International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Y.4480
(11/2021)

SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

Internet of things and smart cities and communities – Frameworks, architectures and protocols

Low power protocol for wide area wireless networks

Recommendation ITU-T Y.4480

ITU-T

ITU-T Y-SERIES RECOMMENDATIONS

GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

GLOBAL INFORMATION INFRASTRUCTURE	
General	Y.100–Y.199
Services, applications and middleware	Y.200–Y.299
Network aspects	Y.300–Y.399
Interfaces and protocols	Y.400–Y.499
Numbering, addressing and naming	Y.500–Y.599
Operation, administration and maintenance	Y.600–Y.699
Security	Y.700–Y.799
Performances	Y.800–Y.899
INTERNET PROTOCOL ASPECTS	
General	Y.1000–Y.1099
Services and applications	Y.1100–Y.1199
Architecture, access, network capabilities and resource management	Y.1200–Y.1299
Transport	Y.1300–Y.1399
Interworking	Y.1400–Y.1499
Quality of service and network performance	Y.1500–Y.1599
Signalling	Y.1600–Y.1699
Operation, administration and maintenance	Y.1700–Y.1799
Charging	Y.1800–Y.1899
IPTV over NGN	Y.1900–Y.1999
NEXT GENERATION NETWORKS	
Frameworks and functional architecture models	Y.2000–Y.2099
Quality of Service and performance	Y.2100–Y.2199
Service aspects: Service capabilities and service architecture	Y.2200–Y.2249
Service aspects: Interoperability of services and networks in NGN	Y.2250–Y.2299
Enhancements to NGN	Y.2300–Y.2399
Network management	Y.2400–Y.2499
Computing power networks	Y.2500–Y.2599
Packet-based Networks	Y.2600–Y.2699
Security	Y.2700–Y.2799
Generalized mobility	Y.2800–Y.2899
Carrier grade open environment	Y.2900–Y.2999
FUTURE NETWORKS	Y.3000–Y.3499
CLOUD COMPUTING	Y.3500–Y.3599
BIG DATA	Y.3600–Y.3799
QUANTUM KEY DISTRIBUTION NETWORKS	Y.3800–Y.3999
INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES	
General	Y.4000–Y.4049
Definitions and terminologies	Y.4050–Y.4099
Requirements and use cases	Y.4100–Y.4249
Infrastructure, connectivity and networks	Y.4250–Y.4399
Frameworks, architectures and protocols	**Y.4400–Y.4549**
Services, applications, computation and data processing	Y.4550–Y.4699
Management, control and performance	Y.4700–Y.4799
Identification and security	Y.4800–Y.4899
Evaluation and assessment	Y.4900–Y.4999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Y.4480

Low power protocol for wide area wireless networks

Summary

Recommendation Y.4480 describes a low power protocol for wide area wireless networks, which is optimized for battery-powered end-devices that may either be mobile or mounted at a fixed location.

NOTE – This protocol is technically equivalent to (and compatible with) the LoRaWAN® Link layer specification protocol [b-LoRaWAN TS001-1.0.4].

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Y.4480	2021-11-29	20	11.1002/1000/14818

Keywords

LoRaWAN, long range, low power protocol, wide area wireless networks.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Table of Contents

Page

Recommendation ITU-T Y.4480

Low power protocol for wide area wireless networks

1 Scope

This Recommendation specifies a low power, wide area networking protocol designed to wirelessly connect battery operated 'things' to the internet in regional, national or global networks, and targets key internet of things requirements such as bidirectional communication, end-to-end security, mobility and localization services.

NOTE – This Recommendation is technically equivalent to [b-LoRaWAN TS001-1.0.4].

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[IEEE 802.15.4]	IEEE Std 802.15.4 (2011), *IEEE Standard for Local and metropolitan area networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*.
[IETF RFC 4493]	IETF RFC 4493 (2006), *The AES-CMAC Algorithm*.
[NIST FIPS 197]	NIST FIPS 197 (2001), *Advanced Encryption Standard (AES)*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following term defined elsewhere:

3.1.1 nNetwork [b-ITU-T G.993.5]: A collection of interconnected elements that provide connection services to users.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 application server: Network element that exchanges encrypted application payloads with an end-device.

3.2.2 class: Mode of operation of an end-device.

3.2.3 join procedure: Activation procedure of an end-device over a network.

3.2.4 join server: Network element that provides session keys to an application server and to a network server.

3.2.5 network server: Network element that controls the operation of end-devices and gateways.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

ABP	Activation By Personalization
ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
CCM	Counter with Cipher block chaining Message
CMAC	Cipher-based Message Authentication Code
CRC	Cyclic Redundancy Check
DR	Data Rate
ECB	Electronic Code Book
EIRP	Effective Isotopic Radiated Power
EUI	Extended Unique Identifier
FUOTA	Firmware Updates Over The Air
IP	Internet Protocol
MAC	Medium Access Control
MIC	Message Integrity Code
OTAA	Over-The-Air Activation
RF	Radio Frequency
RFU	Reserved for Future Usage
RX	Reception
SNR	Signal-to-Noise Ratio
TX	Transmission

5 Conventions

In the body of this Recommendation, the words "SHALL", "SHALL NOT", "SHOULD" or "IS RECOMMENDED", "MAY" and "OPTIONAL" sometimes appear in all capitals, in which case they are to be interpreted, respectively, as "is required to", "is prohibited from", "is recommended", "can optionally" and "optional". The appearance of such phrases or keywords in an appendix or a material explicitly marked as informative is to be interpreted as having no normative intent.

Medium access control (MAC) commands are written in bold italics like ***LinkCheckReq***. Bits and bit fields are written with this font: `FRMPayload`. Constants are written all in capitals like RECEIVE_DELAY1, and variables are written in italics like *N*.

The octet order for all multi-octet fields SHALL be little endian.

Extended unique identifiers (EUI) are 8-octet fields and SHALL be transmitted as little endian.

By default, reserved for future usage (RFU) bits SHALL be set to 0 by the transmitter of the frame and SHALL be silently ignored by the receiver.

6 Introduction

Networks as defined in this Recommendation are typically laid out in a star-of-stars topology in which gateways (also known as concentrators, routers, access points or base stations) relay transmissions between end-devices and a central network server at the backend (see Figure 1). Gateways are connected to a network server via standard internet protocol (IP) connections, whereas end-devices use single-hop radio frequency (RF) communication to one or many gateways. All communication is generally bidirectional, although uplink communication from an end-device to a network server is expected to be the predominant traffic.

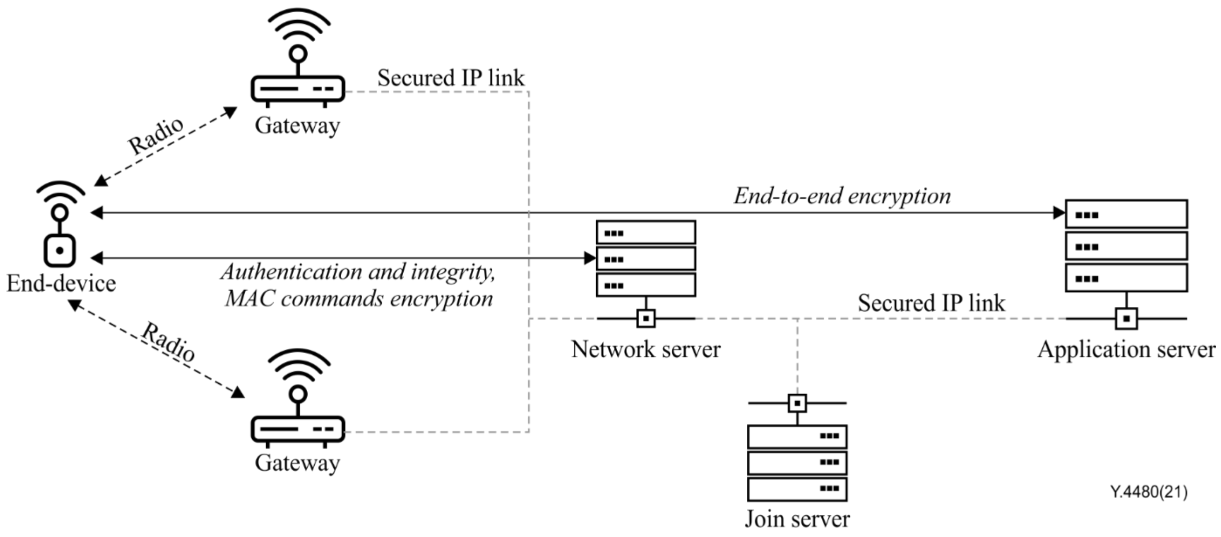

Figure 1 – Network architecture

Communication between end-devices and gateways is distributed over different frequency channels and data rates. Selecting the data rate (DR) is a trade-off between communication range and transmission (TX) duration. To maximize both the battery life of end-devices and the overall network capacity, the network infrastructure may manage the data rate and radio frequency (RF) transmit power for each end-device individually by means of an adaptive data rate (ADR) scheme.

An end-device may transmit on any channel available at any time using any available data rate, as long as the following rules are observed:

- the end-device changes channels in a pseudo-random fashion for every transmission; the resulting frequency diversity makes the system more robust to interference;
- the end-device pseudo-randomly changes its transmit periodicity to prevent systematic synchronization of populations of end-device transmissions;
- the end-device complies with all local regulations governing its behaviour in the band and sub-bands in which it is currently operating including, but not limited to, duty-cycle and dwell-time (transmit-duration) limitations.

All end-devices shall implement at least Class A functionality as described in this Recommendation. In addition, they may implement Class B and/or Class C also described in this Recommendation.

End-devices implementing Class B are referred to as Class B-capable. When operating in Class B, end-devices are referred to as Class B-enabled. Transition from Class B-disabled to Class B-enabled is called switching to Class B.

End-devices implementing Class C are referred to as Class C-capable. When operating in Class C, end-devices are referred to as Class C-enabled. Transition from Class C-disabled to Class C-enabled is called switching to Class C.

In all cases, end-devices remain compatible with Class A.

6.1 Classes of end-devices

A network distinguishes between a basic protocol (called Class A) and optional features (Class B, Class C…) as shown in Figure 2.

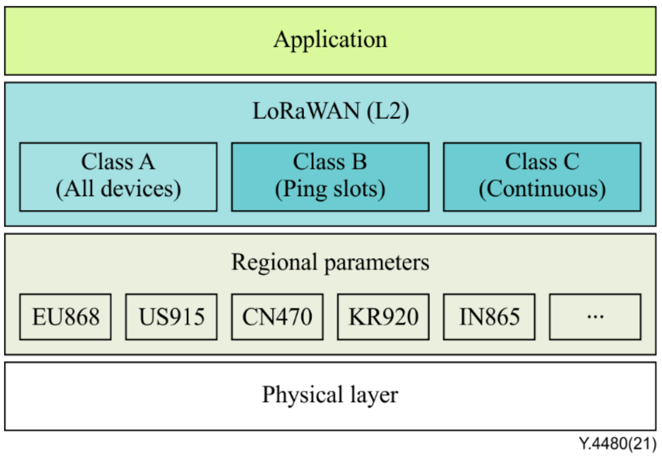

Figure 2 – Classes of end-devices

– **Bidirectional end-devices (Class A)**: Class A end-devices allow bidirectional 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. Class A operation is the lowest power end-device system for applications that require only 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 uplink is initiated by the end-device.

– **Bidirectional end-devices with scheduled receive slots (Class B)**: Class B-capable end-devices allow more receive slots. In addition to Class A receive windows, Class B-enabled end-devices open extra receive windows at scheduled times. In order for the end-device to open its receive windows at a scheduled time, it receives a time-synchronized beacon from the gateway. This allows the network server to know when the end-device is listening.

– **Bidirectional end-devices with maximal receive slots (Class C)**: Class C-capable end-devices allow nearly continuously open receive windows, which are closed only when transmitting. Class C-enabled end-devices use more power to operate than Class A or Class B-enabled, but they feature the lowest latency for communication between servers and end-devices.

Class A

All end-devices

All end-devices SHALL implement all Class A features not explicitly marked optional.

NOTE – Physical packet format, MAC frame format and other parts of this Recommendation that are common to both end-devices of Class A and higher Classes (i.e., end-devices implementing more than Class A) are described only in the Class A specification in order to avoid redundancy.

7 Physical packet formats

The protocol defined in this Recommendation may use any long range physical layer suitable to star the network topology. If an unlicensed spectrum is used, local regulations apply. The protocol has parameters which depend on local regulations and a set of such parameters is referred to as a regional channel plan or region. This Recommendation can handle fixed channel plans where the list of usable channels is fixed, and dynamic channel plans are where, dynamically created channels may be added to the default channels. An example of possible channel plans and physical layer mappings, defined for several regulatory regions, may be found in [b-LoRaWAN RP002].

This Recommendation distinguishes between uplink and downlink frames.

7.1 Uplink packets

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

7.2 Downlink packets

Downlink packets are sent by a network server to only one end-device and are transmitted by a network server through one or more gateways.

NOTE – This Recommendation does not describe the transmission of multicast frames from a network server to many end-devices.

7.3 Receive windows

Following each uplink transmission, the end-device SHALL open one or two receive windows (RX1 and RX2); if no packet destined for the end-device is received in RX1, it SHALL open RX2. The receive windows start times are defined using the end of the transmission as a reference (see Figure 3).

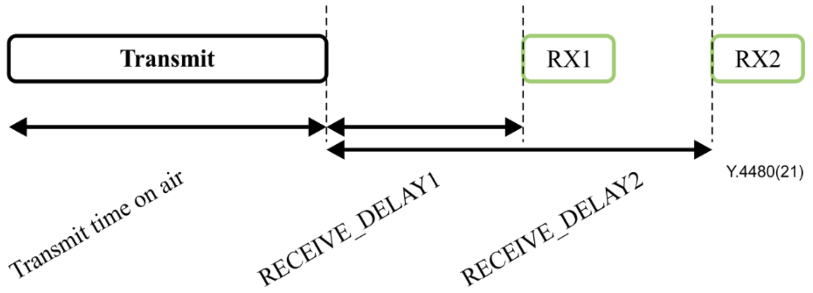

Figure 3 – End-device receive-slot timing

7.3.1 Receiver activity during receive windows

If a preamble is detected during one of the receive windows, the radio receiver SHOULD stay 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 the address and message integrity code (MIC) checks, the end-device SHALL NOT open the second receive window.

7.3.2 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 uplink data rate. RX1 SHALL be open no longer than RECEIVE_DELAY1 seconds after the end of the uplink modulation. The relationship between uplink and RX1 physical layer data rates vary and examples of region-specific definitions may be found in the regional parameters [b-LoRaWAN RP002].

NOTE – RECEIVE_DELAY1 is described in clause 10.

7.3.3 Second receive window channel, data rate, and start

The second receive window RX2, if opened (see clause 7.3.1), uses a fixed configurable frequency and data rate, and SHALL be open no longer than RECEIVE_DELAY2 seconds after the end of the uplink modulation. The frequency and data rate can be modified by MAC commands (see clause 9). The default frequency and data rate depend on the physical layer and the regulatory region.

NOTE – RECEIVE_DELAY2 is described in clause 10.

7.3.4 Receive window duration

The duration of a receive window SHALL be at least the time required by the end-device's radio transceiver to detect a downlink preamble starting at RECEIVE_DELAY1 or RECEIVE_DELAY2 after the end of the uplink modulation.

NOTE – The end-device receive window duration has to accommodate the maximum potential imprecision of the end-device's clock. The delay between the end of the uplink and the start of the receive windows can be changed from 1 s to 15 s for RX1 (and thereby from 2 s to 16 s for RX2) using the over-the-air activation (OTAA) Join-Accept frame or the ***RXTimingSetupReq*** MAC command. Therefore, this delay must be accommodated when computing the maximum clock imprecision.

EXAMPLE – A 30 ppm XTAL frequency error translates to ±30 µs after 1 s and ±450 µs after 15 s.

7.3.5 Network transmitting to an end-device

If a network server intends to transmit a downlink to an end-device, it SHALL initiate the transmission of the downlink frame precisely at the beginning of one of the two receive windows. Such a downlink is referred to as a Class A downlink. The end-device SHALL open a Class A RX1 receive window. If no frame intended for this end-device was received during the RX1 receive window, the end-device SHALL open an RX2 receive window at the specified timing, even if this interrupts the reception of a transmission using Class B or Class C downlink timing and receive parameters.

7.3.6 Important notice regarding receive windows

An end-device SHALL NOT transmit another uplink packet before it has either received a downlink packet in the first or second receive window related to the previous transmission or if the second receive window related to the previous transmission has expired.

7.3.7 Receiving or transmitting other protocols

The end-device MAY listen for or transmit other protocols or perform arbitrary processing between transmission and reception windows, as long as the end-device remains compatible with local regulations and compliant with this Recommendation.

8 MAC frame formats

All uplink and downlink packets carry a PHY payload (`PHYPayload`) starting with a single-octet MAC header (`MHDR`), followed by a MAC payload (`MACPayload`), and ending with a 4-octet `MIC`, as shown in Table 1.

Table 1 – Frame format elements

PHYPayload:

MHDR	MACPayload	MIC

or

MHDR	Join-Request (see clause 10.2.4)	MIC

or

MHDR	Join-Accept (see clause 10.2.4)	MIC

PHYPayload structure

MACPayload:

FHDR	FPort	FRMPayload

MACPayload structure

FHDR:

DevAddr	FCtrl	FCnt	FOpts

Frame header structure

8.1 PHY payload (PHYPayload)

The maximum length M of the MACPayload field depends on the physical layer and regulatory region (see Table 2). A region and data-rate specific example may be found in [b-LoRaWAN RP002]. Neither the end-device nor the network SHALL send a frame containing a MACPayload greater than the specified maximum length M over the data rate used to transmit the frame. Any frame received by an end-device or a network server containing a MACPayload greater than the specified maximum length M over the data rate used to receive the frame SHALL be silently discarded.

Table 2 – PHYPayload format

Size (octets)	1	7..*M*	4
PHYPayload	MHDR	MACPayload	MIC

8.2 MAC header (MHDR field)

The MAC header specifies the frame type (FType) and the major version (Major) of the frame format of this Recommendation according to which the frame has been encoded (see Table 3).

Table 3 – MHDR format

Bits	[7:5]	[4:2]	[1:0]
MHDR	FType	RFU	Major

8.2.1 Frame types (FType bit field)

This Recommendation distinguishes among six different MAC frame types (see Table 4): Join-Request, Join-Accept, unconfirmed data uplink/downlink, and confirmed data uplink/downlink.

Table 4 – MAC frame types

FType	Description
000	Join-Request
001	Join-Accept
010	unconfirmed data uplink
011	unconfirmed data downlink
100	confirmed data uplink
101	confirmed data downlink
110	RFU
111	Proprietary

8.2.1.1 Join-Request and Join-Accept frames

Join-Request and Join-Accept frames are used by the over-the-air activation procedure described in clause 10.2.

8.2.1.2 Data frames

Data frames transfer MAC commands and application data, which can be combined into a single frame. A confirmed data frame SHALL be acknowledged by the receiver, whereas an unconfirmed data frame SHALL NOT be acknowledged. Proprietary frames MAY be used to implement non-standard formats that are not interoperable with standard frames but SHALL be used only among end-devices that have a common understanding of the proprietary extensions.

NOTE – A detailed timing diagram of the acknowledge mechanism is shown in Appendix I.

Frame integrity is ensured in different ways for different frame types and is described per frame type below.

8.2.2 Major data frame version (`Major` bit field)

The major version specifies the format of the frames (see Table 5) exchanged in the join procedure (see clause 10.2.4) and the first four octets of the `MACPayload` as described in clause 8. For each major version, end-devices MAY implement different minor versions of the frame format. The minor version used by an end-device SHALL be made known to the network server beforehand using out-of-band communication (e.g., as part of the end-device personalization information).

Table 5 – Major list

Major	Description
00	LoRaWAN R1
01…11	RFU

8.3 MAC payload of data frames (`MACPayload`)

The MAC payload of data frames contains a frame header (`FHDR`) followed by an OPTIONAL port field (`FPort`) and an OPTIONAL frame payload field (`FRMPayload`).

A frame with a valid `FHDR`, no `FOpts` (`FOptsLen=0`), no `FPort` and no `FRMPayload` is a valid frame.

8.3.1 Frame header (`FHDR`)

An `FHDR` contains the short end-device address (`DevAddr`), a frame control octet (`FCtrl`), a 2-octet frame counter (`FCnt`) and up to 15 octets of frame options (`FOpts`) to transport MAC commands (see Table 6).

Table 6 – `FHDR` format

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

For downlink frames, the `FCtrl` content of the frame header is given in Table 7.

Table 7 – `FCtrl` downlink frames format

Bits	7	6	5	4	[3..0]
`FCtrl`	`ADR`	`RFU`	`ACK`	`FPending`	`FOptsLen`

For uplink frames, the `FCtrl` content of the frame header is given in Table 8.

Table 8 – `FCtrl` uplink frame format

Bits	7	6	5	4	[3..0]
`FCtrl`	`ADR`	`ADRACKReq`	`ACK`	`ClassB`	`FOptsLen`

8.3.1.1 Adaptive data-rate control in frame header (`ADR`, `ADRACKReq` in `FCtrl`)

This Recommendation allows end-devices to use any of the possible data rates and transmission (TX) power individually. This feature is used by network servers to adapt and optimize the number of retransmissions, the data rate, and the TX power of end-devices. This is referred to as the adaptive data rate (ADR) and, when it is enabled, end-devices will be optimized to use the fastest data rate and minimum TX power possible.

ADR control may not be possible when radio channel attenuation changes rapidly and/or continuously. When the network is unable to control the data rate of an end-device, the end-device's application layer SHOULD control it. It IS RECOMMENDED that a variety of different data rates be used in this case. The application layer SHOULD always try to minimize the aggregated airtime, given the network conditions.

If the uplink `ADR` bit is set, the network may control the number of retransmissions, the data rate, and the TX power of the end-device through the appropriate MAC commands. If the `ADR` bit is unset, the network server SHALL accept that the end-device MAY not comply with any attempt to control the number of retransmissions, the data rate, or the TX power of the end-device regardless of the received signal quality. The network MAY still send commands to inform the end-device of the recommended configuration using the ***LinkADRReq*** command. An end-device SHOULD accept the channel mask controls present in ***LinkADRReq***, even when the `ADR` bit is not set. The end-device SHALL respond to all ***LinkADRReq*** commands with a ***LinkADRAns*** indicating which command elements were accepted and which were rejected. This behaviour differs from when the uplink `ADR` bit is set, in which case the end-device accepts or rejects the entire command.

NOTE 1 – A network server may not infer any actual end-device state in the case where the uplink `ADR` bit is not set, regardless of the state of the individual `Status` bits of ***LinkADRAns***. These are provided for offline debugging.

When the downlink `ADR` bit is set, it informs the end-device that the network server is able to send ADR commands. The end-device MAY set/unset the uplink `ADR` bit independently.

When the downlink `ADR` bit is unset, it signals the end-device that, owing to rapid changes of the radio channel, the network temporarily cannot estimate the best data rate. In that case, the end-device has the choice to:

- unset the `ADR` uplink bit and control its uplink data rate, TX power and channel plan following its own strategy. This SHOULD be the typical strategy for a mobile end-device; or

- ignore it (keep the uplink `ADR` bit set) and apply the normal ADR backoff algorithm in the absence of downlinks. This SHOULD be the typical strategy for a stationary end-device.

The `ADR` bit MAY be set and unset on demand by the end-device or a network server. However, whenever possible, the ADR scheme SHOULD be enabled to increase the battery life of the end-device and maximize the network capacity.

NOTE 2 – Even mobile end-devices are immobile most of the time. Depending on its state of mobility, an end-device can request that the network optimize its data rate using ADR.

Default TX power is the maximum transmit power allowed for an end-device, considering its capabilities and any applicable regional regulatory constraints. End-devices SHALL use this power level until the network attempts to change it using the ***LinkADRReq*** MAC command, or if the end-device has unset the `ADR` bit.

The default data rate is the minimum data rate allowed for an end-device, considering its capabilities and any applicable regional regulatory constraints. An end-device using activation by personalization (ABP, see clause 10), with the `ADR` bit set, SHALL use this data rate until the network requests a higher data rate through the ***LinkADRReq*** MAC command. End-devices which use OTAA follow a join procedure (see clause 10) which determines the initial uplink data rate.

If an end-device wishes to check for connectivity loss or if an end-device whose data rate is optimized by the network uses a data rate higher than its default data rate or a TX power lower than its default, the end-device SHALL periodically validate whether the network is still receiving the uplink frames. Each time the uplink frame counter is incremented (for each new uplink frame, because repeated transmissions do not increment the frame counter), the end-device SHALL increment an ADRACKCnt counter. After ADR_ACK_LIMIT uplinks (ADRACKCnt ≥ ADR_ACK_LIMIT) without receiving a Class A downlink response, the end-device SHALL set the ADR acknowledgment request bit (`ADRACKReq`) on uplink transmissions. The network shall respond with a Class A downlink frame within the next ADR_ACK_DELAY frames. A Class A downlink frame received following an uplink frame SHALL reset the ADRACKCnt counter. Upon receipt of any Class A downlink, the end-device SHALL clear the `ADRACKReq` bit. The downlink ACK bit does not need to be set because any Class A downlink frame received by the end-device indicates that the network has received uplinks from this end-device. If no Class A downlink frame is received within the next ADR_ACK_DELAY uplinks (i.e., after a total of ADR_ACK_LIMIT + ADR_ACK_DELAY transmitted frames), the end-device SHALL try to regain connectivity by first setting the TX power to the default power, then switching to the next lower data rate that provides a longer radio range. The end-device SHALL further lower its data rate step by step every time ADR_ACK_DELAY uplink frames are transmitted. Once the end-device has reached the default data rate, and transmitted for ADR_ACK_DELAY uplinks with ADRACKReq=1 without receiving a downlink, it SHALL re-enable all default uplink frequency channels and reset `NbTrans` to its default value of 1. Furthermore, if at any point during the backoff the resulting configuration results in an invalid combination of TX power, data rate or channel mask, the end-device SHALL immediately re-enable all default channels and use the maximum TX power permissible for and available to this end-device.

NOTE 3 – Other configurations of the end-device by the network are not modified during ADR backoff. Specifically, configurations that affect downlink connectivity (controlled by ***RXParamsSetupReq***, ***DlChannelReq***, ***RXTimingSetupReq***, and ***TXParamSetupReq***) are not modified during ADR backoff.

For fixed channel plan regions, end-devices SHALL enable all channels. For dynamic channel plan regions, end-devices SHALL enable the region's default channels and make no change to the configuration of the dynamically configured channels.

NOTE 4 – Not requiring an immediate response to an ADR acknowledgment request provides flexibility to the network to schedule its downlinks in an optimal manner.

Table 9 provides an example of a data rate backoff sequence, assuming ADR_ACK_LIMIT is set to 64 and ADR_ACK_DELAY is equal to 32.

Table 9 – Example of a data rate backoff sequence

ADRACKCnt	`ADRACKReq`	Data rate	TX power	`NbTrans`	Channel mask
0 to 63	`0`	DR1	Max –9 dBm	`3`	Normal operations channel mask
64 to 95	`1`	No change	No change	No change	No change
96 to 127	`1`	No change	Default	No change	No change
128 to 159	`1`	DR0 (Default)	Default	No change	No change
≥ 160	`1`	DR0 (Default)	Default	**`1`**	For dynamic channel plans: re-enable default channels. For fixed channel plans: All channels enabled

8.3.1.2 Frame acknowledge bit and acknowledgment procedure (`ACK` in `FCtrl`)

When receiving a confirmed data frame, the receiver responds with a data frame that has the acknowledgment bit (`ACK`) set. If the network receives such a confirmed frame, it SHOULD send an acknowledgment. If the network sends an acknowledgement, it SHALL send it using one of the Class A receive windows opened by the end-device after the send operation. If an end-device receives such a confirmed frame in one of its Class A receive windows, it SHALL transmit an acknowledgment with its next uplink. If an end-device receives such a confirmed frame outside of its Class A receive windows, i.e., in a Class B ping slot (see clause 13) or in RXC (see clause 19), it SHOULD send an acknowledgement.

Acknowledgments SHALL only be sent in response to the latest frame received and SHALL NOT be transmitted more than `NbTrans` times. The network SHALL only send an acknowledgment in the Class A receive windows (RX1/RX2) of the confirmed uplink that requested it.

NOTE – To allow an end-device to be as simple as possible and have as few states as possible, it may transmit an explicit (possibly empty) acknowledgment data frame immediately after receiving a data frame requiring a confirmation. Alternatively, an end-device may defer the transmission of an acknowledgment to piggyback it with its next data frame.

8.3.1.3 Retransmission procedure

Downlink frames

A downlink confirmed or unconfirmed frame SHALL NOT be retransmitted using the same frame counter value. In the case of a confirmed downlink, if the acknowledgement is not received, the application server is notified and may decide to transmit a new confirmed frame.

Uplink frames

Uplink confirmed and unconfirmed frames are transmitted `NbTrans` times (see clause 9.2) unless a valid Class A downlink is received following one of the transmissions. The `NbTrans` parameter can be used by a network server to control the redundancy of end-device uplinks to achieve a given quality of service. An end-device SHALL perform frequency hopping as usual between repeated transmissions. It SHALL wait after each repetition until the receive windows have expired. The delay between retransmissions is at the discretion of the end-device and MAY be different for each end-device. When an end-device has requested an ACK from the network but has not yet received it, it SHALL wait RETRANSMIT_TIMEOUT seconds after the RECEIVE_DELAY2 seconds have elapsed, after the end of the previous uplink transmission before sending a new uplink (repetition or new frame). The RETRANSMIT_TIMEOUT delay is not required between unconfirmed uplinks, or after the ACK has been successfully demodulated by the end-device. An example of the RETRANSMIT_TIMEOUT value may be found in [b-LoRaWAN RP002]. The retransmission backoff mechanism, defined in clause 11, may also extend the interval between retransmissions, if applicable.

End-devices SHALL stop any further retransmission of an uplink confirmed frame if a corresponding downlink acknowledgment frame is received.

End-devices SHALL also stop any further retransmission of an uplink unconfirmed frame whenever a valid downlink frame is received in a Class A receive window.

If the network receives more than `NbTrans` transmissions of the same uplink frame having the ADR bit set, this may indicate a replay attack or a malfunctioning end-device, and therefore the network SHALL silently discard the extra frames.

NOTE – A network server that detects a replay attack may take additional measures, such as reducing the `NbTrans` parameter to 1 or discarding uplink frames received over a channel that was already used by an earlier transmission of the same frame, or by some other unspecified mechanism.

8.3.1.4 Frame pending bit (`FPending` in `FCtrl`, downlink only)

The frame pending bit (`FPending`) is used only in downlink communication. For Class A end-devices, `FPending` indicates that the network server has more data pending to be sent and therefore the end-device MAY send an uplink frame as soon as possible. For Class B end-devices, `FPending` indicates the priority in which conflicting ping slots of the end-device SHALL listen to in case of a collision (see clause 13.2).

An example use of the `FPending` bit is described in clause I.4.

8.3.1.5 Frame counter (`FCnt`)

There are two frame counters for each end-device. `FCntUp` is incremented by an end-device when a data frame is transmitted to a network server (uplink). `FCntDown` is incremented by a network server when a data frame is transmitted to an end-device (downlink). The network server tracks the uplink frame counter and generates the downlink counter for each end-device.

Whenever an OTAA end-device successfully processes a Join-Accept frame, the frame counters on the end-device (`FCntUp`) and the network side (`FCntDown`) are reset to `0`.

For ABP end-devices, the frame counters are initialized to `0` by the manufacturer. ABP end-devices SHALL NOT reset the frame counters during the end-device's lifetime. If the end-device is susceptible to losing power during its lifetime (battery replacement, for example), the frame counters SHALL persist during such an event.

Subsequently, `FCntUp` is incremented with each uplink and `FCntDown` is incremented with each downlink. At the receiver side, the corresponding counter is kept in sync with the received value, provided the received value has been incremented compared to the current counter value, and the frame MIC field matches the MIC value computed locally using the appropriate network session key. `FCntUp` SHALL NOT be incremented in the case of multiple transmissions of a confirmed or unconfirmed frame (see `NbTrans` parameter). Network servers SHALL drop the application payload of the retransmitted frames and only forward a single instance to the appropriate application server.

A first uplink with `FCntUp=0` sent by an ABP or an OTAA end-device after a successful join procedure SHALL be accepted by a network server, provided the `MIC` field is valid. Analogously, a first downlink with FCntDown=0 sent by a network server to an ABP or an OTAA end-device after a successful join procedure SHALL be accepted by the end-device, provided the MIC field is valid.

Frame counters are 32 bits wide. The FCnt field SHALL correspond to the least-significant 16 bits of the 32-bit 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 with the same application or network session keys, except for retransmission.

The end-device SHALL NOT process any retransmission of the same downlink frame. Subsequent retransmissions SHALL be ignored without being processed.

NOTE 1 – This means that an end-device will only acknowledge receipt of a downlink confirmed frame NbTrans times. Similarly, an end-device will only generate NbTrans uplinks following receipt of a frame with the FPending bit set before incrementing its FCntUp.

NOTE 2 – As the FCnt field carries only the least-significant 16 bits of the 32-bit frame counter, the server must infer the 16 most-significant bits of the frame counter by observing the traffic.

8.3.1.6 Frame options (FOptsLen in FCtrl, FOpts)

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

FOpts transports MAC commands of a maximum length of 15 octets that are piggybacked onto data frames; see clause 9 for a list of valid MAC commands.

If FOptsLen=0, the FOpts field SHALL be absent. If FOptsLen≠0, i.e., if MAC commands are present in the FOpts field, the FPort value 0 SHALL NOT be used (FPort SHALL either not be present or not equal to 0).

MAC commands SHALL NOT be present in the payload field and the frame options field simultaneously. Should this occur, the end-device SHALL silently discard the frame.

8.3.1.7 Class B enabled bit (ClassB in FCtrl, uplink only)

The ClassB bit set to 1, in an uplink, signals to the network server that the end-device has enabled class B and is now ready to receive scheduled downlink pings. Please refer to the Class B section of this Recommendation for the Class B specification.

8.3.2 Port field (FPort)

If the frame payload field is not empty, the port field SHALL be present (see Table 10).

If present, an FPort value of 0 indicates that the FRMPayload contains only MAC commands (see clause 9 for a list of valid MAC commands).

FPort values 1..223 (0x01..0xDF) are application-specific.

FPort value 224 is dedicated to the MAC layer test protocol.

NOTE – The purpose of the FPort value 224 is to run MAC compliance test scenarios over-the-air on final versions of end-devices, without having to rely on specific test versions of end-devices for practical aspects. The test protocol running at the application layer is defined in [b-LoRaWAN TS009].

FPort values 225..255 (0xE0..0xFF) are reserved for future use.

Table 10 – MACPayload format

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

In Table 10, N is the number of octets of the application payload and SHALL be equal to or less than $N \leq M - 1 -$ (length of FHDR in octets), where M is the maximum MACPayload length. The valid ranges of both N and M depend on the physical layer and the regional channel plan.

8.3.3 MAC frame payload encryption (FRMPayload)

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

The encryption scheme is based on the generic algorithm described in [IEEE 802.15.4] Annex B, using advanced encryption standard (AES) encryption with a key length of 128 bits. AES encryption is defined in [NIST FIPS 197].

Key K depends on the FPort of the data frame (see Table 11).

Table 11 – FPort list

FPort	***K***
0	NwkSKey
1..255	AppSKey

The encrypted fields are *pld* = FRMPayload.

For each data frame, the algorithm defines a sequence of blocks A_i as specified in Table 12, for i = 1..k where:

- k = ceil(len(*pld*) / 16),
- "ceil" designates rounding to the immediately greater integer value, and
- "len" is the number of bytes.

Table 12 – A_i format

Size (octets)	1	4	1	4	4	1	1
A_i	0x01	4 × 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 SHALL be encrypted to obtain a sequence S of blocks S_i as follows:

$S_i = \text{aes128\_encrypt}(K, A_i)$ for $i = 1..k$
$S = S_1 \mid S_2 \mid .. \mid S_k$

Encryption and decryption of the payload SHALL be calculated as follows:

FRMPayloadPad = $(pld \mid \text{pad}_{16})$ xor S
FRMPayload = FRMPayloadPad[0..len(*pld*) – 1]

8.4 Message integrity code (MIC)

The MIC is calculated over all the fields in the frame.

msg = MHDR | FHDR | FPort | FRMPayload

where len(*msg*) denotes the length of the frame in octets.

The MIC SHALL be calculated as follows, where CMAC is the cipher-based message authentication code (see [IETF RFC 4493]):

CMAC = aes128_cmac(NwkSKey, $B_0 \mid msg$)
MIC = CMAC[0..3]

where block B_0 is defined in Table 13.

Table 13 – B_0 format

Size (octets)	1	4	1	4	4	1	1
B_0	`0x49`	4 × `0x00`	`Dir`	`DevAddr`	`FCntUp` or `FCntDown`	`0x00`	len(*msg*)

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

9 MAC commands

For network administration, a set of MAC commands may be exchanged exclusively between a network server and the MAC layer of an end-device. MAC layer commands are never visible to the application server, nor to the application running on the end-device.

A single data frame MAY 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 set to `0`. Piggybacked MAC commands SHALL always be sent without encryption and SHALL NOT exceed 15 octets. MAC commands sent as `FRMPayload` SHALL always be encrypted and SHALL NOT exceed the maximum `FRMPayload` length.

NOTE 1 – MAC commands whose content shall be encrypted must be sent in the `FRMPayload` of a separate data frame.

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

Table 14 – MAC commands

CID	Command	Transmitted by		Brief description
		End-device	Network server	
`0x02`	***LinkCheckReq***	x		Used by an end-device to validate its connectivity to a network.
`0x02`	***LinkCheckAns***		x	Answers ***LinkCheckReq***. Contains the received signal power estimation, which indicates the quality of reception (link margin) to the end-device.
`0x03`	***LinkADRReq***		x	Requests the end-device to change data rate, TX power, redundancy, or channel mask.
`0x03`	***LinkADRAns***	x		Acknowledges ***LinkADRReq***.
`0x04`	***DutyCycleReq***		x	Sets the maximum aggregated transmit duty cycle of an end-device.
`0x04`	***DutyCycleAns***	x		Acknowledges ***DutyCycleReq***.
`0x05`	***RXParamSetupReq***		x	Sets the reception slot parameters. NOTE – This command has a different acknowledgment mechanism as described in the command definition.
`0x05`	***RXParamSetupAns***	x		Acknowledges ***RXParamSetupReq.***
`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 radio status.
`0x07`	***NewChannelReq***		x	Creates or modifies the definition of a radio channel.
`0x07`	***NewChannelAns***	x		Acknowledges ***NewChannelReq***.
`0x08`	***RXTimingSetupReq***[7]		x	Sets the timing of the reception slots.
`0x08`	***RXTimingSetupAns***	x		Acknowledges ***RXTimingSetupReq***.
`0x09`	***TXParamSetupReq***[7]		x	Used by a network server to set the maximum allowed dwell time and `MaxEIRP` of end-device, based on local regulations.
`0x09`	***TXParamSetupAns***	x		Acknowledges ***TXParamSetupReq***.
`0x0A`	***DlChannelReq***[7]		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***.
`0x0B` to `0x0C`	RFU			
`0x0D`	***DeviceTimeReq***	x		Used by an end-device to request the current GPS time.
`0x0D`	***DeviceTimeAns***		x	Answers ***DeviceTimeReq***.
`0x0E` to `0x0F`	RFU			
`0x10` to `0x1F`	Class B commands (see clause 16).			
`0x20` to `0x2F`	Reserved for Class C commands.			
`0x30` to `0x7F`	RFU			
`0x80` to `0xFF`	Proprietary	x	x	Reserved for proprietary network command extensions.

MAC commands which require an answer from the network expire after the Class A receive windows have elapsed.

MAC commands are answered/acknowledged by the receiving end in the same order they were transmitted. The answer to each MAC command is sequentially added to a buffer. All MAC commands received in a single frame SHALL be answered in a single frame, which means that the buffer containing the answers SHALL be sent in a single frame.

If the transmitter has a combination of application payload and MAC answers, or new MAC commands to send and they cannot fit in the same frame, the priority for including information in the frame is shown in Table 15. Within a single frame, a transmitter SHALL send all the higher-priority information before sending any lower-priority information.

Table 15 – Transmit data insertion prioritization

Priority level	Information type
Highest	MAC answers
	New MAC commands
Lowest	Application payload

NOTE 2 – MAC answers are defined as MAC commands sent by the transmitter in response to the received MAC commands. New MAC commands are defined as MAC commands sent by the transmitter but not in response to a received MAC command.

If the MAC command buffer is too large to fit in the frame, the transmitter SHALL truncate the buffer at the end of the last MAC command that is able to fit within the frame. In all the cases, the full list of MAC commands SHALL be executed by the receiver, even if the buffer containing the MAC answers must be truncated.

NOTE 3 – When receiving a truncated MAC answer, a network server may retransmit the MAC commands that could not be answered. The network server may elect to send a list of MAC commands, which cannot be answered in a single frame, in order to transition the end-device rapidly to an optimal configuration.

NOTE 4 – In general, the transmitter will reply once to a MAC command. If the answer is lost, the original sender must resend the command. The original sender decides that the command must be resent when it receives a new frame that does not contain the answer. Only the ***RXParamSetupReq***, ***RXTimingSetupReq***, ***TXParamSetupReq*** and ***DlChannelReq*** commands impact the downlink parameters and therefore have a different acknowledgment mechanism as described in their corresponding clauses.

NOTE 5 – When a MAC command is initiated by an end-device, the network server may only send the acknowledgment/answer in the RX1/RX2 windows immediately following the request. If the answer is not received in that slot, the end-device is free to implement any retry mechanism it requires.

NOTE 6 (backwards compatibility) – The length of a MAC command is variable and is determined during decoding. 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 define out-of-band the lowest common specification version of the network server and the end-device. Without such knowledge, the order of the MAC commands shall be according to the version of the protocol specification (i.e., a version that predates this Recommendation) that introduced a MAC command for the first time. This way, all the MAC commands up to the implemented version of this protocol can be processed even in the presence of the MAC commands specified in newer versions of this protocol.

9.1 Link check commands (*LinkCheckReq, LinkCheckAns*)

End-devices and network servers SHALL implement these commands.

An end-device MAY use the ***LinkCheckReq*** command to 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, the network server SHALL respond with a ***LinkCheckAns*** command.

Table 16 – *LinkCheckAns* payload format

Size (octets)	1	1
***LinkCheckAns* payload**	`Margin`	`GwCnt`

The demodulation margin (`Margin`) is an 8-bit unsigned integer in the range of 0 to 254, which indicates the link margin in dB of the most recently transmitted ***LinkCheckReq*** command. A value of 0 means that the frame was received at the demodulation floor (0 dB or no margin) whereas a value of 20, for example, means that the frame reached the best gateway 20 dB above the demodulation floor. The value 255 is reserved.

The gateway count (`GwCnt`) is the number of gateways that received the most recent ***LinkCheckReq*** command.

9.2 Link ADR commands (*LinkADRReq*, *LinkADRAns*)

End-devices and network servers SHALL implement these commands.

A network server MAY use the ***LinkADRReq*** command to request that an end-device performs a rate adaptation (see Tables 17 and 18).

Table 17 – *LinkADRReq* payload format

Size (octets)	1	2	1
***LinkADRReq* payload**	`DataRate_TXPower`	`ChMask`	`Redundancy`

Table 18 – `DataRate_TXPower` field format

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

The requested date rate (`DataRate`) and the TX output power (`TXPower`) encoding are physical layer and regulatory region dependent. A region-specific example of encoding may be found in [b-LoRaWAN RP002]. The TX output power indicated in the command is to be considered the maximum TX power at which the end-device may operate. An end-device SHALL acknowledge the successful receipt of a command that specifies a higher TX power than it is capable of using. In that case, the end-device SHALL operate at its maximum possible power. An end-device will negatively acknowledge a command that specifies a lower TX power than the end-device is capable of using. In that case, the end-device SHALL operate at its previously configured TX power. The value 0xF (decimal 15) of either `DataRate` or `TXPower` means that the end-device SHALL ignore that field and keep the current parameter values. An end-device SHALL support a minimum power control range, such that it can operate from its maximum TX power down to the max (2 dBm, maximum TX power – 14 dB). It IS RECOMMENDED that an end-device supports a minimum TX power of +2 dBm.

NOTE – In case of good RF conditions, the network should slowly lower an end-device's TX power and/or raise the end-device's data rate to avoid making drastic changes that may strand the end-device.

The channel mask (`ChMask`) SHALL encode the channels usable for uplink access as follows with bit 0 corresponding to the least significant bit (see Table 19).

Table 19 – Channel mask format

`Bits`	**`Usable channels`**
0	Channel 1
1	Channel 2
..	..
15	Channel 16

If a bit in the `ChMask` field is set to `1`, the corresponding channel SHOULD be used for uplink transmissions if this channel allows the data rate currently used by the end-device. A bit set to `0` means that the corresponding channel SHALL NOT be used.

Table 20 – Redundancy field format

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

The requested date rate (`DataRate`) and the TX output power (`TXPower`) encoding are physical layer and regulatory region dependent. A region-specific example of encoding may be found in [b-LoRaWAN RP002]. The TX output power indicated in the command is to be considered the maximum TX power at which the end-device may operate. An end-device SHALL acknowledge the successful receipt of a command that specifies a higher TX power than it is capable of using. In that case, the end-device SHALL operate at its maximum possible power. An end-device will negatively acknowledge a command that specifies a lower TX power than the end-device is capable of using. In that case, the end-device SHALL operate at its previously configured TX power. The value 0xF (decimal 15) of either `DataRate` or `TXPower` means that the end-device SHALL ignore that field and keep the current parameter values. An end-device SHALL support a minimum power control range, such that it can operate from its maximum TX power down to the max (2 dBm, maximum TX power – 14 dB). It IS RECOMMENDED that an end-device supports a minimum TX power of +2 dBm.

In the `Redundancy` bits (see Table 20), the `NbTrans` field is the number of transmissions for each uplink frame. This applies to both confirmed and unconfirmed uplink frames. The default value is `1`, which corresponds to a single transmission of each frame. The valid range is `[1:15]`. If an `NbTrans` value of `0` is received, the end-device SHALL use the default value. This field MAY be used by a network server to control the redundancy of the uplink transmissions (retransmissions) to obtain a given quality of service. The end-device performs frequency hopping for retransmissions as usual, and it waits as usual after each retransmission until RX2 has expired. Whenever a downlink frame is received during either RX1 or RX2, the end-device SHALL stop any further retransmission of that same uplink frame.

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 turn all channels on or off globally using a specific modulation. The meaning of `ChMaskCntl` depends on the physical layer and regulatory region. A region-specific example may be found in [b-LoRaWAN RP002].

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

Valid channel frequencies depend on the physical layer and regulatory region of operation.

An end-device SHALL answer a ***LinkADRReq*** with a ***LinkADRAns*** command (see Tables 21 and 22).

Table 21 – *LinkADRAns* payload format

Size (octets)	1
***LinkADRAns* payload**	`Status`

Table 22 – `Status` field format

`Bits`	[7:3]	2	1	0
`Status`	RFU	PowerACK	DataRateACK	ChannelMaskACK

The ***LinkADRAns*** `Status` bits have the meaning specified in Table 23.

Table 23 – *LinkADRAns* `Status` bits signification

	`Bit=0`	`Bit=1`
`ChannelMaskACK`	The channel mask enables a yet undefined channel, or the channel mask required all the channels to be disabled or the channel mask is incompatible with the resulting data rate or TX power. 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.
`DataRateACK`	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.
`PowerACK`	The end-device is unable to operate at or is below the requested power level. The command was discarded, and the end-device state was not changed.	The power level was successfully set.

If any of the three bits equals `0`, the command did not succeed, and the end-device SHALL keep its previous state.

9.3 End-device transmit duty cycle (*DutyCycleReq*, *DutyCycleAns*)

End-devices and network servers SHALL implement these commands.

The ***DutyCycleReq*** command can be used by the network to limit the maximum aggregated transmit duty cycle of an end-device (see Tables 24 and 25). The aggregated transmit duty cycle corresponds to the transmit duty cycle over all sub-bands.

Table 24 – *DutyCycleReq* payload format

Size (octets)	1
***DutyCycleReq* payload**	`DutyCyclePL`

Table 25 – `DutyCyclePL` field format

`Bits`	`7:4`	`3:0`
`DutyCyclePL`	`RFU`	`MaxDutyCycle`

The maximum end-device transmit duty cycle allowed is:

$$\text{Aggregated duty cycle} = 1/2^{\texttt{MaxDutyCycle}}$$

The valid range for `MaxDutyCycle` is `[0:15]`. A value of `0` corresponds to 100 % duty cycle, therefore "no duty cycle limitation" except the one set by the regional regulation.

NOTE 1 – When applying a ***DutyCycleReq*** command, the end-device will use whichever is the lowest i.e., either the region limitation for the specific sub-band, or the value from the restriction defined by ***DutyCycleReq*** which is aggregated over all sub-bands.

When `MaxDutyCycle` is different than `0`, an end-device SHALL respect a silence period T_{off} before transmitting a frame on any channel. This silence ensures that the duty cycle limit is met even over short observation windows. With *TimeOnAir* defined as the duration of the transmitted frame, the silence is computed as:

$$T_{off} = TimeOnAir * (2^{\texttt{MaxDutyCycle}} - 1)$$

NOTE 2 – The ***DutyCycleReq*** command is used by the network server for traffic shaping, to limit the transmissions from a given end-device. It is calculated by the above formula, which may be different than the duty cycle measurement methods defined by the regulations which have such a limitation.

An end-device SHALL answer a ***DutyCycleReq*** with a ***DutyCycleAns*** command. The ***DutyCycleAns*** MAC reply contains no payload.

9.4 Receive windows parameters (*RXParamSetupReq*, *RXParamSetupAns*)

End-devices and network servers SHALL implement these commands.

The ***RXParamSetupReq*** command allows a change to the frequency and the data rate set for RX2 following each uplink (see Table 26). The command also allows an offset to be programmed between the uplink and the RX1 slot downlink data rates (see Table 27).

Table 26 – *RXParamSetupReq* payload format

Size (octets)	1	3
***RXParamSetupReq* payload**	DLSettings	Frequency

Table 27 – `DLSettings` field format

Bits	7	6:4	3:0
DLSettings	RFU	RX1DROffset	RX2DataRate

The RX1 data-rate offset (`RX1DROffset`) field sets the offset between the uplink data rate and the downlink data rate used to communicate with the end-device on RX1. The default offset is 0. The offset takes into account the maximum power density constraints for gateways in some regions and balances the uplink and downlink radio link margins.

The reception (RX) data rate (`RX2DataRate`) field defines the data rate of a downlink using the second receive window following the same convention as the ***LinkADRReq*** command. For example, 0 means DR0/125 kHz. 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 SHALL be used by the end-device to acknowledge the receipt of an ***RXParamSetupReq*** command. The ***RXParamSetupAns*** command SHALL be added in the `FOpts` field (if `FPort` is either missing or > `0`) or in the `FRMPayload` field (if `FPort = 0`) of all uplinks until a Class A downlink is received by the end-device. This guarantees that, even in the case of an uplink frame loss, the network is always aware of the downlink parameters used by the end-device.

Following the transmission of an ***RXParamSetupReq*** command that modifies RX2 (`Frequency` or `RX2DataRate` fields), the network server SHALL NOT transmit a Class C downlink before it has received a valid uplink frame containing ***RXParamSetupAns***.

NOTE – An end-device that expects to receive Class C downlink frames will send an uplink frame as soon as possible after receiving a valid ***RXParamSetupReq*** that modifies RX2 (`Frequency` or `RX2DataRate` fields).

The payload contains a single `Status` octet (see Tables 28 and 29).

Table 28 – *RXParamSetupAns* payload format

Size (octets)	1
***RXParamSetupAns* payload**	Status

Table 29 – `Status` field format

Bits	7:3	2	1	0
Status	RFU	RX1DROffsetACK	RX2DataRateACK	ChannelACK

The status (`Status`) bits have the meaning specified in Table 30.

Table 30 – *RX2SetupAns* `Status` bits signification

	`Bit=0`	`Bit=1`
`ChannelACK`	The frequency requested is not usable by the end-device.	RX2 slot channel was successfully set
`RX2DataRateACK`	The data rate requested is unknown to the end-device.	RX2 slot data rate was successfully set
`RX1DROffsetACK`	The uplink/downlink data rate offset for RX1 slot is not within the allowed range	RX1 data-rate offset was successfully set

If any of the three bits is equal to `0`, the command did not succeed, and the end-device SHALL keep its previous state.

9.5 End-device status (*DevStatusReq*, *DevStatusAns*)

End-devices and network servers SHALL implement these commands.

A network server can use the ***DevStatusReq*** command to request status information from an end-device. The command has no payload. If a ***DevStatusReq*** is received by an end-device, it SHALL respond with a ***DevStatusAns*** command (see Table 31).

Table 31 – *DevStatusAns* payload format

Size (octets)	1	1
***DevStatusAns* payload**	`Battery`	`RadioStatus`

The reported battery level (`Battery`) is encoded as specified in Table 32.

Table 32 – Battery-level decoding

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

The `RadioStatus` field contains the signal-to-noise ratio (SNR) information encoded in the six lowest bits in dB rounded to the nearest integer value for the last successfully received ***DevStatusReq*** command. It is a signed integer with a minimum value of −32 and a maximum value of 31.

Table 33 – Status field format

`Bits`	`7:6`	`5:0`
`RadioStatus`	`RFU`	`SNR`

9.6 Creation / modification of a channel (NewChannelReq, NewChannelAns, DlChannelReq, DlChannelAns)

End-devices and network servers SHALL implement these commands, unless an end-device is operating in a region where a fixed channel plan is defined, in which case these commands SHALL NOT be implemented.

A network server can use the ***NewChannelReq*** command either to create a new bidirectional channel or to modify the parameters of an existing one. The command sets the centre frequency of the new channel and the range of uplink data rates that are usable on this channel (see Table 34).

Table 34 – *NewChannelReq* payload format

Size (octets)	1	3	1
***NewChannelReq* payload**	`ChIndex`	`Frequency`	`DRRange`

The channel index (`ChIndex`) is the index of the channel being created or modified. Depending on the regulatory region of operation, frequency band, and physical layer, default channels SHALL be defined as default channels SHALL be common to all end-devices within a regulatory region and SHALL NOT be modified by the ***NewChannelReq*** command. 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. An end-device SHALL be able to handle at least 16 different channel definitions.

The `Frequency` field is a 24-bit unsigned integer. The actual channel frequency (in Hz) is 100 × `Frequency`, whereby values representing frequencies below 100 MHz are reserved for future use. This allows the frequency of a channel to be set anywhere from 100 MHz to 1.67 GHz in increments of 100 Hz. A `Frequency` value of `0` disables the channel. The end-device SHALL check that the frequency is allowed by its radio hardware and SHALL NOT set the channel frequency bit in the `Status` field of the answer if the end-device cannot use this frequency (see below).

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 as shown in Table 35.

Table 35 – `DRRange` field format

Bits	7:4	3:0
DRRange	MaxDR	MinDR

The minimum data rate (`MinDR`) subfield designates the lowest uplink data rate allowed on this channel. The maximum data rate (`MaxDR`) designates the highest uplink data rate. An example of mapping of the data rate index to physical layer may be found in [b-LoRaWAN RP002].

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

The end-device SHALL acknowledge the receipt of a ***NewChannelReq*** by returning a ***NewChannelAns*** command. The payload of this frame contains the information of Table 36.

Table 36 – NewChannelAns payload format

Size (octets)	1
***NewChannelAns* payload**	Status

The status (`Status`) bits have the meaning given in Tables 37 and 38.

Table 37 – `Status` field format

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

Table 38 – *NewChannelAns* `Status` bits signification

	Bit=0	**Bit=1**
Data-rate range ok	The designated data-rate range exceeds the ones currently defined for this end-device	The data-rate range is compatible with the capabilities of the end-device
Channel frequency ok	The end-device cannot use this frequency	The end-device is able to use this frequency.

If either of those bits equals `0`, the command did not succeed, and the end-device SHALL keep its previous state.

The ***DlChannelReq*** command allows the network to associate a different downlink frequency with the RX1 slot. This command is applicable to dynamic channel plans. In regions where that command is not defined, the end-device SHALL silently drop it.

The command sets the center frequency for the downlink RX1 slot as specified in Table 39.

Table 39 – *DlChannelReq* payload format

Size (octets)	1	3
***DlChannelReq* payload**	`ChIndex`	`Frequency`

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

The frequency (`Frequency`) field is a 24-bit unsigned integer. The actual downlink frequency (in Hz) is 100 × `Frequency`, where values representing frequencies below 100 MHz are reserved for future use. The end-device SHALL check that the frequency is allowed by its radio hardware and return an error otherwise.

NOTE – To revert the RX1 frequency to the default value, a network server can send another ***DlChannelReq*** with the same value as uplink frequency.

If the ***DlChannelReq*** command is defined in the region where the end-device is operating, the end-device SHALL acknowledge the receipt of a ***DlChannelReq*** by returning a ***DlChannelAns*** command. The ***DlChannelAns*** command SHALL be added to the `FOpts` field (if `FPort` is either missing or >`0`) or to the `FRMPayload` field (if `FPort=0`) of all uplinks until a Class A downlink is received by the end-device. This guarantees that, even in the case of an uplink frame loss, the network is always aware of the downlink frequencies used by the end-device.

The payload of this frame contains the information specified in Table 40.

Table 40 – *DlChannelAns* payload format

Size (octets)	1
***DlChannelAns* payload**	`Status`

The status (`Status`) bits have the meaning given in Tables 41 and 42.

Table 41 – `Status` field format

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

Table 42 – *DlChannelAns* `Status` bits signification

	`Bit=0`	**`Bit=1`**
`Channel frequency ok`	The end-device cannot use this frequency	The end-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

If either of those bits equals `0`, the command did not succeed, and the end-device SHALL keep its previous state.

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

End-devices and network servers SHALL implement these commands.

A network server can use the ***RXTimingSetupReq*** command to configure the delay between the end of the TX uplink transmission and the opening of RX1 (see Table 43). RX2 SHALL always open 1 second after RX1.

Table 43 – *RXTimingSetupReq* payload format

Size (octets)	1
***RXTimingSetupReq* payload**	`RxTimingSettings`

The `RxTimingSettings` field specifies the delay. The field is split in two 4-bit indexes as shown in Table 44.

Table 44 – *RxTiminingSettings* field format

Bits	7:4	3:0
RxTimingSettings	RFU	Del

The delay is expressed in seconds. `Del 0` is mapped to 1 s.

Table 45 – `Del` mapping

Del	Delay [s]
0	1
1	1
2	2
3	3
…	…
15	15

An end-device SHALL answer ***RXTimingSetupReq*** with ***RXTimingSetupAns***, which has no payload.

The ***RXTimingSetupAns*** command SHALL be added to the `FOpts` field (if `FPort` is either missing or greater than `0`) or to the `FRMPayload` field (if `FPort=0`) of all uplinks until a Class A downlink is received by the end-device. This guarantees that, even in the case of an uplink frame loss, the network is always aware of the downlink parameters used by the end-device.

9.8 End-device transmit parameters (*TXParamSetupReq*, *TXParamSetupAns*)

A network server MAY use the ***TXParamSetupReq*** command (see Table 46) to notify the end-device of the maximum allowed dwell time, i.e., the maximum continuous transmit time of a packet over the air, as well as the maximum allowed end-device effective isotropic radiated power (EIRP).

Table 46 – *TxParamSetup* payload format

Size (octets)	1
***TXParamSetup* payload**	`EIRP_DwellTime`

The structure of the `EIRP_DwellTime` field is described in Table 47.

Table 47 – `MaxDwellTime` field format

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

Bits [`0...3`] of ***TXParamSetupReq*** command are used to encode the `MaxEIRP` value, as per the following table. The EIRP values in Table 48 are chosen in a way that covers a wide range of maximum EIRP limits imposed by the different regional regulations.

Table 48 – Maximum EIRP encoding

Coded value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MaxEIRP (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 end-device's radio TX power. The end-device is not required to transmit at that power but SHALL NOT radiate more than the specified EIRP.

Bits `4` and `5` define the maximum uplink and downlink dwell times, respectively, which are encoded as per Table 49.

Table 49 – Maximum dwell time encoding

Coded value	**Dwell time**
0	No limit
1	400 ms

If this MAC command is implemented (region-specific), the end-device SHALL acknowledge the ***TXParamSetupReq*** command by sending a ***TXParamSetupAns*** command. This ***TXParamSetupAns*** command contains no payload.

NOTE – When applying a ***TxParamSetupReq*** command, the end-device must use whichever is the lowest of either the region limitation for EIRP or the value encoded in `MaxEIRP`. It must also use whichever is the lowest of either region limitation of dwell time or value encoded in `UplinkDwellTime`.

The **TXParamSetupAns** command SHALL be added in the `FOpts` field (if `FPort` is either missing or >`0`) or in the `FRMPayload` field (if `FPort=0`) of all uplinks until a Class A downlink is received by the end-device. This guarantees that, even in the case of an uplink frame loss, the network is always aware of the downlink parameters used by the end-device.

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

9.9 End-device time commands (*DeviceTimeReq*, *DeviceTimeAns*)

End-devices and network servers SHALL implement these commands.

An end-device MAY use the ***DeviceTimeReq*** command to request the current network time from the network server. This allows the end-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.

A network server MAY use the ***DeviceTimeAns*** command to provide the GPS time to the end-device. The time provided is the GPS time at the end of the uplink transmission. The command has a 5-octet payload defined as specified in Table 50.

Table 50 – *DeviceTimeAns* payload format

Size (octets)	4	1
***DeviceTimeAns* payload**	32-bit unsigned integer: seconds since epoch*	8-bit unsigned integer: fractional-second in $\frac{1}{256}$ s increments

(*) The GPS epoch (i.e., 6 January 1980 00:00:00 UTC) is used as origin. The seconds field is the number of seconds elapsed since the origin. This field increases monotonically by 1 every second. To convert this field to UTC time, the leap seconds SHALL be taken into account.
EXAMPLE – 12 February 2016 at 14:24:31 UTC corresponds to 1139322288s since GPS epoch. As of June 2017, the GPS time is 17 s ahead of UTC time.

The time provided by the network SHALL have a worst-case accuracy of ±100 ms. The ***DeviceTimeAns*** command SHALL be sent as a Class A downlink (i.e., over RX1/RX2 of the Class A mode).

10 End-device activation

To participate in a network as defined in this Recommendation, each end-device SHALL be personalized and activated.

Activation of an end-device can be achieved in two ways, either via OTAA when an end-device is deployed or reset, or via ABP in which the two steps of the end-device personalization and activation are performed in one step.

An end-device SHALL implement either OTAA or ABP, and MAY implement both OTAA and ABP.

10.1 Data stored in the end-device after activation

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

10.1.1 End-device address (`DevAddr`)

The `DevAddr` consists of 32 bits and identifies the end-device within the current network. The `DevAddr` is allocated by the network server of the end-device. Its format SHALL be as specified in Table 51.

Table 51 – `DevAddr` fields

Bits	[31...32−N]	[31−N..0]
DevAddr	AddrPrefix	NwkAddr

The variable N is an integer within the [7:25] range.

The protocol specified in this Recommendation supports various network address types with different network address space sizes. The variable-size `AddrPrefix` field SHALL be derived from the network server's unique identifier `NetID` (24-bit number). The `AddrPrefix` field enables the discovery of the network server that had assigned the `DevAddr`. Eight types of `NetID` are defined, which correspond to different values of N, and of `AddrPrefix` bitfield format (see Table 52).

Table 52 – Mapping of *N* and `AddrPrefix` per NetID type

`NetID` type	Value of *N*	`AddrPrefix` format
0	7	7'b0xx_xxxx
1	8	8'b10xx_xxxx
2	12	12'b110x_xxxx_xxxx
3	15	15'b111_0xxx_xxxx_xxxx
4	17	17'b1_1110_xxxx_xxxx_xxxx
5	19	19'b111_110x_xxxx_xxxx_xxxx
6	22	22'b11_1111_0xxx_xxxx_xxxx_xxxx
7	25	25'b1_1111_110x_xxxx_xxxx_xxxx_xxxx

The least significant (32-*N*) bits are the network addresses (`NwkAddr`) of the end-device. They SHALL be arbitrarily assigned by the network server.

The `AddrPrefix` values of Table 53 may be used by private / experimental networks.

Table 53 – `AddrPrefix` values recommended for use by private / experimental networks

Private/experimental network reserved `AddrPrefix`
***N* = 7**
`AddrPrefix = 7'b0000000 or AddrPrefix = 7'b0000001`
`NwkAddr` = 25-bit range freely allocated by a network server

An example of derivation of the `AddrPrefix` field from `NetID`, definition of the different `NetID` address classes and addresses allocation guidelines may be found in [b-LoRaWAN TS002].

10.1.2 Network session key (`NwkSKey`)

`NwkSKey` is a network session key specific to the end-device. It is used by both the network server and the end-device to calculate and verify the MIC of all data frames to ensure data integrity. It is further used to encrypt and decrypt the payload field of MAC-only data frames, where `FPort=0`.

`NwkSKey` SHOULD be stored such that extraction and re-use by malicious actors is prevented.

10.1.3 Application session key (`AppSKey`)

`AppSKey` is an application session key specific to 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 frames. Application payloads SHALL be encrypted end-to-end between the end-device and the application server, but they are integrity-protected only over-the-air and not end-to-end. This means that a network server may be able to alter the encrypted content of the data frames in transit (yet without being able to read the plain content). Network servers are considered to be trusted, but it IS RECOMMENDED that applications wishing to implement end-to-end confidentiality and integrity protection use additional end-to-end security solutions, which are beyond the scope of this specification.

`AppSKey` SHOULD be stored such that extraction and re-use by malicious actors is prevented.

10.2 Over-the-air activation

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

An end-device SHALL be personalized with the following information before it starts the join procedure: a globally unique end-device identifier (`DevEUI`), the join server identifier (`JoinEUI`, see clause 10.2.2) and an AES-128 key (`AppKey`).

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 to that end-device is derived to encrypt and verify transmissions at the network level. This facilitates roaming of end-devices between networks of different providers. Furthermore, using both a network session key and an application session key allows federated network servers in which application data cannot be read by the network provider.

10.2.1 End-device identifier (`DevEUI`)

`DevEUI` is a global end-device ID in the IEEE EUI-64 address space that uniquely identifies the end-device across roaming networks.

All end-devices SHALL have an assigned `DevEUI`, regardless of which activation procedure is used (i.e., ABP or OTAA).

For OTAA end-devices, `DevEUI` SHALL be stored in the end-device before the join procedure is executed. For ABP end-devices, `DevEUI` SHOULD be stored in the end-device itself.

NOTE – It is a recommended practice that `DevEUI` should also be available on an end-device label for the purpose of an end-device administration.

10.2.2 Join-Server identifier (`JoinEUI`)

`JoinEUI` is a global application ID in the IEEE EUI-64 address space that uniquely identifies the Join-Server that is able to assist in the processing of the Join Procedure and the derivation of session keys.

For OTAA end-devices, `JoinEUI` SHALL be stored in the end-device before the Join Procedure is executed. `JoinEUI` is not required for ABP-only end-devices.

10.2.3 Application key (`AppKey`)

The `AppKey` is an AES-128 root key specific to the end-device.

NOTE – As all end-devices end up with unrelated application keys specific to each end-device, extracting the `AppKey` from an end-device compromises only that one end-device.

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 to that end-device to encrypt and verify network communication and application data.

An `AppKey` SHALL be stored on an end-device intending to use the OTAA procedure.

An `Appkey` is not required for ABP-only end-devices.

10.2.4 Join procedure

From an end-device's point of view, the join procedure consists of two MAC frames exchanged with the server, namely a Join-Request and a Join-Accept.

10.2.5 Join-Request frame

The join procedure is always initiated by the end-device sending a Join-Request frame (see Table 54).

Table 54 – Join-Request payload format

Size (octets)	8	8	2
Join-Request payload	`JoinEUI`	`DevEUI`	`DevNonce`

The Join-Request frame contains the JoinEUI and DevEUI of the end-device followed by a nonce of 2 octets (DevNonce).

DevNonce is a counter starting at 0 when the end-device is initially powered up and incremented with every Join-Request. A DevNonce value SHALL never be reused for a given JoinEUI value. If the end-device can be power-cycled, then DevNonce SHALL be persistent (e.g., stored in a non-volatile memory). Resetting DevNonce without changing JoinEUI will cause the join server to discard the Join-Requests of the end-device. For each end-device, the Join Server keeps track of the last DevNonce value used by the end-device and ignores the join-requests if DevNonce is not incremented.

The MIC value (see clause 8 for MAC frame description) for a Join-Request frame SHALL be calculated as follows (see [IETF RFC 4493]):

```
CMAC = aes128_cmac(AppKey, MHDR | JoinEUI | DevEUI | DevNonce)
MIC = CMAC[0..3]
```

The Join-Request frame is not encrypted.

The Join-Request frame MAY be transmitted using any data rate and following a random frequency-hopping sequence across the specified join channels. Join-Request transmission physical layer parameters may be different than data traffic. If the Join-Request transmission physical layer parameters are not specifically defined, the end-devices SHALL transmit the join-requests across all the available channels at the lowest data rate for the region. Additionally, end-devices SHOULD transmit the join-requests across all the available channels at all the required data rates for that region. The intervals between transmissions of the join-requests SHALL respect the conditions described in clause 11. For each transmission of a join-request, the end-device SHALL increment the DevNonce value.

10.2.6 Join-Accept frame

The network SHALL respond to a Join-Request frame with a Join-Accept frame if the end-device is permitted to join the network. The Join-Accept frame 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 ones used for the RX1 and RX2 receive windows described in clause 7.3 and SHALL use a different timing than RX1 and RX2.

A Join-Accept response SHALL NOT be given to the end-device if the Join-Request is not accepted.

The Join-Accept frame contains a Join-Server nonce (JoinNonce) of 3 octets, a network identifier (NetID), an end-device address (DevAddr), downlink configuration settings (DLSettings), a delay between TX and RX (RXDelay) and an OPTIONAL list of network parameters (CFList) for the network the end-device is joining. The OPTIONAL CFList is region-specific and is defined relative to the channel plan. An example of region specific use of CFList may be found in [b-LoRaWAN RP002].

Table 55 – Join-Accept payload format

Size (octets)	3	3	4	1	1	(16) optional
Join-accept payload	JoinNonce	NetID	DevAddr	DLSettings	RXDelay	CFList

JoinNonce is a non-repeating value provided by the join server and used by the end-device to derive the two session keys NwkSKey and AppSKey, which SHALL be calculated as follows:

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

where the pad_{16} function appends all-zero octets so that the length of the data is a multiple of 16.

The MIC value for a Join-Accept frame SHALL be calculated as follows (see [IETF RFC 4493]):

> CMAC = aes128_cmac(AppKey, MHDR | JoinNonce | NetID | DevAddr | DLSettings | RXDelay | CFList)
> MIC = CMAC[0..3]

The Join-Accept frame itself SHALL be encrypted with the AppKey as follows:

> aes128_decrypt(AppKey, JoinNonce | NetID | DevAddr | DLSettings | RXDelay | CFList | MIC)

NOTE 1 – [b-LoRaWAN TR001] proposes additional behaviour for the JoinNonce value in the join server to prevent synchronization issues related to the version 1.0.x of the join procedure. Some of the remedies include additional behaviour both at the end-device and the join server, which are expected to be configured synchronously.

NOTE 2 – An AES decrypt operation in electronic code book (ECB) mode encrypts the Join-Accept frame so that the end-device can use an AES encrypt operation to decrypt the frame. This way, an end-device has to implement only an AES encrypt but not an AES decrypt.

NOTE 3 – Establishing these two session keys allows for a federated network infrastructure in which the network operators are not able to eavesdrop on the application data. 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 DLSettings field SHALL contain the downlink configuration as specified in Table 56.

Table 56 – DLSettings field format

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 receive window (RX1). The default offset is 0. The offset accommodates the maximum power density constraints for gateways in some regions and balances the uplink and downlink radio link margins.

The RX2DataRate field sets the downlink data rate that serves to communicate with the end-device on the second receive window (RX2).

The RXDelay field sets the downlink RX1 delay and follows the same convention as the Del field in the ***RXTimingSetupReq*** command.

Default RX data rates, default RX receive delays and the actual relationship between the uplink and downlink data rate are region-specific. An example of such parameters may be found in [b-LoRaWAN RP002].

The CFList parameters, when present, SHALL be used in combination with implicit parameters to emulate the receipt of existing MAC commands, such as ***NewChannelReq*** or ***LinkADRReq***. The end-device behaviour when processing CFList SHALL be identical to what would result from processing those MAC commands if they were received in a single downlink frame, after the processing of the non-optional join-accept parameters, with the exception that CFList processing does not generate a MAC answer. The standard behaviour for processing MAC commands is defined in a subsequent clause in this Recommendation.

If the Join-Accept frame is received following the transmission of a Join-Request, the end-device SHALL revert to its default channel and RF parameters definitions. All the MAC layer parameters (except `RXDelay`, `RX2DataRate`, and `RX1DROffset` that are transported by the Join-Accept frame) SHALL be reset to their default values. If the `CFlist` is present, it is then applied.

It IS RECOMMENDED that the first uplink that follows the reception of the Join-Accept frame uses the data rate of the successful join-request, or a lower data rate. The end-device SHALL then apply the adaptive date-rate control as defined in clause 8.3.1.1.

10.2.7 Join procedure completion for Class C

The end-device that expects to receive Class C downlink frames SHALL send a confirmed uplink frame or a frame that requires an acknowledgment as soon as possible after receiving a valid Join-Accept frame. The end-device SHALL continue to send such frames until it receives the first downlink from the network (while respecting duty cycles, if applicable, and retransmission timers).

The network server SHALL NOT transmit a downlink before it has received a first uplink frame.

10.3 Activation by personalization

Activation by personalization ties an end-device directly to a specific network, thus bypassing the join procedure.

Activating an end-device by personalization means that the `DevAddr` and the two session keys `NwkSKey` and `AppSKey` are stored directly in the end-device instead of being derived from `DevEUI`, `JoinEUI` and the `AppKey`. The end-device is equipped with the required information for participating, as soon as it is started, in a specific network as defined in this Recommendation.

Each end-device SHALL have a unique set of `NwkSKey` and `AppSKey` values. Compromising the keys of one end-device SHALL NOT compromise the security of the communications of other end-devices. The process to build those keys SHALL be such that the keys cannot be derived in any way from publicly available information such as the end-device address or `DevEUI`.

Upon first boot and following a reset, personalized end-devices SHALL have all the available channels for that region enabled and SHOULD use all the required data rates for that region. Configurations of the end-device by the network that controls the downlink connectivity (controlled by ***RXParamsSetupReq***, ***DlChannelReq***, ***RXTimingSetupReq*** and ***TXParamSetupReq***) SHALL be persisted by the end-device, even after a reset.

Frame counter values SHALL be used only once in all invocations of a same key with the counter with cipher block chaining message (CCM) mode of operation [IEEE 802.15.4]. Therefore, re-initialization of an ABP end-device frame counters is forbidden. ABP end-devices SHALL store the frame counters persistently (e.g., in non-volatile memory).

NOTE – ABP end-devices use the same session keys throughout their lifetime (i.e., no rekeying is possible). Therefore, it is recommended that OTAA end-devices be used for higher security applications.

11 Retransmissions backoff

Uplink frames that:

- require an acknowledgment or answer from the network or an application server and are retransmitted by the end-device if the acknowledgment or answer is not received, and
- can be triggered by an external event causing synchronization across a large (>100) number of end-devices (power outage, radio jamming, network outage, earthquake, etc.)

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

NOTE – A typical example of such an uplink frame is a join-request if the implementation of a group of end-devices decides to reset the MAC layer in the case of a network outage. The entire group of end-devices will start broadcasting the join-request uplinks and will stop only upon receiving a join-accept 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 end-device (for example using a pseudo-random generator seeded with the end-device's address). The transmission duty-cycle of such a frame SHALL respect local regulations and the following limits, whichever is more constraining (see Table 57).

Table 57 – Transmit duty-cycle limitations

Aggregated during the first hour following power-up or reset	$T_0 < t < T_{0+1}$	Transmit time < 36 s per hour	1% duty cycle
Aggregated during the next 10 hours	$T_{0+1} < t < T_{0+11}$	Transmit time < 36 s per 10 h	0.1% duty cycle
After the first 11 hours, aggregated over 24 h, where *N* refers to days starting at 0	$T_{0+11} + N \times (24\ \text{hours/day}) < t < T_0 + 35 + N \times (24\ \text{hours/day})$, $N \geq 0$	Transmit time < 8.7 s per 24 h	0.01% duty cycle

Class B

Beacon

12 Introduction to Class B

This section describes the 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 network-initiated downlink frames. Class B-capable end-devices SHALL NOT enable Class B and Class C operation concurrently.

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

One of the limitations of Class A is the 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 send a beacon on a regular basis to synchronize all the end-devices in the network so that the end-device can open a short additional reception window (called a 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 switch from Class A to Class B has to be controlled from the network side, the customer application shall 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 by this Recommendation).

12.1 Principle of synchronous network-initiated Class B downlinks

For a network to support end-devices of Class B, the network SHALL broadcast a beacon that provides a timing reference to end-devices. Based on this timing reference, the Class B-enabled end-devices SHALL periodically open receive windows, hereafter called ping slots, which can be used by the network 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 SHALL send an uplink to the network server so that the server can update the downlink routing path database.

When enabling Class B mode, an end-device SHALL use the defined values for the following parameters:

- default ping-slot periodicity,
- default ping-slot data rate,
- default ping-slot channel.

These parameters SHALL have default values and MAY be updated via Class B MAC commands (see clause 16). A region specific example of default values may be found in [b-LoRaWAN RP002].

All end-devices start and join the network as Class A end-devices with Class B disabled. The end-device application can then decide to enable Class B. Class B-capable end-devices still implement all the functionalities of Class A end-devices. In particular, Class B-enabled end-devices SHALL respect the Class A RX1 and RX2 receive window definition following every uplink (see clause 7.3).

Class B is enabled by the following process:

- The end-device application requests the protocol layer to enable Class B mode. The protocol layer in the end-device searches for a beacon. To accelerate beacon discovery, the protocol layer MAY use the ***DeviceTimeReq*** MAC command.
- Once the end-device has found a beacon, it MAY enable Class B mode.
- Once Class B is enabled, the MAC layer SHALL set to 1 the `ClassB` bit of the `FCtrl` field of every uplink frame transmitted to remain Class B-enabled. This bit signals to the server that the end-device has enabled Class B.
- The MAC layer SHOULD autonomously schedule a reception slot for each beacon and each ping slot. The end-device SHALL take 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 SHALL be processed similarly to a downlink as described in the 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 protocol layer SHALL appropriately set the `ClassB` bit to 1 in the frame's `FCtrl` field. This can be done more efficiently if the end-device detects that it is moving by analyzing the beacon content. In that case, to avoid systematic uplink collisions, the end-device SHALL apply a random delay (as defined in clause 17.4) between having received the beacon and transmitting the uplink.
- During any Class A downlink, the network server MAY change the end-device's ping-slot downlink frequency or data rate by sending a ***PingSlotChannelReq*** MAC command and receiving the corresponding ***PingSlotChannelAns***.
- The end-device MAY change the periodicity of its ping slots at any time. To do so, it SHALL temporarily disable Class B operation (unset `ClassB` bit in its uplink frames) and send a ***PingSlotInfoReq*** to the network server. Once this command is acknowledged, the end-device MAY re-enable Class B operation with the new ping-slot periodicity.
- If no beacon has been received for a given period (as defined in clause 16.2), synchronization with the network is lost. The end-device protocol layer SHALL stop setting the `ClassB` bit in all uplinks, which informs the network server that the end-device has disabled Class B mode.

Figure 4 illustrates the concept of beacon reception slots and ping slots. In this example, given a beacon period of 128 s, the end-device also opens a ping-slot reception window every 32 s. Most of the time, this ping slot is not used by the network 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 MIC is valid before forwarding it to the application layer. The end-device response shown in this example is optional, depending on the downlink, and, if present, this is a Class A uplink.

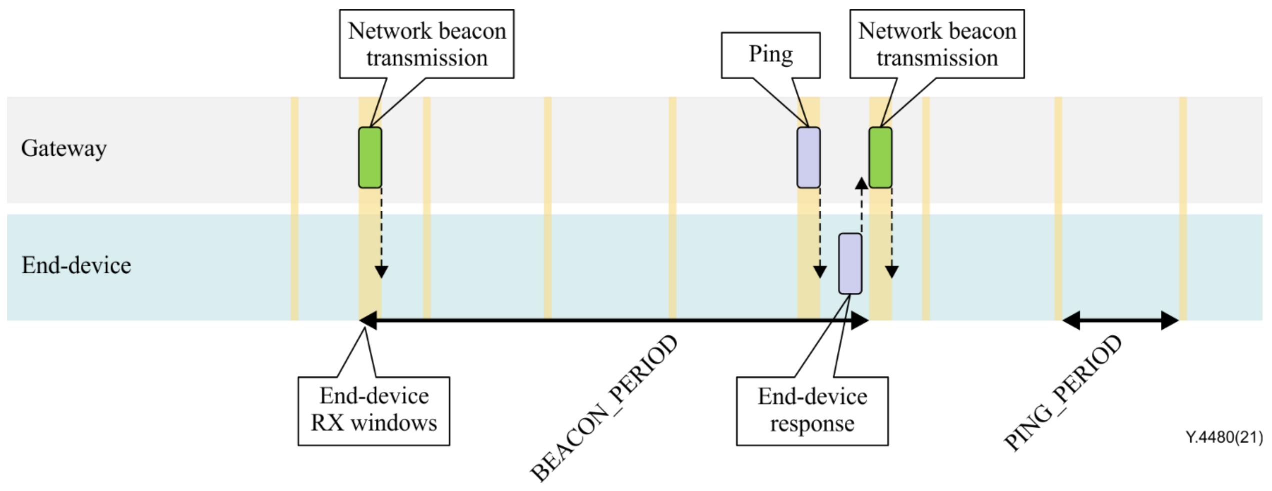

Figure 4 – Example of beacon reception slot and ping slots

13 Class B frame formats

13.1 Uplink frames

The uplink frames in Class B mode are the same as the Class A uplinks. The `ClassB` bit SHALL be set to `1` in an uplink to signal the network server that the end-device is Class B-enabled and is ready to receive scheduled downlink pings.

13.2 Downlink frames

The `FPending` bit for Class B downlink frames signals that, in the event of a ping-slot collision between multiple Class B ping slots, the ping-slot sequence whose `FPending` bit was set first will take priority. If the `FPending` bit has been set for multiple Class B ping-slot sequences, they will take priority over ping-slot sequences for which `FPending` has not been set. Further prioritization can be determined based on whether the downlink ping slot is used for multicast or unicast frames.

Table 58 – `FPending` Class B prioritization

Priority	Type of Class B downlink
Highest	Multicast with `FPending` previously set
	Unicast with `FPending` previously set
	Multicast with `FPending` not previously set
Lowest	Unicast with `FPending` not previously set

If there are multiple colliding ping-slot sequences at the same priority level as shown in Table 58, the highest unicast or multicast `DevAddr` SHALL take priority.

13.3 Downlink ping frames

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

Frames can be unicast or multicast. Unicast frames are sent to a single end-device, whereas multicast frames are sent to multiple end-devices. All end-devices of a multicast group SHALL share the same multicast address and associated encryption keys. A Class B-capable end-device SHALL support at least one multicast group and an end-device SHALL NOT transmit using a multicast address.

More precisely, inside a given end-device, a multicast group is defined by the following parameters called the multicast group context:

1) A 4-octet network address of the multicast group, common to all end-devices of the group;
2) The multicast group sessions keys (different for every multicast group, but all end-devices of a given multicast group have the same session keys);
3) The multicast group downlink frame counter.

The Class B specification does not specify means to set up such a multicast group remotely or to distribute the required multicast key material securely. This can be performed either during the end-device personalization or through the application layer.

EXAMPLE – [b-LoRaWAN TS005] describes a possible application layer mechanism for over-the-air multicast key distribution.

13.3.1 Unicast downlink ping frame format

The MAC payload of a unicast downlink ping uses the format defined in the Class A specification. The same frame counter is incremented, whether the downlink uses a Class B ping slot or a Class A downlink slot. A downlink ping is processed by the end-device in the same way as a Class A downlink, except for MAC commands and confirmed frames.

A downlink ping SHALL NOT transport any MAC command. If an end-device receives a downlink ping containing a MAC command, either in the `FOpts` field (if `FPort` is either missing or >`0`) or in the `FRMPayload` field (if `FPort=0`), it SHALL silently discard the entire frame.

In case a confirmed downlink ping frame is received, the end-device SHALL NOT answer later than a time period equal to:

CLASS_B_RESP_TIMEOUT * `NbTrans` + RECEIVE_DELAY2 * (`NbTrans – 1`)

when the end-device sets its uplink ADR bit, and CLASS_B_RESP_TIMEOUT when the end-device unsets its uplink ADR bit. The default value of CLASS_B_RESP_TIMEOUT is 8 s. It can be modified to fit regional regulatory and physical layer constraints; it SHALL not be smaller than RETRANSMIT_TIMEOUT plus the maximum time on air of the uplink frame.

NOTE 1 – The purpose of this timeout is for the network server to know how long to wait for a response from the end-device, before it transmits another confirmed downlink. This ensures that the network server can process responses without any ambiguity: There may be only a single confirmed downlink ping pending an acknowledgement.

The uplink frame sent in response MAY be sent up to `NbTrans` times, but no retransmission SHALL occur after the timeout period. It IS RECOMMENDED that the end-device transmits each copy of the uplink frame at a random time within CLASS_B_RESP_TIMEOUT.

As this adds a timing requirement compared to responding to Class A downlinks, the end-device may not send an uplink frame within the timeout, for instance if it has a duty cycle limitation. When such an uplink frame is not sent, the end-device SHALL act as if the uplink frame with the ACK bit set was sent.

After sending a confirmed downlink frame sent over ping slot, the network server SHALL NOT send any other confirmed downlink to the end-device until this timeout expires or an uplink frame is received from that end-device.

After sending a Class A confirmed downlink, a network server SHALL NOT send any other confirmed downlink to the end-device until an uplink frame is received from that end-device.

NOTE 2 – Unconfirmed downlink frames sent over ping slots may be sent without a minimum delay between them.

13.3.2 Multicast downlink ping frame format

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

- They SHALL NOT carry MAC commands in the `FOpts` field nor in the payload on port 0 because a multicast downlink does not have the same authentication robustness as a unicast frame. The end-device SHALL discard any multicast frame carrying MAC commands;
- The `ACK` bit SHALL be `0` and the `FType` field SHALL have the value `Unconfirmed Data Down` (see Table 4). The end-device SHALL discard the multicast frame otherwise.

14 Class B beacon acquisition and tracking

Before enabling Class B operation, the end-device SHOULD first synchronize with the network beacons to align its internal timing reference with the network.

Once Class B is enabled, the end-device SHOULD 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-enabled end-device may be temporarily unable to receive beacons (out of range from the network gateways, presence of interference, etc.). In this event, the end-device SHOULD gradually widen its beacon and ping-slot reception windows to accommodate a possible drift of its internal clock.

NOTE – For example, an end-device whose internal clock is defined with a precision of ±10 ppm may drift by ±1.3 ms every beacon period.

14.1 Minimal beaconless operation time

In the event of beacon loss, an end-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 beaconless operation. It relies on the end-device's own clock to keep time.

During beaconless operation (see Figure 5), Class B unicast, multicast and beacon reception slots SHALL be progressively expanded to accommodate the end-device's possible clock drift.

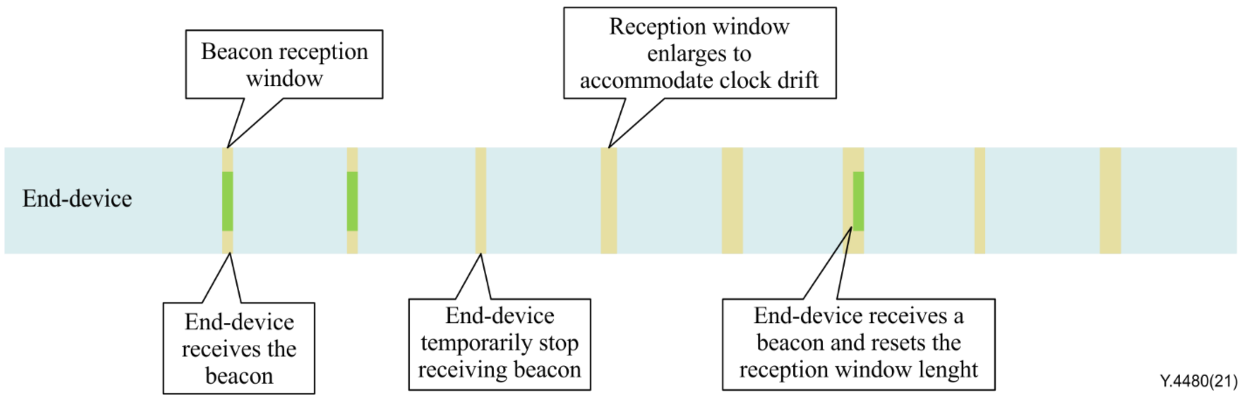
Beacon reception window
Reception window enlarges to accommodate clock drift
End-device
End-device receives the beacon
End-device temporarily stop receiving beacon
End-device receives a beacon and resets the reception window lenght
Y.4480(21)

Figure 5 – Beaconless temporary operation

14.2 Extension of beaconless operation upon receipt

During the 120-minute time interval described above, the receipt of any beacon directed to the end-device SHOULD extend Class B beaconless operation by another 120 minutes allowing the end-device to correct any timing drift and reset the duration of the receive slots.

NOTE – An end-device can also use Class B ping-slot downlinks to resynchronize its internal clock.

14.3 Minimizing timing drift

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

15 Class B downlink slot timing

15.1 Definitions

To operate successfully in Class B, end-devices SHALL open reception slots at precise instants relative to the infrastructure beacon. This clause defines the required timing.

The interval between the start of two successive beacons is called the beacon period (see Figure 6). The beacon frame transmission is aligned with the beginning of the BEACON_RESERVED interval (see Table 59). Each beacon is preceded by a BEACON_GUARD time interval, where no ping slot can be placed. The length of the BEACON_GUARD time interval corresponds to the time on air of the longest allowed frame. This is to ensure that a Class B downlink initiated during a ping slot just before the BEACON_GUARD time interval will always have time to finish without colliding with the beacon transmission. The usable time interval for a ping slot therefore spans from the end of the BEACON_RESERVED time interval to the beginning of the next BEACON_GUARD time interval.

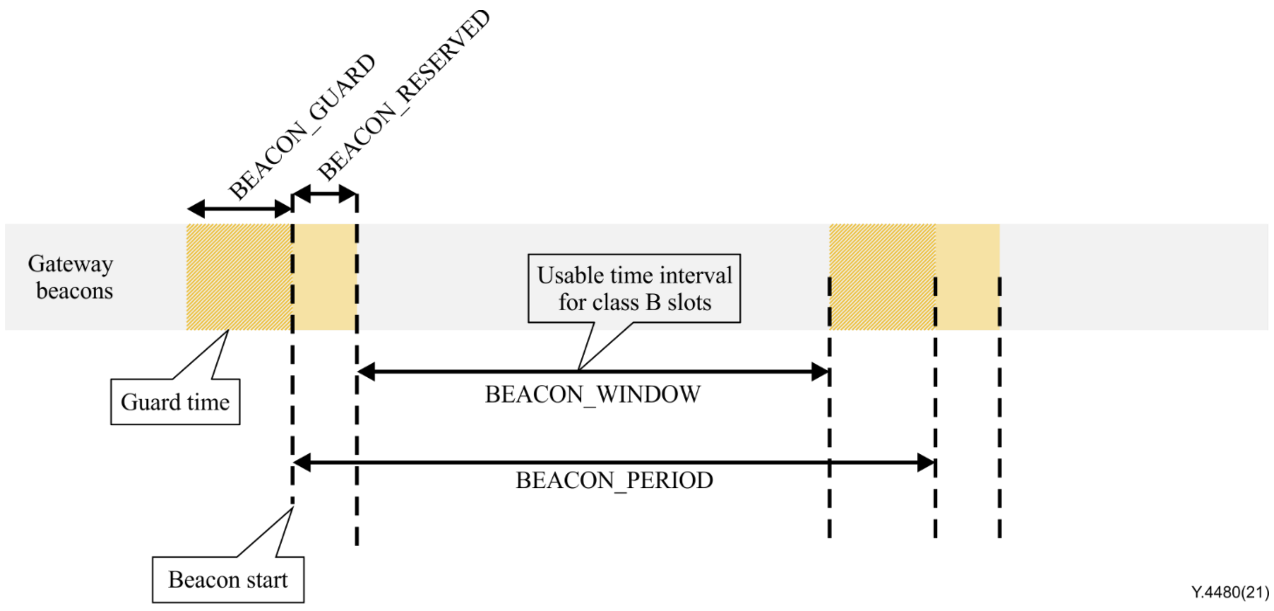

Figure 6 – Beacon timing

Table 59 – Beacon timing

BEACON_PERIOD	128 s
BEACON_RESERVED	2.120 s
BEACON_GUARD	3.000 s
BEACON_WINDOW	122.880 s

The beacon frame time on air is much shorter than the BEACON_RESERVED time interval so as to allow the network management broadcast frames to be appended in the future.

The BEACON_WINDOW time interval is divided into 212 = 4 096 ping slots of 30 ms each, numbered from 0 to 4 095.

An end-device using the slot number N SHALL turn its receiver on Ton s after the start of the beacon, where:

$$T_{on} = \text{BEACON\_RESERVED} + N \times 30 \text{ ms.}$$

N is called the slot index.

The latest ping slot starts at BEACON_RESERVED + 4 095 × 30 ms = 124 970 s after the beacon starts or 3 030 ms before the beginning of the next beacon.

15.2 Slot randomization

To avoid systematic collisions or overhearing problems, the slot index is randomized and changed at every beacon period.

The parameters of Table 60 are used (Periodicity is defined in clause 16.1).

Table 60 – Class B slot randomization algorithm parameters

`DevAddr`	Device 32-bit network unicast or multicast address
`PingNb`	Number of ping slots per beacon period. This is a power of 2 integer: $\texttt{PingNb} = 2^{7-\text{Periodicity}}$
`PingPeriod`	Period of the end-device receiver wake-up expressed in number of slots: $\texttt{PingPeriod} = 2^{5+\text{Periodicity}}$
`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`
`SlotLen`	Length of a unit ping slot = 30 ms

At each beacon period, the end-device and the server SHALL compute a new pseudo-random offset to align the reception slots. An AES encryption with a fixed key of 16 all-zero octets SHALL be used to randomize:

Key = 16 × `0x00`
Rand = aes128_encrypt(Key, `BeaconTime` | `DevAddr` | pad16)
`PingOffset` = (Rand[0] + Rand[1] × 256) modulo `PingPeriod`

The slots used for this beacon period SHALL be:

`PingOffset` + *N* × `PingPeriod` with *N* = [`0`:`PingNb-1`]

The end-device therefore opens receive slots starting at the times given in Table 61.

Table 61 – Receive-slot starting times

First slot	BEACON_RESERVED + `PingOffset` × `SlotLen`
Slot 2	BEACON_RESERVED + (`PingOffset` + `PingPeriod`) × `SlotLen`
Slot 3	BEACON_RESERVED + (`PingOffset` + 2 × `PingPeriod`) × `SlotLen`
…	…

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

If a Class A RX1 or RX2 receive slot collides with a Class B multicast or unicast slot, the end-device SHALL listen to the Class A RX slot that has priority.

NOTE – As defined in the Class A section, the end-device does not open the RX2 receive window if a valid unicast downlink addressed to the end-device is received in the RX1 receive window.

The randomization scheme prevents a systematic collision between unicast and multicast slots. If a collision occurs during a beacon period, one is unlikely to occur again during the next beacon period.

16 Class B MAC commands

All commands described in the Class A specification SHALL be implemented in Class B-capable end-devices. End-devices implementing the Class B specification SHALL further implement the MAC commands listed in Table 62 (see also Table 14).

Table 62 – Class B MAC command table

CID	**Command**	**Transmitted by**		**Brief description**
		End-device	Network server	
`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 Note – This command has a different acknowledgment mechanism as described in the command definition.
`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 a beacon broadcast
`0x13`	***BeaconFreqAns***	x		Used by the end-device to acknowledge a ***BeaconFreqReq*** command

MAC commands which require an answer from the network expire after the "Class A receive windows" have elapsed.

16.1 PingSlotInfoReq

An end-device MAY use the ***PingSlotInfoReq*** command to inform the server of its unicast ping-slot periodicity (see Tables 63 and 64). This command SHALL be used only to inform the server of the periodicity of a unicast ping slot. A multicast slot is entirely defined by the application and SHALL NOT use this command.

Table 63 – *PingSlotInfoReq* payload format

Size (octets)	1
***PingSlotInfoReq* payload**	`PingSlotParam`

Table 64 – *PingSlotParam* field format

`Bits`	`7:3`	`[2:0]`
`PingSlotParam`	`RFU`	`Periodicity`

The `Periodicity` subfield is an unsigned 3-bit integer encoding the ping-slot period currently used by the end-device using the following equations:

`pingNb` = $2^{7-\texttt{Periodicity}}$
`pingPeriod` = $2^{5+\texttt{Periodicity}}$ slots

The actual ping-slot periodicity will be $0.96 \times 2^{\texttt{Periodicity}}$ seconds.

- `Periodicity` = `0` means that the end-device opens a ping slot approximately every 1 second during the BEACON_WINDOW interval.
- `Periodicity` = `7` means that the end-device opens a ping slot approximately every 128 seconds, which is the maximum ping-slot period supported by the Class B specification.

To change its ping-slot periodicity, an end-device SHALL first revert to Class A. Next it SHALL send the new periodicity through a ***PingSlotInfoReq*** command. Then it SHALL receive an acknowledgement from the server through a ***PingSlotInfoAns***. Only then MAY the end-device switch back to Class B with the new periodicity.

This command MAY be concatenated with any other MAC command in the `FOpts` field of FHDR, as described in the Class A specification frame format.

Upon receiving this ***PingSlotInfoReq*** command, the network server SHALL answer with a ***PingSlotInfoAns*** frame. The MAC payload of this frame is empty.

16.2 BeaconFreqReq

This command is sent by the server to the end-device to modify the frequency on which this end-device expects the beacon (see Table 65).

Table 65 – *BeaconFreqReq* payload format

Octets	3
***BeaconFreqReq* payload**	`Frequency`

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

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

A value of `0` instructs the end-device that it SHALL use the default beacon frequency plan. Where applicable, the end-device SHALL resume a frequency hopping beacon search.

Upon receiving this command, the end-device SHALL answer with a ***BeaconFreqAns*** frame. The MAC payload of this frame contains the information given in Table 66.

Table 66 – *BeaconFreqAns* payload format

Size (octets)	1
***BeaconFreqAns* payload**	`Status`

The bits of the `Status` octet have the meaning given in Tables 67 and 68.

Table 67 – Status field format

Bits	7:1	0
Status	RFU	Beacon frequency ok

Table 68 – Meaning of beacon frequency bits

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

16.3 PingSlotChannelReq

The server MAY send this command to the end-device to modify the frequency and/or the data rate at which the end-device expects the downlink pings.

Once the network server has sent the ***PingSlotChannelReq*** command, it SHALL NOT attempt to use a Class B ping slot until it receives the ***PingSlotChannelAns***.

Note 1 – In order to take advantage of the network-initiated downlink capabilities provided by Class B, the network needs relatively recent information of how to best contact the end-device. This information is provided to the network through any and all the uplinks received from the end-device. For this reason, it is important for Class B enabled end-devices to send regular uplinks which implicitly inform the network of the best way to contact it as well as providing the network a chance to send new MAC commands which may be required for proper end-device operation.

Table 69 – *PingSlotChannelReq* payload format

Octets	3	1
***PingSlotChannelReq* payload**	`Frequency`	`DR`

The `Frequency` coding is identical to the ***NewChannelReq*** MAC command defined for Class A. A value of `0` instructs the end-device that it SHALL use the default frequency plan.

The data rate (`DR`) octet contains the fields specified in Table 70.

Table 70 – `DR` field format

`Bits`	`7:4`	`3:0`
`DR`	`RFU`	`DataRate`

The `DataRate` subfield is the index of the data rate used for the ping-slot downlinks. The relationship between the index and the physical data rate SHALL be defined for each physical layer and regulatory region of operation. A region-specific example may be found in [b-LoRaWAN RP002].

Upon receiving this command, the end-device SHALL answer with a frame containing the ***PingSlotChannelAns*** command. The ***PingSlotChannelAns*** command SHALL be added in the `FOpts` field (if `FPort` is either missing or `>0`) or in the `FRMPayload` field (if `FPort=0`) of all uplinks until a Class A downlink is received by the end-device.

NOTE 2 – To limit the unavailability of ping slots, the end-device will answer as soon as possible to this MAC command.

Table 71 – *PingSlotChannelAns* payload format

Size (octets)	1
***PingSlotChannelAns* payload**	`Status`

The `Status` bits have the meaning specified in Tables 72 and 73.

Table 72 – `Status` field format

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

Table 73 – `Status` field bits signification

	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 end-device cannot receive on this frequency	This frequency can be used by the end-device

If either bit equals `0`, the command did not succeed, and the ping-slot parameters SHALL NOT be modified.

16.4 *BeaconTimingReq* and *BeaconTimingAns*

These MAC commands have been deprecated since [b-LoRaWAN TS001-1.0.3]. End-devices SHALL use ***DeviceTimeReq*** and ***DeviceTimeAns*** commands as substitutes.

17 Class B beaconing

In addition to relaying frames between end-devices and network servers, gateways MAY participate in providing a time-synchronization mechanism by sending beacons at regular fixed intervals. The beacon content, modulation parameters and frequencies SHALL be defined for each physical layer and regulatory region. An example may be found in [b-LoRaWAN RP002].

17.1 Beacon frame format

The beacon payload `BCNPayload` consists of a network common part and an OPTIONAL gateway-specific part (see Table 74).

Table 74 – Beacon frame content

Size (octets)	*N*	1	4	2	7	*M*	2
BCNPayload	RFU	Param	Time	CRC	GwSpecific	RFU	CRC

The length N and M of the `RFU` fields of `BCNPayload` depend on the modulation parameters and SHALL be defined for each physical layer and regulatory region.

The `Param` bits have meaning specified in Table 75, where `Prec` encodes the timing precision of the beacon.

Table 75 – `Param` bits

Bits	7:2	1:0
Param	RFU	Prec

The precision is interpreted as a base-ten exponent of the beacon's transmit time precision $10^{(-6+\texttt{Prec})}$ seconds**,** where the default value of `Prec` is `0`. The `Prec` field can take any value within the range [`0`:`3`]. The `RFU` portion of `Param` SHALL be set to `0` and the end-device SHALL silently ignore this field. Networks that use a beacon precision value other than `0` for `Prec` SHOULD send the beacon using a non-default value of `BeaconFrequency`.

The common part MAY contain an `RFU` field equal to `0`. It also contains a `Param` including a timestamp precision `Prec` and a timestamp `Time` expressed in seconds elapsed 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-bit cyclic redundancy check (CRC). The value of CRC-16 SHALL be computed on the `RFU` +`Param` + `Time` fields as defined in [IEEE 802.15.4]. This cyclic redundancy check (CRC) SHALL use the polynomial $P(x) = x^{16} + x^{12} + x^5 + x^0$. The CRC SHALL be calculated on the octets in the order they are sent over the air.

17.2 Beacon `GwSpecific` field format

The content of the `GwSpecific` field is specified in Table 76.

Table 76 – Beacon `GwSpecific` field format

Size (octets)	1	6
`GwSpecific`	`InfoDesc`	`Info`

The information descriptor `InfoDesc` describes how the information field `Info` SHALL be interpreted (see Table 77).

Table 77 – Beacon `InfoDesc` index mapping

`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

For a single omnidirectional antenna gateway, the value of `InfoDesc` is `0` when broadcasting GPS coordinates. For a site featuring sector antennas, for example, the first antenna broadcasts the beacon with `InfoDesc=0`, the second antenna with `InfoDesc=1`, and so on.

17.2.1 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 (see Table 78).

Table 78 – Beacon `Info` field format, `InfoDesc=0,1,2`

Size (octets)	3	3
`Info`	`Lat`	`Lng`

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

- The north–south latitude SHALL be encoded using a two's complement 24-bit word, where -2^{23} corresponds to 90° south (the South Pole) and 2^{23} corresponds to 90° north (the North Pole);
- The east–west longitude SHALL be encoded using a two's complement 24-bit word, where -2^{23} corresponds to 180° west and 2^{23} corresponds to 180° east.

NOTE – It is not possible to describe 90° north because 2^{23}-1 is the largest number that can be represented in two's complement notation. This position error of approximately 1.2 meter at the North Pole is considered small.

17.2.2 `NetID` + `GatewayID`

For `InfoDesc=3`, the content of the `Info` field encodes the network's `NetID` plus a freely allocated gateway or a cell identifier. The format of the `Info` field is specified in Table 79.

Table 79 – Beacon `Info` field format, `InfoDesc=3`

Size (octets)	3	3
`Info`	`NetID`	`GatewayID`

17.3 Beaconing precise timing

A beacon SHALL be sent every 128 s starting on 6 January 1980, 00:00:00 UTC (start of the GPS epoch) plus $T_{\texttt{BeaconDelay}}$. Therefore, a beacon is sent at $B_T = k \times 128 + T_{\texttt{BeaconDelay}} \pm T_{\text{Accuracy}}$ seconds after the GPS epoch, where k is the smallest integer for which $k \times 128 > T$ and T = number of seconds since 6 January 1980, 00:00:00 UTC (start of the GPS time).

NOTE – T is GPS time and unlike unix time, it increases strictly monotonically and is not influenced by leap seconds.

$T_{\texttt{BeaconDelay}}$ is 1.5 ms. It allows a slight transmit delay (1.5 ms) of the gateways required by their radio system to switch from receive to transmit mode.

The variable T_{Accuracy} denotes the timing precision guaranteed by the gateway. T_{Accuracy} SHALL be provided by the gateway manufacturer with the set of operational conditions required to guarantee it. T_{Accuracy} SHALL be less than or equal to the timing precision indicated by the `Prec` field of the beacon $T_{\text{Accuracy}} \leq 10^{-6+\texttt{Prec}}$ seconds.

All gateways participating in the broadcast of the Class B beacon shall be synchronized. Depending on the timing precision that can be guaranteed by the gateway, two possible modes of the Class B beacon transmission are possible:

- $T_{\text{Accuracy}} \leq 1$ µs: gateways tightly synchronized to GPS time: If the gateway transmissions can be synchronized to the GPS clock with an accuracy of better than 1 µs, the gateway MAY transmit the beacon every 128 s. The `Prec` field of the beacon is set to `0`, indicating a timing accuracy of better than 1 µs for the listening end-devices.
- $T_{\text{Accuracy}} > 1$ µs: gateways loosely synchronized to GPS time: If gateway transmissions can be synchronized to the GPS clock with an accuracy of better than 1 ms, but an accuracy of 1 µs cannot be guaranteed, the transmission of a Class B beacon SHALL be randomized. For each beacon, the gateway SHALL draw a random number P with a uniform distribution between 0 and 1. The gateway transmits the beacon if $P < P_{\text{Beacon}}$. If $P \geq P_{\text{Beacon}}$, the gateway remains silent and does not transmit the beacon. The parameter P_{Beacon} exists on the gateway and MAY be remotely set by the network server. The value of P_{Beacon} SHALL be ≤ 0.5 (50%). Different gateways may use different values of the P_{Beacon} parameter.

  The parameter P may be a pseudo-random number but, in this case, each gateway in the network SHALL use a different seed, resulting in a unique series of P values.

If the Class B beacons of two or more loosely synchronized gateways reach the antenna of an end-device with equivalent power, the end-device may not be able to demodulate the beacon. The timing difference between the colliding beacons may be too great to be compensated correctly by the end-device's demodulator. To avoid systematic beacon collision at the antenna of a stationary end-device, randomization must take place. Each loosely synchronized gateway randomly transmits the beacon no more than half of the time. Therefore, although beacon collisions do happen at the end-device's antenna, they are not systematic, and the end-device can still demodulate the Class B beacon with sufficient probability to operate in Class B mode. The P_{Beacon} parameter should be optimized by the network infrastructure based on the average number of gateways that local end-devices can receive. The denser the gateway population becomes, the higher the probability of beacon collision, so lower the parameter should be. The optimal parameter value depends on too many network-driven factors and is beyond the scope of this specification.

All end-device ping slots SHALL use the start of the receipt of the preamble of the Class B beacon as a timing reference. Therefore, the network server SHALL take $T_{\texttt{BeaconDelay}}$ into account when scheduling the Class B downlinks.

17.4 Network downlink route update requirements

When the network attempts to communicate with an end-device using a Class B downlink slot, it SHOULD transmit the downlink from the gateway closest to the end-device when the most recent uplink was received. Therefore, the network server SHOULD keep track of the approximate position of every Class B end-device.

Whenever a Class B end-device moves and changes cells, it SHALL communicate with the network server in order to update its downlink route. This update is performed simply by sending a confirmed or unconfirmed uplink, possibly without an applicative payload.

The end-device can communicate in accordance with two basic strategies:

- Systematic periodic uplink: This is the simplest method as it does not require demodulation of the gateway-specific field of the beacon. It is applicable only to slowly moving or stationery end-devices. No requirements are imposed on such periodic uplinks.
- Uplink on cell change: The end-device can demodulate the optional gateway-specific field of the beacon. It detects that the ID of the gateway broadcasting the beacon it demodulates has changed and sends an uplink. In that case, the end-device SHALL respect a pseudo-random delay within the range of [0 s:120 s] between the beacon demodulation and the uplink transmission to ensure that the uplinks of multiple Class B end-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 a cell change can result in a Class B downlink being temporarily not operational.

18 Class B unicast and multicast downlink channel frequencies

The Class B downlink channel selection mechanism depends on the way the Class B beacon is broadcast at the physical layer.

18.1 Single-channel beacon transmission

In certain regions, a beacon is transmitted on a single channel. In that case, all unicast and multicast Class B downlinks SHALL use a single frequency channel defined by the ***PingSlotChannelReq*** MAC command. The default frequency SHALL be defined specific to the regulatory region and physical layer.

18.2 Frequency-hopping beacon transmission

This specification supports and offers two options for Class B beacon transmission. In any region, a beacon SHALL either be transmitted following a frequency-hopping pattern or use a fixed frequency.

In regions with a hopping beacon, by default Class B ping slots SHALL use a channel that is a function of the time field of the previous beacon (see clause 17.1) and the value of `DevAddr`.

$$\text{Class B ping} - \text{slot channel} = \left[\text{DevAddr} + \text{floor}\left(\frac{\text{BeaconTime}}{\text{BeaconPeriod}}\right)\right] \text{ modulo NbChannel}$$

where:

- `BeaconTime` is the 32-bit `Time` field of the current beacon period;
- `BeaconPeriod` is the length of the beacon period (defined as 128 s in the specification);
- floor designates rounding to the immediately lower integer value;
- `DevAddr` is the 32-bit network address of the end-device or multicast group;
- `NbChannel` is the number of channels over which the beacon is frequency hopping.

Class B downlinks therefore hop across the `NbChannel` channels (identical to the beacon transmit 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 `Frequency` argument sets the Class B downlink frequency, then all the subsequent ping slots SHOULD be opened on this single frequency independent of the previous beacon frequency.

If the ***PingSlotChannelReq*** command with a zero `Frequency` argument is sent, the end-device SHOULD resume the default frequency plan, that is, Class B ping slots hopping across the `NbChannel` channels.

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

Class C

Continuous listening end-devices

19 Introduction to Class C

Class C mode is used for applications that have sufficient power available such that they do not need to minimize the time the radio receiver is active as they do in Class A or Class B applications.

Class C-capable end-devices SHALL NOT enable Class B and Class C concurrently. Class A downlink, however, are always available after an end-device uplink.

A Class C-enabled end-device listens as often as possible using a combination of channel / DR parameters referred to as RXC (see Figure 7). The end-device SHALL listen on RXC when it is not (a) transmitting or (b) receiving on RX1 or (c) receiving on RX2, according to the Class A definition. To do so, it SHALL open a short window on RXC parameters between the end of the uplink transmission and the beginning of the RX1 reception window. It SHALL open another RXC window between the end of the RX1 window and the beginning of the RX2 window, and it SHALL switch to RXC reception parameters as soon as the RX2 reception window is closed. This final RXC reception window SHALL remain open until the end-device begins to send another packet.

If the end-device is in the process of demodulating a downlink using the RXC parameters when the RX1 or RX2 window should be opened, it SHALL stop the demodulation and switch to the RX1 or RX2 receive window. This applies even when the RXC and RX2 parameters are identical. The purpose of this rule is to achieve a clear separation between the downlinks received during RX1 and RX2 windows (called Class A downlinks) and the downlinks received during a Class C window (called Class C downlinks). The same frame counter is incremented, whether the downlink uses RX1 / RX2 or RXC.

NOTE 1 – In order to take advantage of the network-initiated downlink capabilities provided by Class C, the network needs relatively recent information of how to best contact the end-device. This information is provided to the network through any and all the uplinks received from the end-device. For this reason, it is important for Class C enabled end-devices to send regular uplinks which implicitly inform the network of the best way to contact it as well as providing the network a chance to send new MAC commands which may be required for proper end-device operation.

A Class C downlink SHALL NOT transport any MAC command. If an end-device receives a Class C downlink containing a MAC command, either in the `FOpts` field (if `FPort` is either missing or >`0`) or in the `FRMPayload` field (if `FPort=0`), it SHALL silently discard the entire frame.

In case a Class C confirmed downlink is received, the end-device SHALL NOT answer later than a time period equal to:

- CLASS_C_RESP_TIMEOUT * `NbTrans` + RECEIVE_DELAY2 * (`NbTrans` – 1) when the end-device sets its uplink `ADR` bit,
- CLASS_C_RESP_TIMEOUT
  when the end-device unsets its uplink `ADR` bit.

The default value of CLASS_C_RESP_TIMEOUT is 8 s. It can be modified but SHALL not be smaller than the RETRANSMIT_TIMEOUT plus the maximum time on air of the uplink frame.

NOTE 2 – The purpose of this timeout is for the network server to know how long to wait for a response from the end-device, before it transmits another Class C confirmed downlink. This ensures that the network server can process responses without any ambiguity: there may be only a single confirmed Class C downlink pending an acknowledgement.

The uplink frame sent in response MAY be sent up to `NbTrans` times, but no retransmission SHALL occur after the timeout period. It IS RECOMMENDED that the end-device transmits each copy of the uplink frame at a random time within CLASS_C_RESP_TIMEOUT.

As this adds a timing requirement compared to responding to Class A downlinks, the end-device may not send an uplink frame within the timeout, for instance if it has a duty cycle limitation. When such an uplink frame is not sent, the end-device SHALL act as if the uplink frame with the `ACK` bit set was sent.

After sending a Class C confirmed downlink, a network server SHALL NOT send any other confirmed downlink to the end-device until this timeout expires or an uplink frame is received from that end-device.

After sending a Class A confirmed downlink, a network server SHALL NOT send any other confirmed downlink to the end-device until an uplink frame is received from that end-device.

NOTE 3 – Class C downlink frames of type `Unconfirmed Data` may be sent without a minimum delay between them.

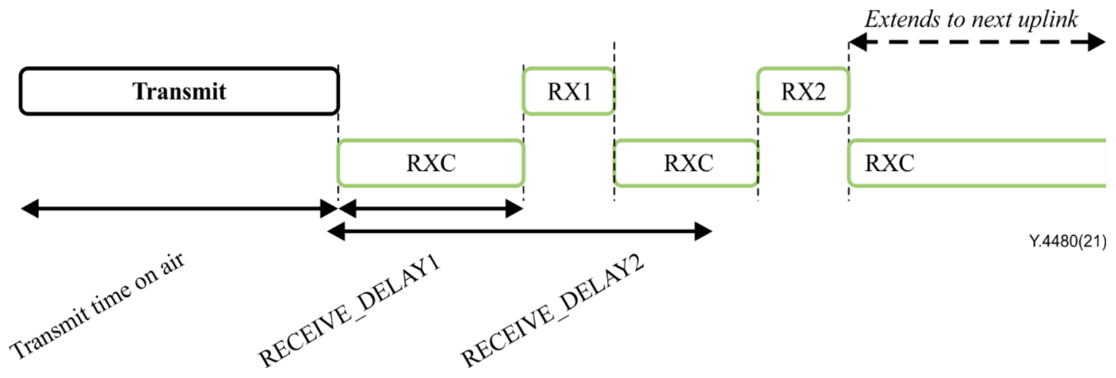

Figure 7 – Class C end-device reception slot timing

NOTE 4 – As defined in clause 7, the end-device does not open RX2 if a downlink is received in RX1. In that case, the end-device opens a continuous RXC receive window at the end of the demodulation of the RX1 downlink that extends until the next uplink.

The Class C mechanism may be used to receive unicast or multicast downlink frames:

- Unicast Class C downlinks are typically used for a command and control signal sent to a specific end-device with very low latency (e.g., single streetlight);
- Multicast Class C downlinks are used to broadcast the same downlink frame(s) to a group of end-devices. Typical applications include firmware updates over the air (FUOTA) and simultaneously controlling a group of end-devices such as streetlights.

The RXC window parameters differ, depending on whether the Class C functionality is used to receive unicast or multicast downlinks:

- Unicast: The RXC parameters are identical to the RX2 parameters, and they use the same channel and data rate. Modifying the RX2 parameters using the appropriate MAC commands also modifies the RXC parameters.
- Multicast: The RXC parameters are provided by the application layer. All end-devices in the multicast group SHALL share the same RXC parameters. If the multicast RXC parameters are different from the end-device's RX2 parameters, then the end-device is not able to listen simultaneously to multicast and unicast downlink. In that case, the decision whether the end-device should use unicast or multicast RXC parameters is application-specific. If the multicast RXC parameters provided by the application layer match the current RX2 parameters of the end-device, then the end-device receives both unicast and multicast traffic during the RXC windows.

19.1 Class C multicast downlinks

Analogously to Class B, Class C end-devices can receive multicast downlink frames. The multicast address and associated network session key SHALL come from the network server, and the application session key SHALL come from the application server.

More precisely, inside a given end-device, a multicast group is defined by the following parameters called the multicast group context:

1) A 4-octet network address of the multicast group, common to all end-devices of the group;
2) A multicast group-specific session key, different for every multicast group; all end-devices of a given multicast group have the same session keys;
3) A multicast group-specific downlink frame counter.

EXAMPLE – [b-LoRaWAN TS005] provides an application layer mechanism to set up a multicast group over the air.

The following limitations apply for Class C multicast downlink frames:

- They SHALL NOT carry MAC commands in the `FOpts` field nor in the payload on port 0 because a multicast downlink does not have the same authentication robustness as a unicast frame. The end-device SHALL discard any multicast frame carrying MAC commands.
- The `ACK` bits SHALL be `0` and the `FType` field SHALL carry the value for `Unconfirmed Data Down`. The end-device SHALL discard the multicast frame otherwise.
- Given that a Class C end-device keeps its receiver active most of the time, the `FPending` bit does not trigger a specific behaviour of the end-device and SHALL NOT be used.

Appendix I

Examples

(This appendix does not form an integral part of this Recommendation.)

The following examples are illustrations of the protocol specification for informational purposes only.

I.1 Uplink timing diagram for unconfirmed data frames

Figure I.1 illustrates the steps executed by an end-device transmitting a single unconfirmed data frame.

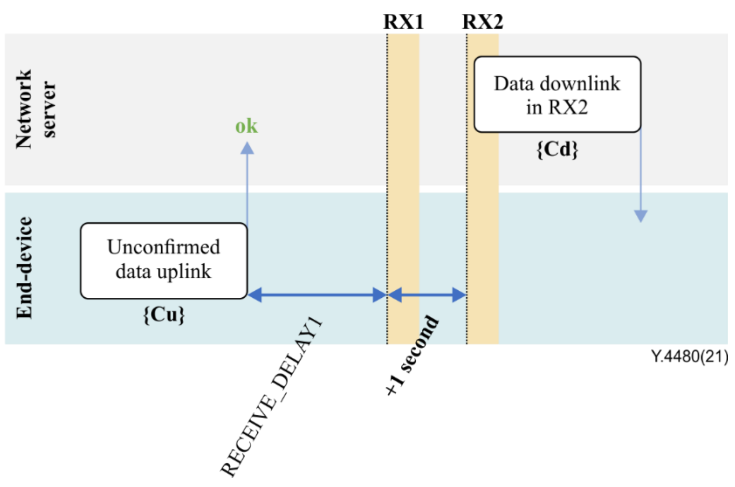

Figure I.1 – Uplink timing diagram for unconfirmed data frames, `NbTrans=1`

The end-device first transmits an unconfirmed data frame at an arbitrary instant and on an arbitrary channel. The uplink frame counter `Cu` is simply derived by adding `1` to the previous uplink frame counter. The network server receives the frame and may generate a downlink frame containing an application payload and/or MAC commands exactly RECEIVE_DELAY1 or RECEIVE_DELAY2 seconds later, using either the first or the second Class A receive windows, respectively. This downlink frame uses a data rate and channel specified by the regional channel plan. The downlink frame counter `Cd` is also derived by adding `1` to the downlink frame counter previously used to transmit to that specific end-device.

The uplink frame counter obeys the constraints imposed by `NbTrans`: The end-device is compelled to transmit the uplink `NbTrans` times or until a downlink is received in the Class A receive windows. In this example, as `NbTrans=1`, the next uplink will be sent with `FCntUp` = `Cu` + `1`. This transmission complies with all the specified behaviours defined in this Recommendation, including channel selection, timing randomization and duty-cycle limitations.

Figure I.2 illustrates another example where an end-device transmits a single payload with unconfirmed uplink data frames using `NbTrans` = `3`.

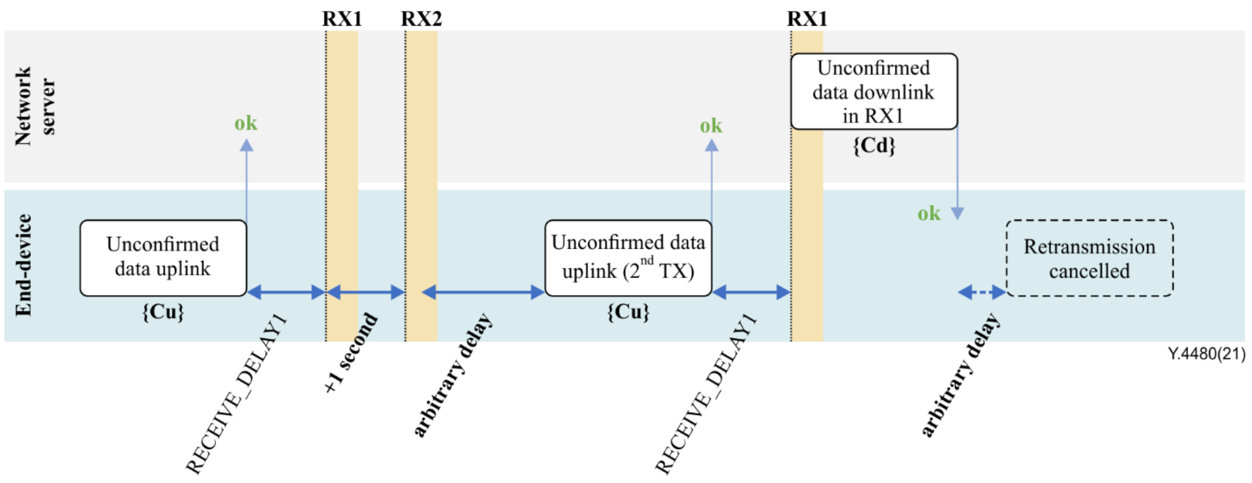

Figure I.2 – Uplink timing diagram for unconfirmed data frames, `NbTrans`>2

The end-device transmits up to `NbTrans` times the same unconfirmed uplink. The uplink frame counter `Cu` is kept constant for the retransmissions. In this example, the first transmission is received correctly by the network server. No downlink transmission occurs in the corresponding Class A windows, as no response is required: The network server might not have data or MAC commands to transmit to the end-device, or a downlink transmission to another end-device might be active. In the absence of downlink, end-device waits an arbitrary delay after the end of second Class A window, before transmitting the uplink again with same `Cu`, using a different channel. A data downlink frame is then received, on the first Class A window following this second transmission. As a consequence of this downlink reception, the third transmission does not occur.

This transmission complies with all the specified behaviours defined in this Recommendation, including channel selection, timing randomization and duty-cycle limitations.

I.2 Uplink timing diagram for confirmed data frames

The diagram of Figure I.3 illustrates the steps executed by an end-device transmitting two confirmed data frames (`Data0` and `Data1`) with `NbTrans=1`.

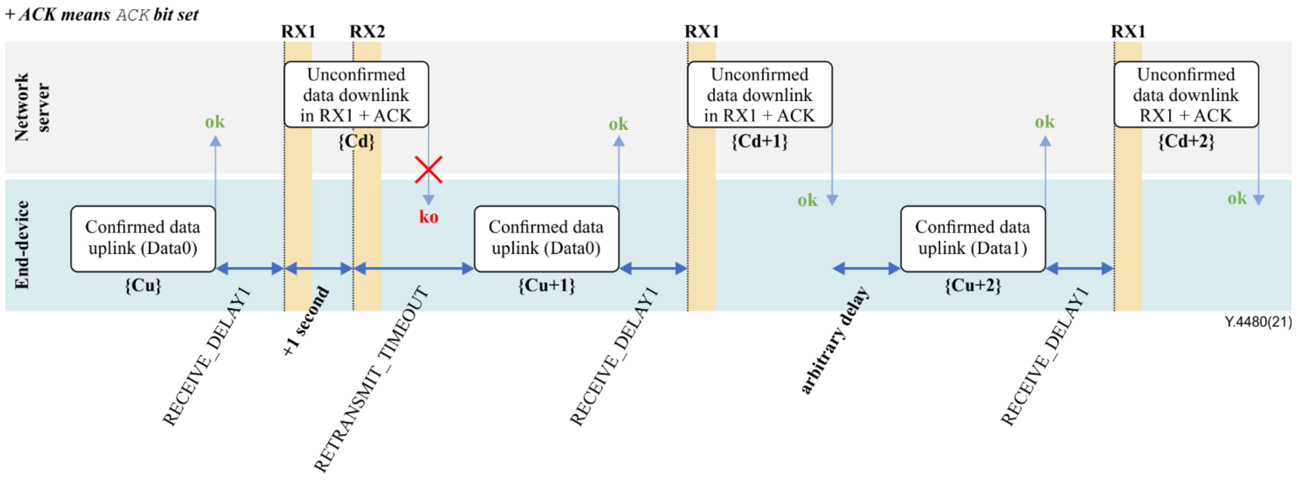
+ ACK means ACK bit set
RX1
RX2
RX1
RX1
Network server
End-device
ok
Unconfirmed data downlink in RX1 + ACK
{Cd}
ko
ok
Unconfirmed data downlink in RX1 + ACK
{Cd+1}
ok
ok
Unconfirmed data downlink RX1 + ACK
{Cd+2}
ok
Confirmed data uplink (Data0)
{Cu}
Confirmed data uplink (Data0)
{Cu+1}
Confirmed data uplink (Data1)
{Cu+2}
RECEIVE_DELAY1
+1 second
RETRANSMIT_TIMEOUT
RECEIVE_DELAY1
arbitrary delay
RECEIVE_DELAY1
Y.4480(21)

Figure I.3 – Uplink timing diagram for confirmed data frames

The end-device first transmits a confirmed data frame containing the `Data0` payload at an arbitrary instant and on an arbitrary channel. The uplink frame counter `Cu` is simply derived by adding `1` to the previous uplink frame counter. The network server receives the frame and generates a downlink frame with the `ACK` bit set, which is transmitted exactly RECEIVE_DELAY1 or RECEIVE_DELAY2 seconds later, using the first or second receive window of the end-device, respectively. This downlink frame uses a data rate and channel specified by the regional channel plan. The downlink frame counter `Cd` is also derived by adding `1` to the downlink frame counter previously used to transmit to that specific end-device. If there is no downlink payload pending, the network server may send a frame without a payload. In this example, the frame carrying the `ACK` bit is not received by the end-device. The second receive window is here opened, because frame reception failure happens before this window starts.

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 payload after waiting at least RETRANSMIT_TIMEOUT seconds after RX2. The uplink frame counter obeys the constraints imposed by `NbTrans`: The end-device is compelled to transmit the uplink `NbTrans` times or until a downlink is received in the Class A receive windows. In this example, as `NbTrans=1`, the repeated payload will be sent with `FCntUp` = `Cu+1`. After this repeated payload, the end-device receives the `ACK` downlink during its RX1, with `FCntDn` = `Cd+1`. The end-device is then free to transmit a new frame on a new channel and is not required to open RX2.

The downlink frames in this example carry an application payload. A downlink frame can carry any combination of `ACK`, MAC control commands and payload.

I.3 Downlink diagram for confirmed data frames

Figure I.4 illustrates the basic sequence of a confirmed downlink data frame.

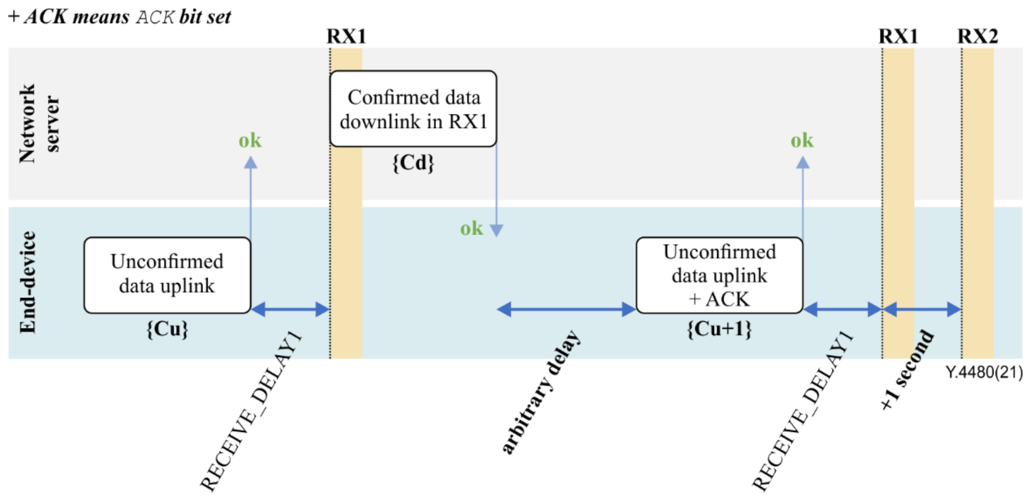

Figure I.4 – Downlink timing diagram for confirmed data frames

The frame exchange may be initiated by the end-device transmitting a confirmed or unconfirmed data frame, or autonomously by the network server using the end-device's Class B ping-slot or Class C RXC window (if the end-device is currently Class B-enabled or Class C-enabled). Upon receiving the downlink data frame requiring an acknowledgement, the end-device transmits an uplink data frame with the ACK bit set at the time of its choosing. This frame might also contain piggybacked application payload data and/or MAC commands. This uplink is treated like any other uplink, and as such, this transmission complies with all the specified behaviours defined in this Recommendation, including channel selection, timing randomization and duty-cycle limitations.

NOTE 1 – To allow end-devices to be as simple as possible and keep as few states as possible, the end-device may transmit an explicit (possibly empty) acknowledgement data frame immediately after receiving a data frame requiring an acknowledgment. Alternatively, the end-device may defer the transmission of an acknowledgement to piggyback it with its next data frame.

NOTE 2 – As there is no specified timing of the acknowledgement sent by the end-device, out-of-band coordination between the network server and the Class B or Class C-capable end-device is recommended to prevent excessive retransmissions of confirmed downlinks on Class B ping-slots or Class C RXC windows.

I.4 Downlink timing for frame-pending frames

Figure I.5 illustrates the use of the `FPending` bit on a downlink. The `FPending` bit can only be set on a downlink frame. When present in a downlink received in a Class A receive window, the `FPending` bit informs the end-device that the network has one or more downlink frames pending for the end-device. When present in a downlink received in a Class B ping-slot, `FPending` is used to prioritize conflicting ping slots (see clause 13.2 for details). `FPending` has no meaning in a downlink received in a Class C RXC window.

If a Class A end-device receives a frame where `FPending` is set, it is recommended that the end-device transmit an uplink data frame as soon as reasonably possible, which would allow the end-device to receive more information from the network. However, the precise timing of the uplink transmission is not specified and is application-dependent.

NOTE – The `FPending` bit is independent of the frame-acknowledgment mechanism.

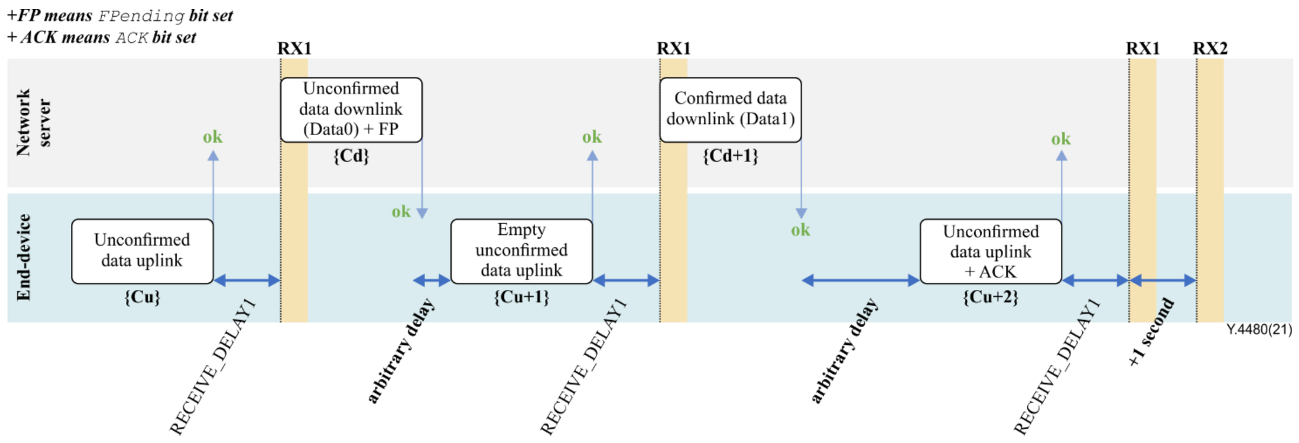

Figure I.5 – Downlink timing diagram for frame-pending frames: example 1

In this example, the network server has two data frames to transmit to the end-device. The frame exchange is initiated by the end-device via a normal unconfirmed uplink data frame. The network server uses the first receive window to transmit the `Data0` downlink data frame with the bit `FPending` set, as an unconfirmed downlink data frame. The end-device, in response to the `FPending` indication, quickly transmits an empty unconfirmed data frame. Exactly RECEIVE_DELAY1 seconds later, the network server transmits the second downlink data frame Data1, using a confirmed downlink data frame but with the `FPending` bit cleared. The end-device transmits an uplink data frame with the `ACK` bit set to acknowledge the confirmed downlink data frame `Data1`.

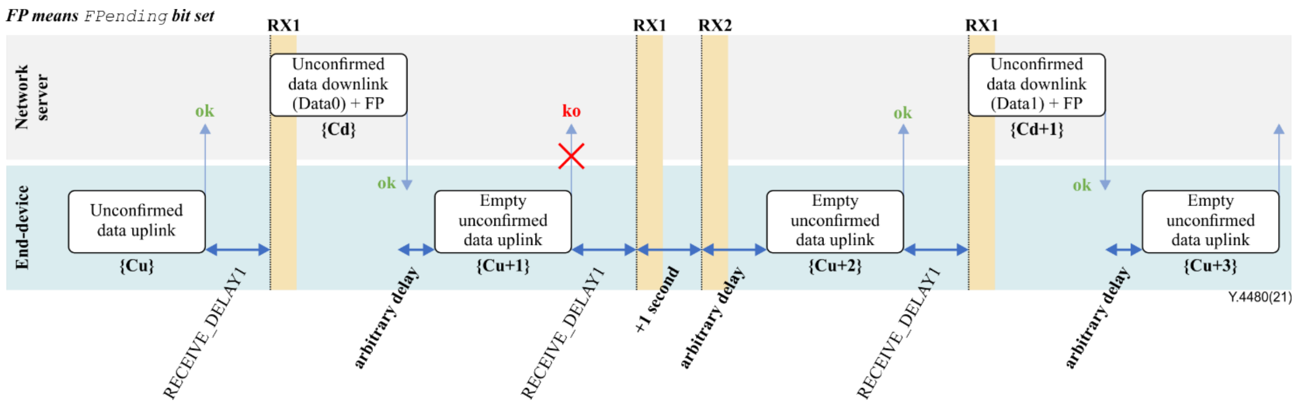

Figure I.6 – Downlink timing diagram for frame-pending frames: example 2

In this example, the downlink data frames are both unconfirmed frames with the `FPending` bit set, and as such, the end-device does not need to transmit an acknowledgement. After receiving the `Data0` unconfirmed downlink data frame with the `FPending` bit set, the end-device sends an empty data frame at a time of its own choosing. As this uplink is not received by the network, the network server is then still waiting for a spontaneous uplink from the end-device to execute the transfer. The end-device may, at its discretion, offer the network server more transmission opportunities by sending a new empty data frame.

The `FPending` bit, the `ACK` bit, and payload data may all be present in the same downlink. The frame exchange in Figure I.7 is a perfectly valid example with `NbTrans=1`.

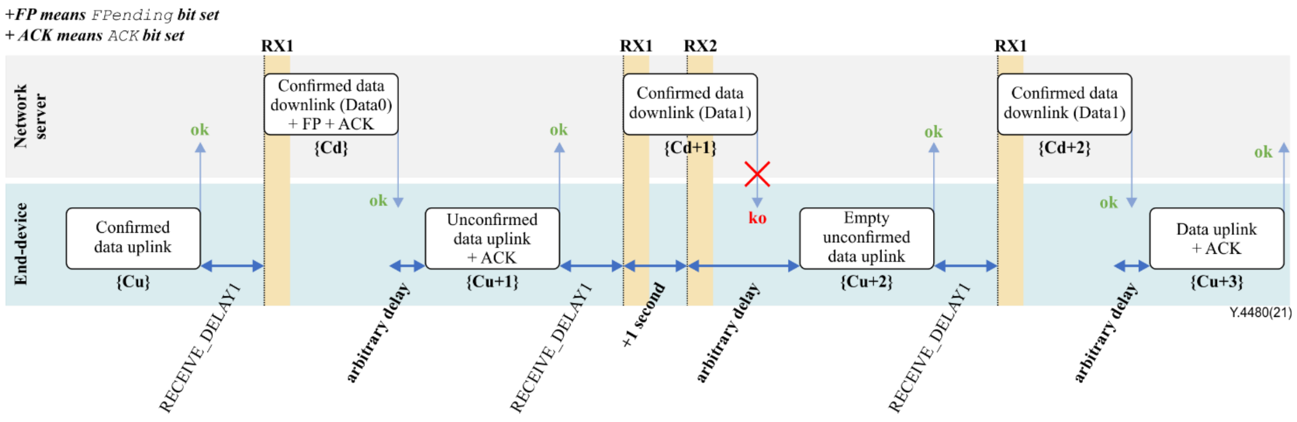

Figure I.7 – Downlink timing diagram for frame-pending frames: example 3

The end-device sends a confirmed uplink data frame. The network server may then answer with a confirmed downlink data frame containing the `Data0` application payload, `ACK` bit set and `FPending` bit set. After an arbitrary delay, the end-device then replies with an uplink with the `ACK` bit set. The end-device misses the next downlink containing `Data1`, this data was expected by the end-device because of the previous frame `FPending` indication. After an arbitrary delay, it offers the network server another opportunity to send the pending data using an empty frame. This time, `Data1` is received by the end-device and acknowledged.

Bibliography

[b-ITU-T G.993.5]	Recommendation ITU-T G.993.5 Corrigendum 1 (2020), *Self-FEXT cancellation (vectoring) for use with VDSL2 transceivers Corrigendum 1*.
[b-LoRaWAN RP002]	LoRa Alliance Technical Committee, RP002-1.0.0 (November 2019), *LoRaWAN® Regional Parameters*. < https://lora-alliance.org/wp-content/uploads/2019/11/rp_2-1.0.0_final_release.pdf>
[b-LoRaWAN TR001]	LoRa Alliance Technical Committee, TR001 (August 2018), *Technical Recommendations for Preventing State Synchronization Issues around LoRaWAN™ 1.0.x Join Procedure*. < https://lora-alliance.org/wp-content/uploads/2020/11/lorawan-1.0.x-join-synch-issues-remedies-v1.0.0.pdf>
[b-LoRaWAN TS001-1.0.3]	LoRa Alliance Technical Committee, TS001-1.0.3 (July 2018), *LoRaWAN™ Specification version 1.0.3*. < https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf>
[b-LoRaWAN TS001-1.0.4]	LoRa Alliance Technical Committee, TS001-1.0.4 (October 2020), *LoRaWAN® Link Layer Specification version 1.0.4*. < https://lora-alliance.org/resource_hub/lorawan-104-specification-package/>
[b-LoRaWAN TS002]	LoRa Alliance Technical Committee, TS002-1.0 (October 2017), *LoRaWAN™ Backend Interfaces 1.0 Specification*. < https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm-backend-interfaces-v1.0.pdf>
[b-LoRaWAN TS005]	LoRa Alliance Technical Committee, TS005-1.0.0 (September 2018), *LoRaWAN® Remote Multicast Setup Specification version 1.0.0*. < https://lora-alliance.org/wp-content/uploads/2020/11/remote_multicast_setup_v1.0.0.pdf>
[b-LoRaWAN TS009]	LoRa Alliance Certification Committee, TS009-1.0.0 (October 2020), *LoRaWAN® Certification Protocol Specification*. < https://lora-alliance.org/resource_hub/lorawan-104-specification-package/>

Printed in Switzerland
Geneva, 2021