it inside a class B ping-slot, the MAC command SHALL NOT be processed. Once the NS has sent the first PingSlotChannelReq command, it SHOULD resend it until it receives a PingSlotChannelAns from the device. It MUST NOT attempt to use a class B ping slot until it receives the PingSlotChannelAns.

Octets	3	1
PingSlotChannelReq Payload	Frequency	DR

Figure 18 : PingSlotChannelReq payload format

The Frequency coding is identical to the ***NewChannelReq*** MAC command defined in the Class A.

Frequency is a 24bits unsigned integer. The actual ping channel frequency in Hz is 100 x frequ. This allows defining the ping channel anywhere between 100MHz to 1.67GHz by 100Hz step. The end-device has to check that the frequency is actually allowed by its radio hardware and return an error otherwise.

A value of 0 instructs the end-device to use the default frequency plan. Where applicable the device resumes frequency hopping beacon search.

The DR byte contains the following fields:

Bits	7:4	3:0
DR	RFU	data rate

The “data rate” subfield is the index of the Data Rate used for the ping-slot downlinks. The relationship between the index and the physical data rate is defined in [PHY] for each region.

Upon reception of this command the end-device answers with a ***PingSlotFreqAns*** message. The MAC payload of this message contains the following information:

Size (bytes)	1
pingSlotFreqAns Payload	Status

Figure 19 : PingSlotFreqAns payload format

The **Status** bits have the following meaning:

Bits	7:2	1	0
Status	RFU	Data rate ok	Channel frequency ok

	Bit = 0	Bit = 1
Data rate ok	The designated data rate is not defined for this end device, the previous data rate is kept	The data rate is compatible with the possibilities of the end device
Channel frequency ok	The device cannot receive	This frequency can be used

	on this frequency	by the end-device

If either of those 2 bits equals 0, the command did not succeed and the ping-slot parameters have not been modified.

14.4 BeaconTimingReq & BeaconTimingAns

These MAC commands are deprecated in the LoRaWAN1.0.3 version. The device may use DeviceTimeReq&Ans commands as a substitute.

15 Beaconing (Class B option)

15.1 Beacon physical layer

Besides relaying messages between end-devices and Network Servers, gateways MAY participate in providing a time-synchronization mechanisms by sending beacons at regular fixed intervals. All beacons are transmitted in radio packet implicit mode, that is, without a LoRa physical header and with no CRC being appended by the radio.

PHY	Preamble	BCNPayload

Figure 20 : beacon physical format

The beacon Preamble SHALL begin with (a longer than default) 10 unmodulated symbols. This allows end-devices to implement a low power duty-cycled beacon search.

The beacon frame length is tightly coupled to the operation of the radio Physical layer. Therefore the actual frame length and content might change from one region implementation to another. The beacon content, modulation parameters and frequencies to use are specified in [PHY] for each region.

15.2 Beacon frame content

The beacon payload **BCNPayload** consists of a network common part and an OPTIONAL gateway-specific part.

Size (bytes)	2 to 5	4	2	7	0 to 3	2
BCNPayload	RFU	Time	CRC	GwSpecific	RFU	CRC
	Compulsory common part			Optional part		

Figure 21 : beacon frame content

The common part contains an RFU field equal to 0, a timestamp **Time** in seconds since January 6, 1980 00:00:00 UTC (start of the GPS epoch) modulo 2^32. The integrity of the beacon's network common part is protected by a 16 bits CRC. The CRC-16 is computed on the RFU+Time fields as defined in the IEEE 802.15.4-2003 section 7.2.1.8. This CRC uses the following polynomial $\mathbf{P(x) = x^{16} + x^{12} + x^{5} + x^{0}}$. The CRC is calculated on the bytes in the order they are sent over-the-air

For example: This is a valid EU868 beacon frame:

00 00 | 00 00 02 CC | A2 7E | 00 | 01 20 00 | 00 81 03 | DE 55

Bytes are transmitted left to right. The first CRC is calculated on [00 00 00 00 02 CC]. The corresponding field values are:

Field	RFU	Time	**CRC**	InfoDesc	lat	long	**CRC**
Value Hex	0000	CC020000	**7EA2**	0	002001	038100	**55DE**

Figure 22 : example of beacon CRC calculation (1)

The optional gateway specific part provides additional information regarding the gateway sending a beacon and therefore may differ for each gateway. The RFU field when applicable (region specific) should be equal to 0. The optional part is protected by a CRC-16 computed on the GwSpecific+RFU fields. The CRC-16 definition is the same as for the mandatory part.

For example: This is a valid US900 beacon:

Field	RFU	Time	**CRC**	InfoDesc	lat	long	**RFU**	**CRC**
Value Hex	000000	CC020000	**7E A2**	00	002001	038100	**00**	**D450**

Figure 23 : example of beacon CRC calculation (2)

Over the air the bytes are sent in the following order:

00 00 00 | 00 00 02 CC | A2 7E | 00 | 01 20 00 | 00 81 03 |00 | 50 D4

Listening and synchronizing to the network common part is sufficient to operate a stationary end-device in Class B mode. A mobile end-device MAY also demodulate the gateway specific part of the beacon to be able to signal to the Network Server whenever he is moving from one cell to another.

> **Note:** As mentioned before, all gateways participating in the beaconing process send their beacon simultaneously so that for network common part there are no visible on-air collisions for a listening end-device even if the end-device simultaneously receives beacons from several gateways. Not all gateways are required to participate in the beaconing process. The participation of a gateway to a given beacon may be randomized. With respect to the gateway specific part, collision occurs but an end-device within the proximity of more than one gateway will still be able to decode the strongest beacon with high probability.

15.3 Beacon *GwSpecific* field format

The content of the **GwSpecific** field is as follow:

Size (bytes)	1	6
GwSpecific	InfoDesc	Info

Figure 24 : beacon GwSpecific field format

The information descriptor **InfoDesc** describes how the information field **Info** shall be interpreted.

InfoDesc	**Meaning**
0	GPS coordinate of the gateway first antenna
1	GPS coordinate of the gateway second antenna
2	GPS coordinate of the gateway third antenna
3	NetID+GatewayID
4:127	RFU
128:255	Reserved for custom network specific broadcasts

Table 16 : beacon infoDesc index mapping

For a single omnidirectional antenna gateway the **InfoDesc** value is 0 when broadcasting GPS coordinates. For a site featuring 3 sectored antennas for example, the first antenna broadcasts the beacon with **InfoDesc** equals 0, the second antenna with **InfoDesc** field equals 1, etc.

15.3.1 Gateway GPS coordinate:InfoDesc = 0, 1 or 2

For **InfoDesc** = 0 ,1 or 2, the content of the **Info** field encodes the GPS coordinates of the antenna broadcasting the beacon

Size (bytes)	3	3
Info	Lat	Lng

Figure 25 : beacon Info field format , infoDesc = 0,1,2

The latitude and longitude fields (**Lat** and **Lng**, respectively) encode the geographical location of the gateway as follows:

- The north-south latitude is encoded using a two's complement 24 bit word where -2^{23} corresponds to 90° south (the South Pole) and 2^{23}-1 corresponds to ~90° north (the North Pole). The Equator corresponds to 0.
- The east-west longitude is encoded using a two's complement 24 bit word where -2^{23} corresponds to 180° West and 2^{23}-1 corresponds to ~180° East. The Greenwich meridian corresponds to 0.

15.3.2 NetID+GatewayID

For **InfoDesc** = 3, the content of the **Info** field encodes the network's NetID plus a freely allocated gateway or cell identifier. The format of the Info field is:

Size (bytes)	3	3
Info	NetID	GatewayID

Figure 26 : beacon Info field format, infoDesc=3

15.4 Beaconing precise timing

The beacon is sent every 128 seconds starting at January 6, 1980 00:00:00 UTC (start of the GPS epoch) plus TBeaconDelay. Therefore the beacon is sent at

$B_T = k * 128 + \text{TBeaconDelay}$

seconds after the GPS epoch.

wherebyk is the smallest integer for which

$k * 128 > T$

whereby

T = seconds since January 6, 1980 00:00:00 UTC (start of the GPS time).

> **Note:** T is GPS time and unlike Unix time, *T* is strictly monotonically increasing and is not influenced by leap seconds.

Whereby TBeaconDelay is 1.5 mSec +/- 1uSec delay.
TBeaconDelay is meant to allow a slight transmission delay of the gateways required by the radio system to switch from receive to transmit mode.

All end-devices ping slots use the beacon transmission start time as a timing reference, therefore the Network Server as to take TBeaconDelay into account when scheduling the class B downlinks.

15.5 Network downlink route update requirements

When the network attempts to communicate with an end-device using a Class B downlink slot, it transmits the downlink from the gateway which was closest to the end-device when the last uplink was received. Therefore the Network Server needs to keep track of the rough position of every Class B device.

Whenever a Class B device moves and changes cell, it needs to communicate with the Network Server in order to update its downlink route. This update can be performed simply by sending a “confirmed” or “unconfirmed” uplink, possibly without applicative payload.

The end-device has the choice between 2 basic strategies:

- Systematic periodic uplink: simplest method that doesn’t require demodulation of the “gateway specific” field of the beacon. Only applicable to slowly moving or stationery end-devices. There are no requirements on those periodic uplinks.
- Uplink on cell change: The end-device demodulates the optional “gateway specific” field of the beacon, detects that the ID of the gateway broadcasting the beacon it demodulates has changed, and sends an uplink. In that case the device SHALL respect a pseudo random delay in the [0:120] seconds range between the beacon demodulation and the uplink transmission. This is required to insure that the uplinks of multiple Class B devices entering or leaving a cell during the same beacon period will not systematically occur at the same time immediately after the beacon broadcast.

Failure to report cell change will result in Class B downlink being temporary not operational. The Network Server may have to wait for the next end-device uplink to transmit downlink traffic.

16 Class B unicast & multicast downlink channel frequencies

The class B downlink channel selection mechanism depends on the way the class B beacon is being broadcasted.

16.1 Single channel beacon transmission

In certain regions (ex EU868) the beacon is transmitted on a single channel. In that case, all unicast&multicastClass B downlinks use a single frequency channel defined by the "***PingSlotChannelReq"*** MAC command. The default frequency is defined in [PHY].

16.2 Frequency-hopping beacon transmission

In certain regions (ex US902-928 or CN470-510) the class B beacon is transmitted following a frequency hopping pattern.

In that case, by default Class B downlinks use a channel which is a function of the Time field of the last beacon (see Beacon Frame content) and the DevAddr.

$$\text{Class B downlink channel} = \left[\text{DevAddr} + \text{floor}\left(\frac{\text{Beacon\_Time}}{\text{Beacon\_period}}\right)\right] \text{ modulo NbChannel}$$

- Whereby Beacon_Time is the 32 bit Time field of the current beacon period
- Beacon_period is the length of the beacon period (defined as 128sec in the specification)
- Floor designates rounding to the immediately lower integer value
- DevAddr is the 32 bits network address of the device
- NbChannel is the number of channel over which the beacon is frequency hopping

Class B downlinks therefore hop across NbChannel channels (identical to the beacon transmission channels) in the ISM band and all Class B end-devices are equally spread amongst the NbChannel downlink channels.

If the "***PingSlotChannelReq"*** command with a valid non-zero argument is used to set the Class B downlink frequency then all subsequent ping slots should be opened on this single frequency independently of the last beacon frequency.

If the "***PingSlotChannelReq"*** command with a zero argument is sent, the end-device should resume the default frequency plan, id Class B ping slots hoping across 8 channels.

The underlying idea is to allow network operators to configure end-devices to use a single proprietary dedicated frequency band for the Class B downlinks if available, and to keep as much frequency diversity as possible when the ISM band is used.

CLASS C – CONTINUOUSLY LISTENING

17 Class C: Continuously listening end-device

The end-devices implanting the Class C option are used for applications that have sufficient power available and thus do not need to minimize reception time.

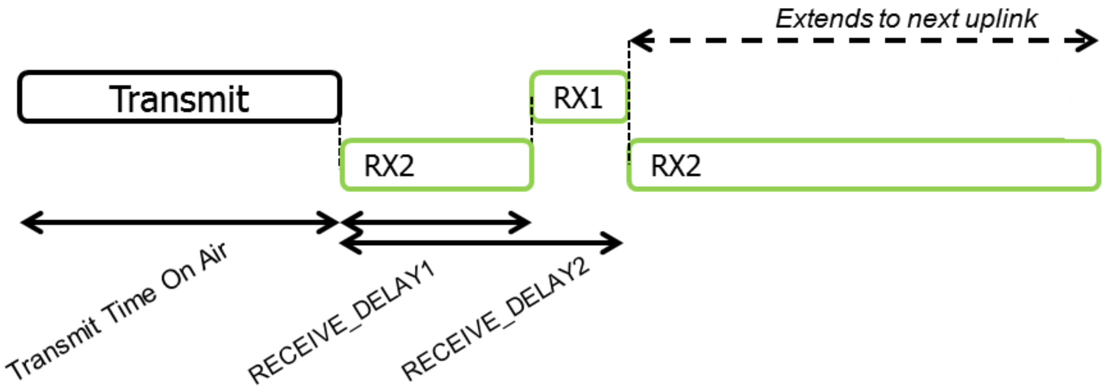
Class C end-devices cannot implement Class B option.

The Class C end-device will listen with RX2 windows parameters as often as possible. The end-device listens on RX2 when it is not either (a) sending or (b) receiving on RX1, according to Class A definition. To do so, it will open a short window on RX2 parameters between the end of the uplink transmission and the beginning of the RX1 reception window and it will switch to RX2 reception parameters as soon as the RX1 reception window is closed; the RX2 reception window will remain open until the end-device has to send another message.

> **Note:** There is not specific message for a node to tell the server that it is a Class C node. It is up to the application on server side to know that it manages Class C nodes based on the contract passed during the join procedure.

17.1 Second receive window duration for Class C

Class C devices implement the same two receive windows as Class A devices, but they do not close RX2 window until they need to send again. Therefore they may receive a downlink in the RX2 window at nearly any time, including downlinks sent for the purpose of MAC command or ACK transmission. A short listening window on RX2 frequency and data rate is also opened between the end of the transmission and the beginning of the RX1 receive window.



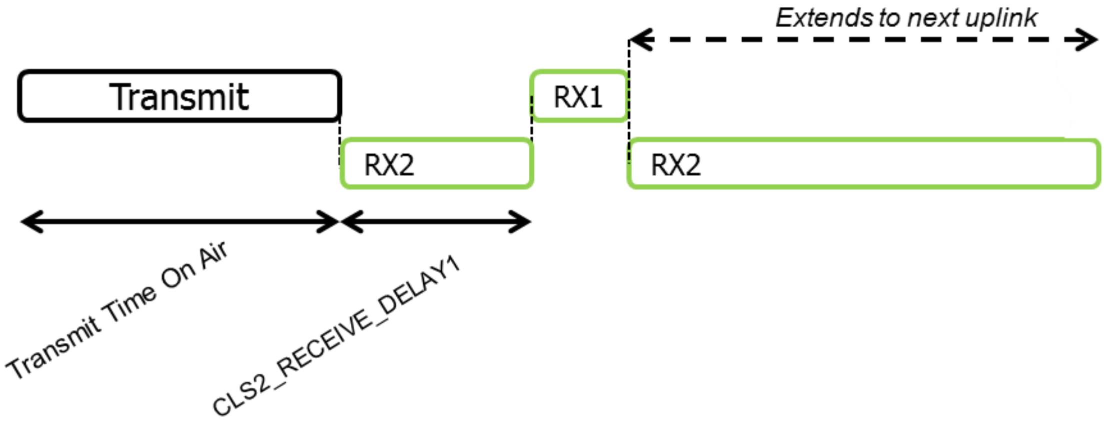


Figure 27: Class C end-device reception slot timing.

17.2 Class C Multicast downlinks

Similarly to Class B, Class C devices may receive multicast downlink frames. The multicast address and associated network session key and application session key must come from the application layer.

> Example: [RPD1] provides an application layer mechanism to setup a multicast group over-the-air.

The same limitations apply for Class C multicast downlink frames:

- They are not allowed to carry MAC commands, neither in the **FOpt** field, nor in the payload on port 0 because a multicast downlink does not have the same authentication robustness as a unicast frame.
- The **ACK** and **ADRACKReq** bits MUST be zero. The **MType** field MUST carry the value for Unconfirmed Data Down.
- The **FPending** bit indicates there is more multicast data to be sent. Given that a Class C device keeps its receiver active most of the time, the **FPending** bit does not trigger any specific behavior of the end-device.

SUPPORT INFORMATION

This sub-section is only a recommendation.

18 Examples and Application Information

Examples are illustrations of the LoRaWAN spec for information, but they are not part of the formal specification.

18.1 Uplink Timing Diagram for Confirmed Data Messages

The following diagram illustrates the steps followed by an end-device trying to transmit two confirmed data frames (Data0 and Data1):

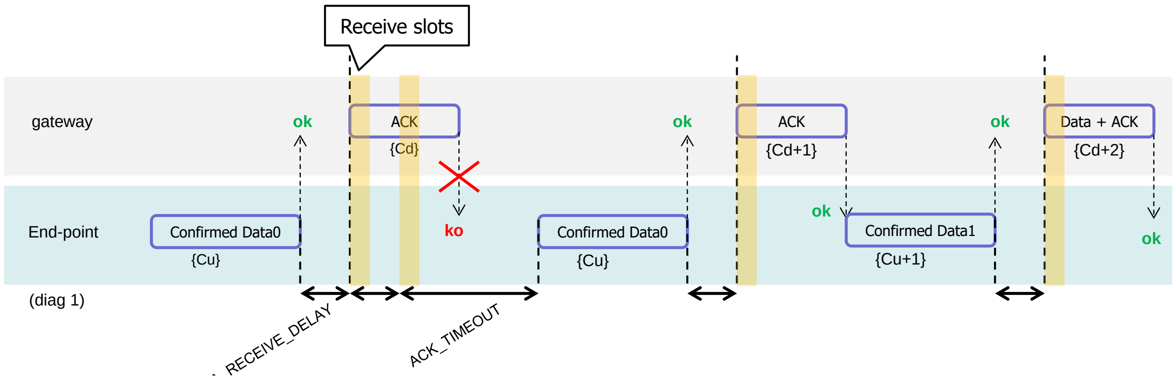


Figure 28: Uplink timing diagram for confirmed data messages

The end-device first transmits a confirmed data frame containing the Data0 payload at an arbitrary instant and on an arbitrary channel. The frame counter Cu is simply derived by adding 1 to the previous uplink frame counter. The network receives the frame and generates a downlink frame with the ACK bit set exactly RECEIVE_DELAY1 seconds later, using the first receive window of the end-device. This downlink frame uses the same data rate and the same channel as the Data0 uplink. The downlink frame counter Cd is also derived by adding 1 to the last downlink towards that specific end-device. If there is no downlink payload pending the network SHALL generate a frame without a payload. In this example the frame carrying the ACK bit is not received.

If an end-device does not receive a frame with the ACK bit set in one of the two receive windows immediately following the uplink transmission it MAY resend the same frame with the same payload and frame counter again at least ACK_TIMEOUT seconds after the second reception window. This resend MUST be done on another channel and MUST obey the duty cycle limitation as any other normal transmission. If this time the end-device receives the ACK downlink during its first receive window, as soon as the ACK frame is demodulated, the end-device is free to transmit a new frame on a new channel.
The third ACK frame in this example also carries an application payload. A downlink frame can carry any combination of ACK, MAC control commands and payload.

18.2 Downlink Diagram for Confirmed Data Messages

The following diagram illustrates the basic sequence of a “confirmed” downlink.

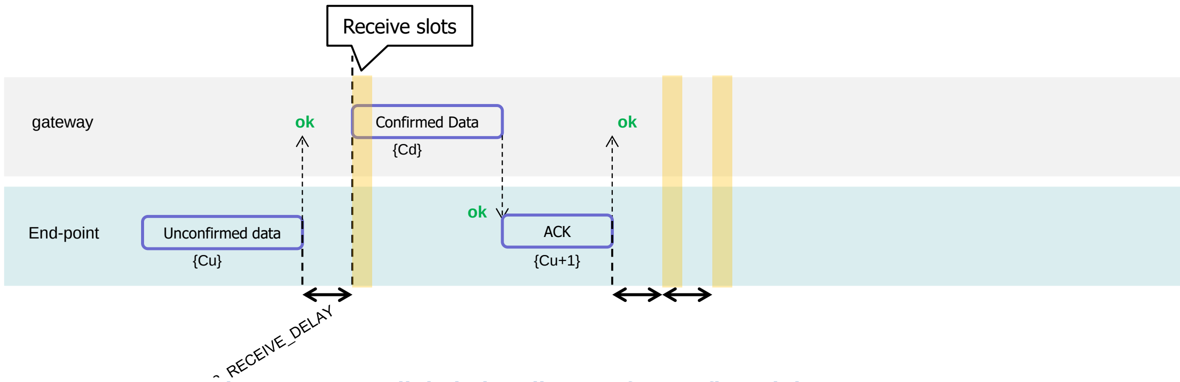


Figure 29: Downlink timing diagram for confirmed data messages

The frame exchange is initiated by the end-device transmitting an “unconfirmed” application payload or any other frame on channel A. The network uses the downlink receive window to transmit a “confirmed” data frame towards the end-device on the same channel A. Upon reception of this data frame requiring an acknowledgement, the end-device transmits a frame with the ACK bit set at its own discretion. This frame might also contain piggybacked data or MAC commands as its payload. This ACK uplink is treated like any standard uplink, and as such is transmitted on a random channel that might be different from channel A.

> **Note:** To allow the end-devices to be as simple as possible and have keep as few states as possible it may transmit an explicit (possibly empty) acknowledgement data message immediately after the reception of a data message requiring an acknowledgment. Alternatively the end-device may defer the transmission of an acknowledgement to piggyback it with its next data message.

18.3 Downlink Timing for Frame-Pending Messages

The next diagram illustrates the use of the **frame pending** (FPending) bit on a downlink. The FPending bit can only be set on a downlink frame and informs the end-device that the network has several frames pending for him; the bit is ignored for all uplink frames.

If a frame with the FPending bit set requires an acknowledgement, the end-device shall do so as described before. If no acknowledgment is required, the end-device MAY send an empty data message to open additional receive windows at its own discretion, or wait until it has some data to transmit itself and open receive windows as usual.

> **Note:** The FPending bit is independent to the acknowledgment scheme.

(*) F_P means ‘frame pending’ bit set

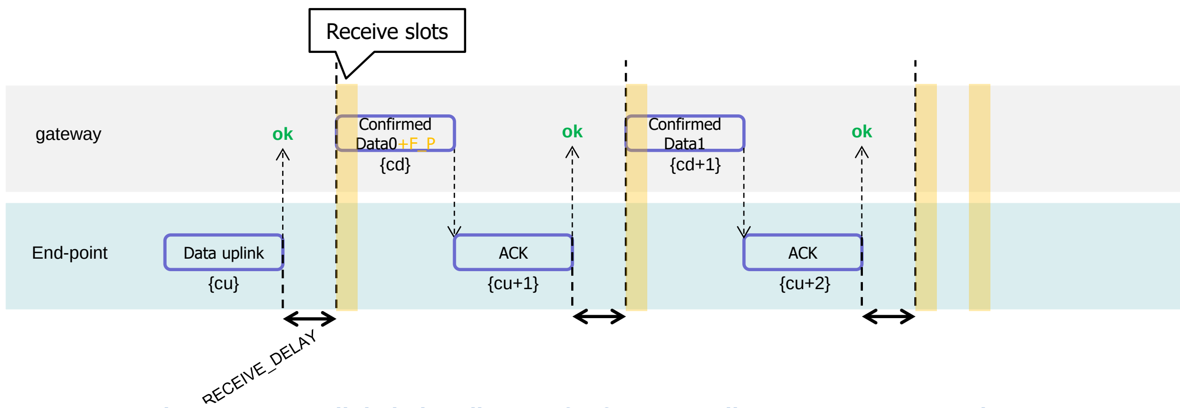


Figure 30: Downlink timing diagram for frame-pending messages, example 1

In this example the network has two confirmed data frames to transmit to the end-device. The frame exchange is initiated by the end-device via a normal “unconfirmed” uplink message on channel A. The network uses the first receive window to transmit the Data0 with the bit FPending set as a confirmed data message. The device acknowledges the reception of the frame by transmitting back an empty frame with the ACK bit set on a new channel B. RECEIVE_DELAY1 seconds later, the network transmits the second frame Data1 on channel B, again using a confirmed data message but with the FPending bit cleared. The end-device acknowledges on channel C.

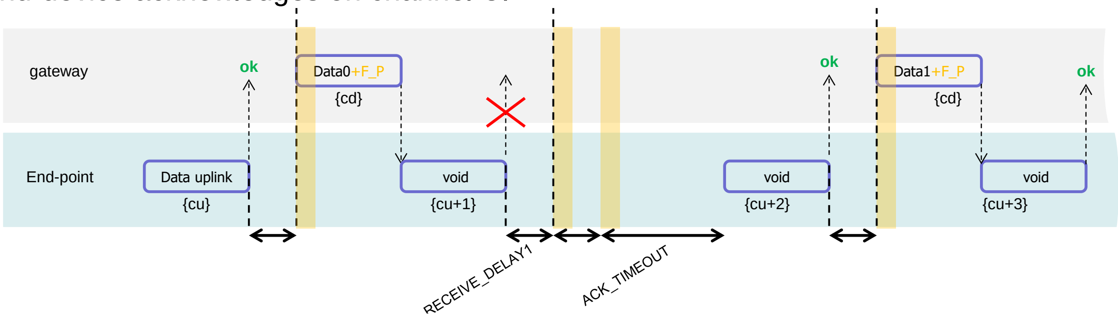


Figure 31: Downlink timing diagram for frame-pending messages, example 2

In this example, the downlink frames are “unconfirmed” frames, the end-device does not need to send back and acknowledge. Receiving the Data0 unconfirmed frame with the FPending bit set the end-device sends an empty data frame. This first uplink is not received by the network. If no downlink is received during the two receive windows, the network has to wait for the next spontaneous uplink of the end-device to retry the transfer. The end-device can speed up the procedure by sending a new empty data frame.

Note: An acknowledgement is never sent twice.

The FPending bit, the ACK bit, and payload data can all be present in the same downlink. For example, the following frame exchange is perfectly valid.

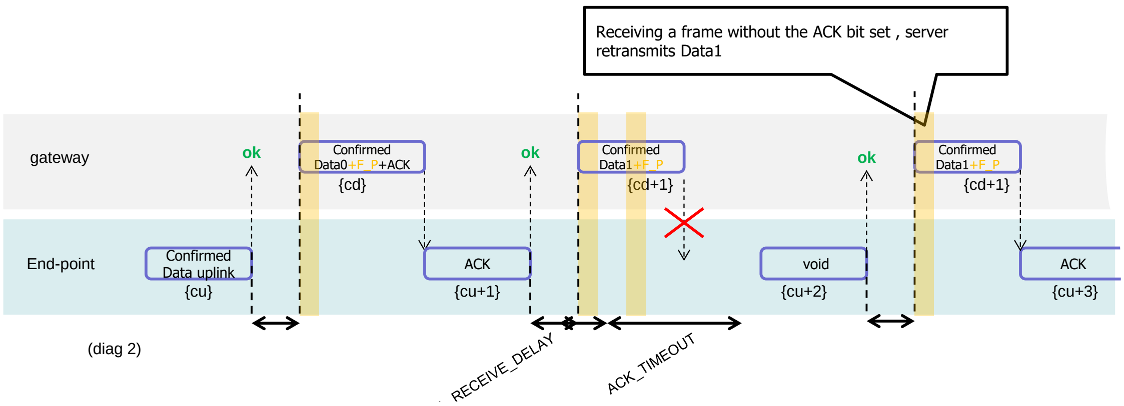


Figure 32: Downlink timing diagram for frame-pending messages, example 3

The end-device sends a “confirmed data” uplink. The network can answer with a confirmed downlink containing Data + ACK + “Frame pending” then the exchange continues as previously described.

18.4 Data-Rate Adaptation during Message Retransmissions

When an end-device attempts the transmission of a “confirmed’ frame toward the network it expects to receive an acknowledgement in one of the subsequent reception slot. In the absence of the acknowledgement it will try to re-transmit the same data again. This retransmission happens on a new frequency channel, but can also happen at a different data rate (preferable lower) than the previous one. It is strongly recommended to adopt the following re-transmission strategy.

The first transmission of the “confirmed” frame happens with a data rate DR.

Transmission nb	**Data Rate**
1 (first)	DR
2	DR
3	max(DR-1,0)
4	max(DR-1,0)
5	max(DR-2,0)
6	max(DR-2,0)
7	max(DR-3,0)
8	max(DR-3,0)

The Data Rate max(a,b) stands for maximum of a and b values.

If after a recommended 8 transmissions, the frame has not been acknowledged the MAC layer should return an error code to the application layer.

> **Note:** For each re-transmission, the frequency channel is selected randomly as for normal transmissions.

Any further transmission uses the last data rate used.

For example if an end-device sends a “confirmed” frame first using DR5 and has to retransmit 3 times (twice at DR5 then twice at DR4), the next frame transmitted will use DR4

Other example, if an end-device sends a “confirmed” frame first using DR5 and does not receive an acknowledge after 8 transmissions (2 at DR5, 2 at DR4, … , 2 at DR2), and the application of this end-device re-initiates a “confirmed” transmission a little later, the first two transmission will be tentatively at DR2, then switch to DR1, then to DR0.

19 Recommendation on contract to be provided to the network server by the end-device provider at the time of provisioning

Configuration data related to the end-device and its characteristics must be known by the network server at the time of provisioning. –This provisioned data is called the “contract”. This contract cannot be provided by the end-device and must be supplied by the end-device provider using another channel (out-of-band communication).

This end-device contract is stored in the network server. It can be used by the application server and the network controller to adapt the algorithms.

This data will include:

- End-device specific radio parameters (device frequency range, device maximal output power, device communication settings - RECEIVE_DELAY1, RECEIVE_DELAY2)
- Application type (Alarm, Metering, Asset Tracking, Supervision, Network Control)

20 Recommendation on finding the locally used channels

End-devices that can be activated in territories that are using different frequencies for LoRaWAN will have to identify what frequencies are supported for join message at their current location before they send any message. The following methods are proposed:

- A GPS enabled end-device can use its GPS location to identify which frequency band to use.
- End-device can search for a beacon and use its frequency to identify its region
- End-device can search for a beacon and if this one is sending the antenna GPS coordinate, it can use this to identify its region
- End-device can search for a beacon and if this one is sending a list of join frequencies, it can use this to send its join message

21 Revisions

21.1 Revision 1.0

- Approved version of LoRaWAN1.0

21.2 Revision 1.0.1

- Clarified the RX window start time definition
- Corrected the maximum payload size for DR2 in the NA section
- Corrected the typo on the downlink data rate range in 7.2.2
- Introduced a requirement for using coding rate 4/5 in 7.2.2 to guarantee a maximum time on air < 400mSec
- Corrected the JoinAccept MIC calculation in 6.2.5
- Clarified the NbRep field and renamed it to NbTrans in 5.2
- Removed the possibility to not encrypt the Applicative payload in the MAC layer , removed the paragraph 4.3.3.2. If further security is required by the application , the payload will be encrypted, using any method, at the application layer then re-encrypted at the MAC layer using the specified default LoRaWAN encryption
- Corrected FHDR field size typo
- Corrected the channels impacted by ChMask when chMaskCntl equals 6 or 7 in 7.2.5
- Clarified 6.2.5 sentence describing the RX1 slot data rate offset in the JoinResp message
- Removed the second half of the DRoffset table in 7.2.7 , as DR>4 will never be used for uplinks by definition
- Removed explicit duty cycle limitation implementation in the EU868Mhz ISM band (chapter7.1)
- Made the RXtimingSetupAns and RXParamSetupAns sticky MAC commands to avoid end-device's hidden state problem. (in 5.4 and 5.7)
- Added a frequency plan for the Chinese 470-510MHz metering band
- Added a frequency plan for the Australian 915-928MHz ISM band

21.3 Revision 1.0.2

- Extracted section 7 "Physical layer" that will now be a separated document "LoRaWAN regional physical layers definition"
- corrected the ADR_backoff sequence description (ADR_ACK_LIMT was written instead of ADR_ACK_DELAY) paragraph 4.3.1.1
- Corrected a formatting issue in the title of section 18.2 (previously section 19.2 in the 1.0.1 version)
- Added the DlChannelRec MAC command, this command is used to modify the frequency at which an end-device expects a downlink.
- Added the Tx ParamSetupRec MAC command. This command enables to remotely modify the maximum TX dwell time and the maximum radio transmit power of a device in certain regions
- Added the ability for the end-device to process several ADRreq commands in a single block in 5.2
- Clarified AppKey definition

21.4 Revision 1.0.3

- Imported the classB chapter from the LoRaWAN1.1 specification
- Added the DeviceTimeReq/Ans MAC command in the classA chapter, those commands are required for the classB beacon acquisition, the MAC commands BeaconTimingReq/Ans are deprecated.
- Corrected incorrect GPS epoch references
- Corrected various typos

22 Glossary

ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
AFA	Adaptive Frequency Agility
AR	Acknowledgement Request
CBC	Cipher Block Chaining
CMAC	Cipher-based Message Authentication Code
CR	Coding Rate
CRC	Cyclic Redundancy Check
DR	Data Rate
ECB	Electronic Code Book
ETSI	European Telecommunications Standards Institute
EIRP	Equivalent Isotropically Radiated Power
FSK	Frequency Shift Keying modulation technique
GPRS	General Packet Radio Service
HAL	Hardware Abstraction Layer
IP	Internet Protocol
LBT	Listen Before Talk
LoRa™	Long Range modulation technique
LoRaWAN™	Long Range Network protocol
MAC	Medium Access Control
MIC	Message Integrity Code
RF	Radio Frequency
RFU	Reserved for Future Usage
Rx	Receiver
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal Noise Ratio
SPI	Serial Peripheral Interface
SSL	Secure Socket Layer
Tx	Transmitter
USB	Universal Serial Bus

23 Bibliography

23.1 References

[IEEE802154]: IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Std 802.15.4TM-2011 (Revision of IEEE Std 802.15.4-2006), September 2011.

[RFC4493]: The AES-CMAC Algorithm, June 2006.

[RPD1] : LoRa Alliance 2018, RPD_Remote_Multicast_Setup_v1.0.0

24 NOTICE OF USE AND DISCLOSURE

LoRa Alliance, Inc.
3855 SW 153rd Drive
Beaverton, OR 97007