

LoRaWAN™ 1.1 Specification

Copyright © 2017 LoRa Alliance, Inc. All rights reserved.

NOTICE OF USE AND DISCLOSURE

Copyright © LoRa Alliance, Inc. (2017). All Rights Reserved.

The information within this document is the property of the LoRa Alliance (“The Alliance”) and its use and disclosure are subject to LoRa Alliance Corporate Bylaws, Intellectual Property Rights (IPR) Policy and Membership Agreements.

Elements of LoRa Alliance specifications may be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights (such a third party may or may not be a member of LoRa Alliance). The Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

This document and the information contained herein are provided on an “AS IS” basis and THE ALLIANCE DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO (A) ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OF THIRD PARTIES (INCLUDING WITHOUT LIMITATION ANY INTELLECTUAL PROPERTY RIGHTS INCLUDING PATENT, COPYRIGHT OR TRADEMARK RIGHTS) OR (B) ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT.

IN NO EVENT WILL THE ALLIANCE BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF DATA, INTERRUPTION OF BUSINESS, OR FOR ANY OTHER DIRECT, INDIRECT, SPECIAL OR EXEMPLARY, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, IN CONTRACT OR IN TORT, IN CONNECTION WITH THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The above notice and this paragraph must be included on all copies of this document that are made.

LoRa Alliance, Inc.
3855 SW 153rd Drive
Beaverton, OR 97003

Note: All Company, brand and product names may be trademarks that are the sole property of their respective owners.



LoRaWAN™ 1.1 Specification

Authored by the LoRa Alliance Technical Committee

Chairs:

N.SORNIN (Semtech), A.YEGIN (Actility)

Editor:

N.SORNIN (Semtech)

Contributors:

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

Version: 1.1

Date: October 11, 2017

Status: Final release

Contents

68	Contents	
69	1 Introduction	8
70	1.1 LoRaWAN Classes	8
71	1.2 Conventions	9
72	2 Introduction on LoRaWAN options	10
73	2.1 LoRaWAN Classes	10
74	Class A – All end-devices	11
75	3 Physical Message Formats	12
76	3.1 Uplink Messages	12
77	3.2 Downlink Messages	12
78	3.3 Receive Windows	13
79	3.3.1 First receive window channel, data rate, and start	14
80	3.3.2 Second receive window channel, data rate, and start	14
81	3.3.3 Receive window duration	14
82	3.3.4 Receiver activity during the receive windows	14
83	3.3.5 Network sending a message to an end-device	14
84	3.3.6 Important notice on receive windows	14
85	3.3.7 Receiving or transmitting other protocols	15
86	4 MAC Message Formats	16
87	4.1 MAC Layer (PHYPayload)	16
88	4.2 MAC Header (MHDR field)	17
89	4.2.1 Message type (MType bit field)	17
90	4.2.2 Major version of data message (Major bit field)	18
91	4.3 MAC Payload of Data Messages (MACPayload)	18
92	4.3.1 Frame header (FHDR)	18
93	4.3.2 Port field (FPort)	24
94	4.3.3 MAC Frame Payload Encryption (FRMPayload)	25
95	4.4 Message Integrity Code (MIC)	26
96	4.4.1 Downlink frames	26
97	4.4.2 Uplink frames	27
98	5 MAC Commands	29
99	5.1 Reset indication commands (<i>ResetInd</i> , <i>ResetConf</i>)	32
100	5.2 Link Check commands (<i>LinkCheckReq</i> , <i>LinkCheckAns</i>)	33
101	5.3 Link ADR commands (<i>LinkADRReq</i> , <i>LinkADRAns</i>)	33
102	5.4 End-Device Transmit Duty Cycle (<i>DutyCycleReq</i> , <i>DutyCycleAns</i>)	35
103	5.5 Receive Windows Parameters (<i>RXParamSetupReq</i> , <i>RXParamSetupAns</i>)	36
104	5.6 End-Device Status (<i>DevStatusReq</i> , <i>DevStatusAns</i>)	37
105	5.7 Creation / Modification of a Channel (<i>NewChannelReq</i> , <i>NewChannelAns</i> , <i>DiChannelReq</i> , <i>DiChannelAns</i>)	38
107	5.8 Setting delay between TX and RX (<i>RXTimingSetupReq</i> , <i>RXTimingSetupAns</i>)	40
108	5.9 End-device transmission parameters (<i>TxParamSetupReq</i> , <i>TxParamSetupAns</i>)	41
109	5.10 Rekey indication commands (<i>RekeyInd</i> , <i>RekeyConf</i>)	42
110	5.11 ADR parameters (<i>ADRParamSetupReq</i> , <i>ADRParamSetupAns</i>)	43
111	5.12 DeviceTime commands (<i>DeviceTimeReq</i> , <i>DeviceTimeAns</i>)	44
112	5.13 Force Rejoin Command (<i>ForceRejoinReq</i>)	44
113	5.14 RejoinParamSetupReq (<i>RejoinParamSetupAns</i>)	45
114	6 End-Device Activation	47
115	6.1 Data Stored in the End-device	47
116	6.1.1 Before Activation	47
117	6.1.2 After Activation	49
118	6.2 Over-the-Air Activation	52

119	6.2.1	Join procedure.....	52
120	6.2.2	Join-request message	52
121	6.2.3	Join-accept message.....	53
122	6.2.4	ReJoin-request message.....	57
123	6.2.5	Key derivation diagram.....	61
124	6.3	Activation by Personalization	64
125	7	Retransmissions back-off.....	65
126		Class B – Beacon	66
127	8	Introduction to Class B.....	67
128	9	Principle of synchronous network initiated downlink (Class-B option).....	68
129	10	Uplink frame in Class B mode.....	71
130	11	Downlink Ping frame format (Class B option)	72
131	11.1	Physical frame format	72
132	11.2	Unicast & Multicast MAC messages.....	72
133	11.2.1	Unicast MAC message format.....	72
134	11.2.2	Multicast MAC message format.....	72
135	12	Beacon acquisition and tracking.....	73
136	12.1	Minimal beacon-less operation time	73
137	12.2	Extension of beacon-less operation upon reception	73
138	12.3	Minimizing timing drift.....	73
139	13	Class B Downlink slot timing	74
140	13.1	Definitions	74
141	13.2	Slot randomization	75
142	14	Class B MAC commands	76
143	14.1	PingSlotInfoReq	76
144	14.2	BeaconFreqReq.....	77
145	14.3	PingSlotChannelReq.....	78
146	14.4	BeaconTimingReq & BeaconTimingAns.....	79
147	15	Beaconing (Class B option).....	80
148	15.1	Beacon physical layer	80
149	15.2	Beacon frame content	80
150	15.3	Beacon <i>GwSpecific</i> field format.....	81
151	15.3.1	Gateway GPS coordinate:InfoDesc = 0, 1 or 2	82
152	15.4	Beaconing precise timing	82
153	15.5	Network downlink route update requirements.....	82
154	16	Class B unicast & multicast downlink channel frequencies.....	84
155	16.1	Single channel beacon transmission	84
156	16.2	Frequency-hopping beacon transmission.....	84
157		Class C – Continuously listening	85
158	17	Class C: Continuously listening end-device.....	86
159	17.1	Second receive window duration for Class C	86
160	17.2	Class C Multicast downlinks.....	87
161	18	Class C MAC command.....	88
162	18.1	Device Mode (DeviceModelInd, DeviceModeConf)	88
163		Support information.....	89
164	19	Examples and Application Information	90
165	19.1	Uplink Timing Diagram for Confirmed Data Messages	90
166	19.2	Downlink Diagram for Confirmed Data Messages	90
167	19.3	Downlink Timing for Frame-Pending Messages	91
168	20	Recommendation on contract to be provided to the network server by the end-	
169		device provider at the time of provisioning	93
170	21	Recommendation on finding the locally used channels	94
171	22	Revisions	95

172	22.1	Revision 1.0	95
173	22.2	Revision 1.0.1	95
174	22.3	Revision 1.0.2	95
175	22.4	Revision 1.1	96
176	22.4.1	Clarifications.....	96
177	22.4.2	Functional modifications	96
178	22.4.3	Examples	98
179	23	Glossary	99
180	24	Bibliography	100
181	24.1	References.....	100
182	25	NOTICE OF USE AND DISCLOSURE.....	101
183			

184 Tables

185	Table 1:	MAC message types	17
186	Table 2:	Major list.....	18
187	Table 3:	FPort list.....	26
188	Table 4:	MAC commands.....	31
189	Table 5:	Channel state table	34
190	Table 6:	LinkADRAAns status bits signification	35
191	Table 7:	RXParamSetupAns status bits signification	37
192	Table 8:	Battery level decoding	37
193	Table 9:	NewChannelAns status bits signification	39
194	Table 10:	DIChannelAns status bits signification	40
195	Table 11:	RXTimingSetup Delay mapping table	40
196	Table 12 :	TxParamSetup EIRP encoding table	41
197	Table 13 :	JoinReqType values	55
198	Table 14 :	Join-Accept encryption key.....	55
199	Table 15 :	summary of RejoinReq messages.....	58
200	Table 18 :	transmission conditions for RejoinReq messages.....	60
201	Table 19 :	Join-request dutycycle limitations	65
202	Table 20:	Beacon timing	74
203	Table 21 :	classB slot randomization algorithm parameters.....	75
204	Table 22 :	classB MAC command table	76
205	Table 23 :	beacon infoDesc index mapping.....	81
206	Table 24 :	Class C MAC command table.....	88
207	Table 25 :	DeviceModInd class mapping.....	88
208			

209 Figures

210	Figure 1:	LoRaWAN Classes	10
211	Figure 2:	Uplink PHY structure.....	12
212	Figure 3:	Downlink PHY structure	12
213	Figure 4:	End-device receive slot timing.....	13
214	Figure 5:	Radio PHY structure (CRC* is only available on uplink messages)	16
215	Figure 6:	PHY payload structure	16
216	Figure 7:	MAC payload structure.....	16
217	Figure 8:	Frame header structure.....	16
218	Figure 9:	PHY paylod format.....	16
219	Figure 10:	MAC header field content.....	17

220	Figure 11 : Frame header format	18
221	Figure 12 : downlink FCtrl fields	19
222	Figure 13 : uplink FCtrl fields.....	19
223	Figure 14 : data rate back-off sequence example.....	20
224	Figure 15 : Encryption block format.....	24
225	Figure 16 : MACPayload field size	25
226	Figure 17 : Encryption block format.....	26
227	Figure 18 : downlink MIC computation block format	27
228	Figure 19 : uplink B ₀ MIC computation block format	27
229	Figure 20 : uplink B ₁ MIC computation block format	27
230	Figure 34 : ResetInd payload format	32
231	Figure 35 : ResetConf payload format.....	33
232	Figure 21: LinkCheckAns payload format.....	33
233	Figure 22 : LinkADRReq payload format.....	33
234	Figure 23 : LinkADRAns payload format	35
235	Figure 24 : DutyCycleReq payload format.....	36
236	Figure 25 : RXParamSetupReq payload format	36
237	Figure 26 : RXParamSetupAns payload format.....	37
238	Figure 27 : DevStatusAns payload format	37
239	Figure 28 : NewChannelReq payload format.....	38
240	Figure 29 : NewChannelAns payload format	38
241	Figure 30 : DLChannelReq payload format	39
242	Figure 31 : DLChannelAns payload format.....	39
243	Figure 32 : RXTimingSetupReq payload format	40
244	Figure 33 : TxParamSetupReq payload format	41
245	Figure 36 : RekeyInd payload format	42
246	Figure 37 : RekeyConf payload format.....	43
247	Figure 38 : ADRParamSetupReq payload format.....	43
248	Figure 39 : DeviceTimeAns payload format.....	44
249	Figure 40 : ForceRejoinReq payload format.....	44
250	Figure 41 : RejoinParamSetupReq payload format	45
251	Figure 42 : RejoinParamSetupAns payload format.....	46
252	Figure 43 : DevAddr fields.....	49
253	Figure 44 : Join-request message fields.....	52
254	Figure 45 : Join-accept message fields	53
255	Figure 46: Rejoin-request type 0&2 message fields	58
256	Figure 47: Rejoin-request type 1 message fields.....	59
257	Figure 48 : LoRaWAN1.0 key derivation scheme	62
258	Figure 49 : LoRaWAN1.1 key derivation scheme	63
259	Figure 50: Beacon reception slot and ping slots	70
260	Figure 51 : classB FCtrl fields	71
261	Figure 52 : beacon-less temporary operation	73
262	Figure 53: Beacon timing	74
263	Figure 54 : PingSlotInfoReq payload format.....	76
264	Figure 55 : BeaconFreqReq payload format.....	77
265	Figure 56 : BeaconFreqAns payload format	77
266	Figure 57 : PingSlotChannelReq payload format.....	78
267	Figure 58 : PingSlotFreqAns payload format.....	78
268	Figure 59 : beacon physical format	80
269	Figure 60 : beacon frame content.....	80
270	Figure 61 : example of beacon CRC calculation (1)	80
271	Figure 62 : example of beacon CRC calculation (2)	81
272	Figure 63 : beacon GwSpecific field format	81

273	Figure 64 : beacon Info field format.....	82
274	Figure 65: Class C end-device reception slot timing.....	87
275	Figure 66 : DeviceModelInd payload format.....	88
276	Figure 67: Uplink timing diagram for confirmed data messages	90
277	Figure 68: Downlink timing diagram for confirmed data messages.....	91
278	Figure 69: Downlink timing diagram for frame-pending messages, example 1	91
279	Figure 70: Downlink timing diagram for frame-pending messages, example 2	92
280	Figure 71: Downlink timing diagram for frame-pending messages, example 3	92
281		

1 Introduction

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

LoRaWAN networks typically are laid out in a star-of-stars topology in which **gateways**¹ relay messages between **end-devices**² and a central **Network Server** the Network Server routes the packets from each device of the network to the associated **Application Server**. To secure radio transmissions the LoRaWAN protocol relies on symmetric cryptography using session keys derived from the device's root keys. In the backend the storage of the device's root keys and the associated key derivation operations are insured by a **Join Server**.

This specification treats the Network Server, Application Server, and Join Server as if they are always co-located. Hosting these functionalities across multiple disjoint network nodes is outside the scope of this specification but is covered by [BACKEND].

Gateways are connected to the Network Server via secured standard IP connections while end-devices use single-hop LoRa™ or FSK communication to one or many gateways.³ All communication is generally bi-directional, although uplink communication from an end-device to the Network Server is expected to be the predominant traffic.

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

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

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

Note: Maximum transmit duty-cycle and dwell time per sub-band are region specific and are defined in [PHY]

1.1 LoRaWAN Classes

All LoRaWAN devices MUST implement at least the Class A functionality as described in this document. In addition they MAY implement options named Class B or Class C as also

¹ Gateways are also known as **concentrators** or **base stations**.

² End-devices are also known as **nodes**.

³ Support for intermediate elements – repeaters – is not described in the document, however payload restrictions for encapsulation overhead are included in this specification. A repeater is defined as using LoRaWAN as its backhaul mechanism.

320 described in this document or others to be defined. In all cases, they MUST remain
321 compatible with Class A.

322 1.2 Conventions

323
324 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
325 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be
326 interpreted as described in RFC 2119.

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

329 In this document,

- 330 • The over-the-air octet order for all multi-octet fields is little endian
- 331 • EUI are 8 bytes multi-octet fields and are transmitted as little endian.
- 332 • By default, RFU bits SHALL be set to zero by the transmitter of the message and
333 SHALL be ignored by the receiver

2 Introduction on LoRaWAN options

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

2.1 LoRaWAN Classes

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

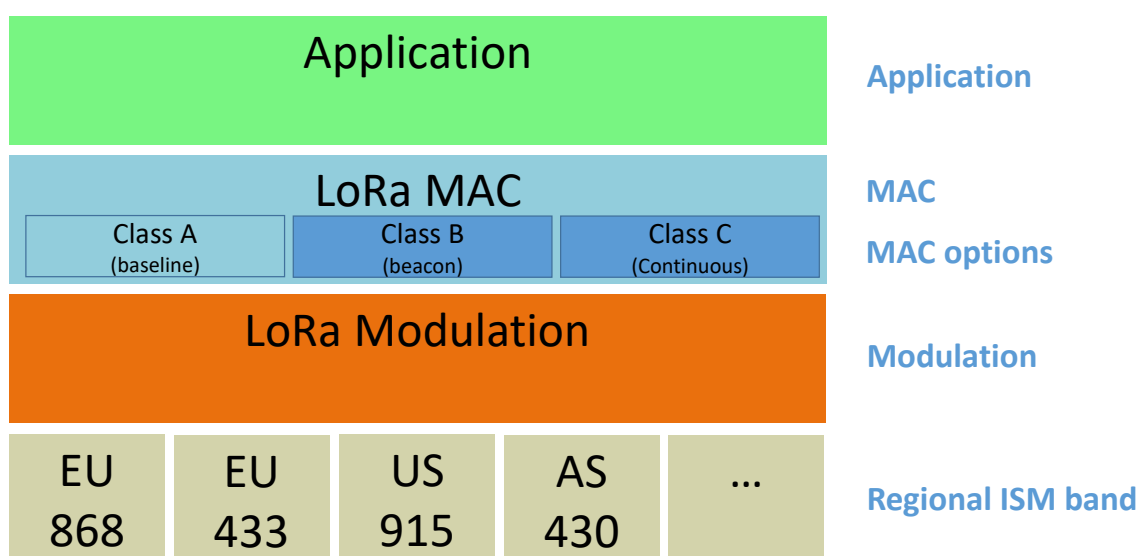


Figure 1: LoRaWAN Classes

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

CLASS A – ALL END-DEVICES

361

362 All LoRaWAN end-devices MUST implement Class A features.

3 Physical Message Formats

The LoRa terminology distinguishes between uplink and downlink messages.

3.1 Uplink Messages

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

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

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

Uplink PHY:

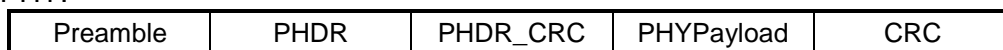


Figure 2: Uplink PHY structure

3.2 Downlink Messages

Each **downlink message** is sent by the Network Server to only one end-device and is relayed by a single gateway.²

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

Downlink PHY:

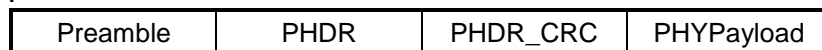


Figure 3: Downlink PHY structure

¹ See the LoRa radio transceiver datasheet for a description of LoRa radio packet implicit/explicit modes.

² This specification does not describe the transmission of multicast messages from a network server to many end-devices.

³ No payload integrity check is done at this level to keep messages as short as possible with minimum impact on any duty-cycle limitations of the ISM bands used.

3.3 Receive Windows

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

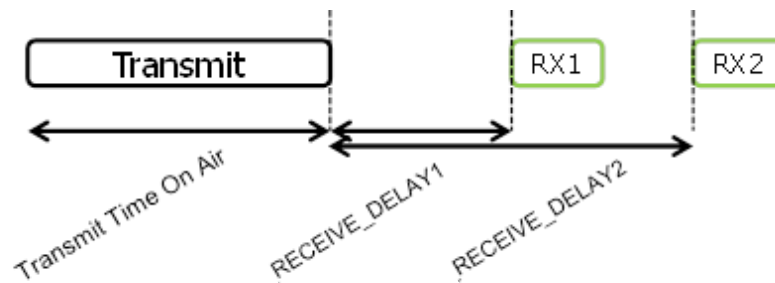


Figure 4: End-device receive slot timing.

3.3.1 First receive window channel, data rate, and start

The first receive window RX1 uses a frequency that is a function of the uplink frequency and a data rate that is a function of the data rate used for the uplink. RX1 opens RECEIVE_DELAY1¹ seconds (+/- 20 microseconds) after the end of the uplink modulation. The relationship between uplink and RX1 slot downlink data rate is region specific and detailed in [PHY]. By default, the first receive window datarate is identical to the datarate of the last uplink.

3.3.2 Second receive window channel, data rate, and start

The second receive window RX2 uses a fixed configurable frequency and data rate and opens RECEIVE_DELAY2¹ seconds (+/- 20 microseconds) after the end of the uplink modulation. The frequency and data rate used can be modified through MAC commands (see Section 5). The default frequency and data rate to use are region specific and detailed in [PHY].

3.3.3 Receive window duration

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

3.3.4 Receiver activity during the receive windows

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

3.3.5 Network sending a message to an end-device

If the network intends to transmit a downlink to an end-device, it MUST initiate the transmission precisely at the beginning of at least one of the two receive windows. If a downlink is transmitted during both windows, identical frames MUST be transmitted during each window.

3.3.6 Important notice on receive windows

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

¹ RECEIVE_DELAY1 and RECEIVE_DELAY2 are described in Chapter 6.

418 3.3.7 Receiving or transmitting other protocols

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

4 MAC Message Formats

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

Radio PHY layer:

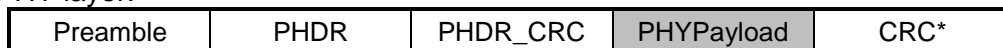
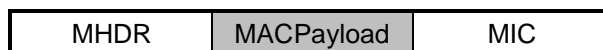
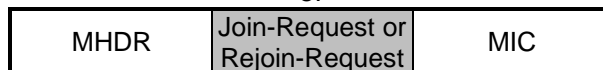


Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

PHYPayload:



or



or



Figure 6: PHY payload structure

MACPayload:

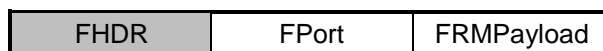


Figure 7: MAC payload structure

FHDR:

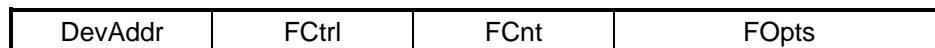


Figure 8: Frame header structure

4.1 MAC Layer (PHYPayload)

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

Figure 9: PHY payload format

¹ Maximum payload size is detailed in the Chapter 6.

² For Join-Accept frame, the MIC field is encrypted with the payload and is not a separate field

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

4.2 MAC Header (MHDR field)

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

Figure 10: MAC header field content

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

4.2.1 Message type (MType bit field)

The LoRaWAN distinguishes between 8 different MAC message types: **Join-request**, **Rejoin-request**, **Join-accept**, **unconfirmed data up/down**, and **confirmed data up/down** and **proprietary** protocol messages.

MType	Description
000	Join-request
001	Join-accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-request
111	Proprietary

Table 1: MAC message types

4.2.1.1 Join-request and join-accept messages

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

4.2.1.2 Data messages

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

¹ A detailed timing diagram of the acknowledge mechanism is given in Section 19.

among devices that have a common understanding of the proprietary extensions. When an end-device or a Network Server receives an unknown proprietary message, it SHALL silently drop it.

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

4.2.2 Major version of data message (Major bit field)

Major bits	Description
00	LoRaWAN R1
01..11	RFU

Table 2: Major list

Note: The Major version specifies the format of the messages exchanged in the join procedure (see Chapter 6.2) and the first four bytes of the MAC Payload as described in Chapter 4. For each major version, end-devices may implement different minor versions of the frame format. The minor version used by an end-device must be made known to the Network Server beforehand using out of band messages (e.g., as part of the device personalization information). When a device or a Network Server receives a frame carrying an unknown or unsupported version of LoRaWAN, it SHALL silently drop it.

4.3 MAC Payload of Data Messages (MACPayload)

The MAC payload of the data messages, 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.

4.3.1 Frame header (FHDR)

The **FHDR** contains the short device address of the end-device (**DevAddr**), a frame control octet (**FCtrl**), a 2-octets frame counter (**FCnt**), and up to 15 octets of frame options (**FOpts**) used to transport MAC commands. . If present, the FOpts field shall be encrypted using the NwkSEncKey as described in section 4.3.1.6.

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

Figure 11 : Frame header format

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

Bit#	7	6	5	4	[3..0]
------	---	---	---	---	--------

FCtrl bits	ADR	RFU	ACK	FPending	FOptsLen
------------	-----	-----	-----	----------	----------

Figure 12 : downlink FCtrl fields

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

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

Figure 13 : uplink FCtrl fields

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

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

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

If the uplink **ADR** bit is set, the network will control the data rate and Tx power of the end-device through the appropriate MAC commands. If the **ADR** bit is not set, the network will not attempt to control the data rate nor the transmit power of the end-device regardless of the received signal quality. The network MAY still send commands to change the Channel mask or the frame repetition parameters.

When the downlink ADR bit is set, it informs the end-device that the Network Server is in a position to send ADR commands. The device MAY set/unset the uplink ADR bit.

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

- unset the ADR uplink bit, and control its uplink data rate following its own strategy. This SHOULD be the typical strategy for a mobile end-device.
- Ignore it (keep the uplink ADR bit set) and apply the normal data rate decay in the absence of ADR downlink commands. This SHOULD be the typical strategy for a stationary end-device.

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

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

Default Tx Power is the maximum transmission power allowed for the device considering device capabilities and regional regulatory constraints. Device shall use this power level, until the network asks for less, through the LinkADRReq MAC command.

If an end-device's data rate is optimized by the network to use a data rate higher than its default data rate, or a TXPower lower than its default TXPower, it periodically needs to validate that the network still receives the uplink frames. Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (**ADRACKReq**). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame resets the ADR_ACK_CNT counter. The downlink **ACK** bit does not need to be set as any response during the receive slot of the end-device indicates that the gateway has still received the uplinks from this device. If no reply is received within the next ADR_ACK_DELAY uplinks (i.e., after a total of ADR_ACK_LIMIT + ADR_ACK_DELAY), the end-device MUST try to regain connectivity by first stepping up the transmit power to default power if possible then switching to the next lower data rate that provides a longer radio range. The end-device MUST further lower its data rate step by step every time ADR_ACK_DELAY is reached. Once the device has reached the lowest data rate, it MUST re-enable all default uplink frequency channels.

The **ADRACKReq** SHALL not be set if the device uses its default data rate and transmit power because in that case no action can be taken to improve the link range.

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

Note: In uplink transmissions the **ADRACKReq** bit is set if ADR_ACK_CNT >= ADR_ACK_LIMIT and the current data-rate is greater than the device defined minimum data rate or its transmit power is lower than the default, or the current channel mask only uses a subset of all the default channels. It is cleared in other conditions.

The following table provides an example of data rate back-off sequence assuming ADR_ACK_LIMIT and ADR_ACK_DELAY constants are both equal to 32.

ADR_ACK_CNT	ADRACKReq bit	Data Rate	TX power	Channel Mask
0 to 63	0	SF11	Max – 9dBm	Single channel enabled
64 to 95	1	Keep	Keep	Keep
96 to 127	1	Keep	Max	Keep
128 to 159	1	SF12	Max	Keep
>= 160	0	SF12	MAX	All channels enabled

Figure 14 : data rate back-off sequence example

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

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

An acknowledgement is only sent in response to the latest message received and it is never retransmitted.

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

4.3.1.3 Retransmission procedure

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 acknowledge is not received, the application server is notified and may decide to retransmit a new “confirmed” frame.

Uplink frames:

Uplink “confirmed” & “unconfirmed” frames are transmitted “NbTrans” times (see 5.3) except if a valid downlink is received following one of the transmissions. The “NbTrans” parameter can be used by the network manager to control the redundancy of the node uplinks to obtain a given Quality of Service. The end-device SHALL perform frequency hopping as usual between repeated transmissions, It SHALL wait after each repetition until the receive windows have expired. The delay between the retransmissions is at the discretion of the end-device and MAY be different for each end-device.

The device SHALL stop any further retransmission of an uplink “confirmed” frame if a corresponding downlink acknowledgement frame is received

Class B&C devices SHALL stop any further retransmission of an uplink “unconfirmed” frame whenever a valid unicast downlink message is received during the RX1 slot window.

Class A devices SHALL stop any further retransmission of an uplink “unconfirmed” frame whenever a valid downlink message is received during the RX1 or the RX2 slot window.

If the network receives more than NbTrans transmissions of the same uplink frame, this may be an indication of a replay attack or a malfunctioning device, and therefore the network SHALL not process the extra frames.

NOTE: The network detecting a replay attack may take additional measures, such as reducing the NbTrans parameter to 1, or discarding uplink frames that are received over a channel that was already used

614 | by an earlier transmission of the same frame, or by some other
 615 | unspecified mechanism

616 4.3.1.4 Frame pending bit (**FPending** in FCtrl, downlink only)

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

620 The exact use of **FPending** bit is described in Chapter 19.3.

621 4.3.1.5 Frame counter (**FCnt**)

622 Each end-device has three frame counters to keep track of the number of data frames sent
 623 uplink to the Network Server (FCntUp), and sent downlink from the Network Server to the
 624 device (FCntDown).

625 In the downlink direction two different frame counter scheme exists; a single counter scheme
 626 in which all ports share the same downlink frame counter FCntDown when the device
 627 operates as a LoRaWAN1.0 device, and a two-counter scheme in which a separate
 628 NFCntDown is used for MAC communication on port 0 and when the FPort field is missing,
 629 and another AFCntDown is used for all other ports when the device operates as a
 630 LoRaWAN1.1 device.

631 In the two counters scheme the NFCntDown is managed by the Network Server, whereas
 632 the AFCntDown is managed by the application server.

633 | **Note:** LoRaWAN v1.0 and earlier support only one FCntDown counter
 634 | (shared across all ports) and the Network Server must take care to
 635 | support this scheme for devices prior to LoRaWAN v1.1.

636

637 Whenever an OTAA device successfully processes a Join-accept message, the frame
638 counters on the end-device (FCntUp) and the frame counters on the network side
639 (NFCntDown & AFCntDown) for that end-device are reset to 0.

640 ABP devices have their Frame Counters initialized to 0 at fabrication. In ABP devices the
641 frame counters MUST NEVER be reset during the device's life time. If the end-device is
642 susceptible of losing power during its life time (battery replacement for example), the frame
643 counters SHALL persist during such event.

644 Subsequently FCntUp is incremented with each uplink. NFCntDown is incremented with
645 each downlink on FPort 0 or when the FPort field is missing. AFCntDown is incremented
646 with each downlink on a port different than 0. At the receiver side, the corresponding
647 counter is kept in sync with the value received provided the value received has been
648 incremented compared to the current counter value and the message MIC field matches the
649 MIC value computed locally using the appropriate network session key . The FCnt is not
650 incremented in case of multiple transmissions of a confirmed or unconfirmed frame (see
651 NbTrans parameter). The Network Server SHALL drop the application payload of the
652 retransmitted frames and only forward a single instance to the application server.

653 Frame counters are 32 bits wide, The **FCnt** field corresponds to the least-significant 16 bits
654 of the 32-bits frame counter (i.e., FCntUp for data frames sent uplink and
655 AFCntDown/NFCntDown for data frames sent downlink).

656 The end-device SHALL NEVER reuse the same FCntUp value with the same application or
657 network session keys, except for retransmission of the same confirmed or unconfirmed
658 frame.

659 The end-device SHALL never process any retransmission of the same downlink frame.
660 Subsequent retransmissions SHALL be ignored without being processed.

661 **Note:** This means that the device will only acknowledge once the
662 reception of a downlink confirmed frame, similarly the device will only
663 generate a single uplink following the reception of a frame with the
664 FPending bit set.

665

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

4.3.1.6 Frame options (FOptsLen in FCtrl, FOpts)

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

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

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

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

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

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

The key *K* used is the NwkSEncKey for FOpts field in both the uplink and downlink direction.

The fields encrypted are: *pld* = **FOpts**

For each message, the algorithm defines a single Block **A**:

Size (bytes)	1	4	1	4	4	1	1
A	0x01	4 x 0x00	Dir	DevAddr	FCntUp or NFCntDwn	0x00	0x00

Figure 15 : Encryption block format

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

The block *A* is encrypted to get a block *S*:

$$S = \text{aes128_encrypt}(K, A)$$

Encryption and decryption of the **FOpts** is done by truncating (*pld* | pad₁₆) xor *S* to the first len(*pld*) octets.

4.3.1.7 Class B

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

4.3.2 Port field (FPort)

If the frame payload field is not empty, the port field MUST be present. If present, an **FPort** value of 0 indicates that the **FRMPayload** contains MAC commands only and any received frames with such an **FPort** shall be processed by the LoRaWAN implementation; see

Chapter 5 for a list of valid MAC commands. **FPort** values 1..223 (0x01..0xDF) are application-specific and any received frames with such an FPort SHALL be made available to the application layer by the LoRaWAN implementation. FPort value 224 is dedicated to LoRaWAN MAC layer test protocol. LoRaWAN implementation SHALL discard any transmission request from the application layer where the FPort value is not in the 1..224 range.

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

The test protocol, running at application layer, is defined outside of the LoRaWAN spec, as it is an application layer protocol.

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

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

Figure 16 : MACPayload field size

N is the number of octets of the application payload. The valid range for N is region specific and is defined in [PHY].

N MUST be equal or smaller than:

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

where M is the maximum MAC payload length.

4.3.3 MAC Frame Payload Encryption (FRMPayload)

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

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

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

FPort	Direction	K
-------	-----------	---

0	Uplink/downlink	NwkSEncKey
1..255	Uplink/downlink	AppSKey

Table 3: FPort list

The fields encrypted are:

$pld = \mathbf{FRMPayload}$

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

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

Figure 17 : Encryption block format

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

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

$S_i = \text{aes128_encrypt}(K, A_i)$ for $i = 1..k$

$S = S_1 \mid S_2 \mid \dots \mid S_k$

Encryption and decryption of the payload is done by truncating

$(pld \mid \text{pad}_{16}) \text{ xor } S$

to the first $\text{len}(pld)$ octets.

4.4 Message Integrity Code (MIC)

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

$\text{msg} = \mathbf{MHDR} \mid \mathbf{FHDR} \mid \mathbf{FPort} \mid \mathbf{FRMPayload}$

whereby $\text{len}(\text{msg})$ denotes the length of the message in octets.

4.4.1 Downlink frames

The **MIC** of a downlink frame is calculated as follows [RFC4493]:

$\text{cmac} = \text{aes128_cmac}(\mathbf{SNwkSIntKey}, B_0 \mid \text{msg})$

$\mathbf{MIC} = \text{cmac}[0..3]$

whereby the block B_0 is defined as follows:

Size (bytes)	1	2	2	1	4	4	1	1
B_0	0x49	ConfFCnt	2 x 0x00	Dir = 0x01	DevAddr	AFCntDwn or NFCntDwn	0x00	len(msg)

Figure 18 : downlink MIC computation block format

If the device is connected to a LoRaWAN1.1 Network Server and the ACK bit of the downlink frame is set, meaning this frame is acknowledging an uplink “confirmed” frame, then ConfFCnt is the frame counter value modulo 2^{16} of the “confirmed” uplink frame that is being acknowledged. In all other cases ConfFCnt = 0x0000.

4.4.2 Uplink frames

The MIC of uplink frames is calculated with the following process:

the block B_0 is defined as follows:

Size (bytes)	1	4	1	4	4	1	1
B_0	0x49	0x0000	Dir = 0x00	DevAddr	FCntUp	0x00	len(msg)

Figure 19 : uplink B_0 MIC computation block format

the block B_1 is defined as follows:

Size (bytes)	1	2	1	1	1	4	4	1	1
B_1	0x49	ConfFCnt	TxDr	TxCh	Dir = 0x00	DevAddr	FCntUp	0x00	len(msg)

Figure 20 : uplink B_1 MIC computation block format

Where:

- TxDr is the data rate used for the transmission of the uplink
- TxCh is the index of the channel used for the transmission.
- If the ACK bit of the uplink frame is set, meaning this frame is acknowledging a downlink “confirmed” frame, then ConfFCnt is the frame counter value modulo 2^{16} of the “confirmed” downlink frame that is being acknowledged. In all other cases ConfFCnt = 0x0000.

$cmacS = \text{aes128_cmac}(\text{SNwkSIntKey}, B_1 \parallel msg)$
 $cmacF = \text{aes128_cmac}(\text{FNwkSIntKey}, B_0 \parallel msg)$

If the device is connected to a LoRaWAN1.0 Network Server then:

MIC = $cmacF[0..3]$

802

803 If the device is connected to a LoRaWAN1.1 Network Server then:

804 **MIC** = *cmacS*[0..1] | *cmacF*[0..1]

805

806

5 MAC Commands

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

A single data frame can contain any sequence of MAC commands, either piggybacked in the **FOpts** field or, when sent as a separate data frame, in the **FRMPayload** field with the **FPort** field being set to 0. Piggybacked MAC commands are always sent encrypted and must not exceed 15 octets. MAC commands sent as **FRMPayload** are always encrypted and MUST NOT exceed the maximum **FRMPayload** length.

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

MAC Commands are answered/acknowledged by the receiving end in the same order than they are transmitted. The answer to each MAC command is sequentially added to a buffer. All MAC commands received in a single frame must be answered in a single frame, which means that the buffer containing the answers must be sent in one single frame. If the MAC answer's buffer length is greater than the maximum FOpt field, the device MUST send the buffer as FRMPayload on port 0. If the device has both application payload and MAC answers to send and both cannot fit in the frame, the MAC answers SHALL be sent in priority. If the length of the buffer is greater than the max FRMPayload size usable, the device SHALL clip the buffer to the max FRMPayload size before assembling the frame. Therefore the last MAC command answers may be truncated. In all cases the full list of MAC command is executed, even if the buffer containing the MAC answers must be clipped. The Network Server MUST NOT generate a sequence of MAC commands that may not be answered by the end-device in one single uplink. The Network Server SHALL compute the max FRMPayload size available for answering MAC commands as follow:

- If the latest uplink ADR bit is 0: The max payload size corresponding to the lowest data rate MUST be considered
- If the latest uplink ADR bit is set to 1: The max payload size corresponding to the data rate used for the last uplink of the device MUST be considered

Note: When receiving a clipped MAC answer the Network Server MAY retransmit the MAC commands that could not be answered

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x01	<i>ResetInd</i>	x		Used by an ABP device to indicate a reset to the network and negotiate protocol version
0x01	<i>ResetConf</i>		x	Acknowledges ResetInd command
0x02	<i>LinkCheckReq</i>	x		Used by an end-device to validate its connectivity to a network.
0x02	<i>LinkCheckAns</i>		x	Answer to LinkCheckReq command. Contains the received signal power estimation indicating to the end-device the quality of reception (link margin).
0x03	<i>LinkADRReq</i>		x	Requests the end-device to change data rate, transmit power, repetition rate or channel.
0x03	<i>LinkADRAns</i>	x		Acknowledges the LinkADRReq.
0x04	<i>DutyCycleReq</i>		x	Sets the maximum aggregated transmit duty-cycle of a device
0x04	<i>DutyCycleAns</i>	x		Acknowledges a DutyCycleReq command
0x05	<i>RXParamSetupReq</i>		x	Sets the reception slots parameters
0x05	<i>RXParamSetupAns</i>	x		Acknowledges a RXParamSetupReq command
0x06	<i>DevStatusReq</i>		x	Requests the status of the end-device
0x06	<i>DevStatusAns</i>	x		Returns the status of the end-device, namely its battery level and its demodulation margin
0x07	<i>NewChannelReq</i>		x	Creates or modifies the definition of a radio channel
0x07	<i>NewChannelAns</i>	x		Acknowledges a NewChannelReq command
0x08	<i>RXTimingSetupReq</i>		x	Sets the timing of the of the reception slots
0x08	<i>RXTimingSetupAns</i>	x		Acknowledges RXTimingSetupReq command
0x09	<i>TxParamSetupReq</i>		x	Used by the Network Server to set the maximum allowed dwell time and Max EIRP of end-device, based on local regulations
0x09	<i>TxParamSetupAns</i>	x		Acknowledges TxParamSetupReq command
0x0A	<i>DIChannelReq</i>		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	<i>DIChannelAns</i>	x		Acknowledges DIChannelReq command
0x0B	<i>RekeyInd</i>	x		Used by an OTA device to signal a security context update (rekeying)
0x0B	<i>RekeyConf</i>		x	Acknowledges RekeyInd command
0x0C	<i>ADRParamSetupReq</i>		x	Used by the Network Server to set the ADR_ACK_LIMT and ADR_ACK_DELAY parameters of an end-device
0x0C	<i>ADRParamSetupAns</i>	x		Acknowledges ADRParamSetupReq command
0x0D	<i>DeviceTimeReq</i>	x		Used by an end-device to request the current date and time
0x0D	<i>DeviceTimeAns</i>		x	Sent by the network, answer to the DeviceTimeReq request
0x0E	<i>ForceRejoinReq</i>		x	Sent by the network, ask the device to

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
				Rejoin immediately with optional periodic retries
0x0F	<i>RejoinParamSetupReq</i>		x	Used by the network to set periodic device Rejoin messages
0x0F	<i>RejoinParamSetupAns</i>	x		Acknowledges RejoinParamSetupReq
0x80 to 0xFF	Proprietary	x	x	Reserved for proprietary network command extensions

Table 4: MAC commands

Note: In general the end device will only reply one time to any Mac command received. If the answer is lost, the network has to send the command again. The network decides that the command must be resent when it receives a new uplink that doesn't contain the answer. Only the ***RxParamSetupReq***, ***RxTimingSetupReq*** and ***DiChannelReq*** have a different acknowledgment mechanism described in their relative section, because they impact the downlink parameters.

Note: When a MAC command is initiated by the end device, the network makes its best effort to 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 needs.

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

867

868 **5.1 Reset indication commands (*ResetInd*, *ResetConf*)**

869 This MAC command is only available to ABP devices activated on a LoRaWAN1.1
870 compatible Network Server. LoRaWAN1.0 servers do not implement this MAC command

871 OTA devices MUST NOT implement this command. The Network Server SHALL ignore the
872 **ResetInd** command coming from an OTA device.

873 With the **ResetInd** command, an ABP end-device indicates to the network that it has been
874 re-initialized and that it has switched back to its default MAC & radio parameters (i.e the
875 parameters originally programmed into the device at fabrication except for the three frame
876 counters). The **ResetInd** command MUST be added to the FOpt field of all uplinks until a
877 **ResetConf** is received.

878 This command does not signal to the Network Server that the downlink frame counters have
879 been reset. The frame counters (both uplink & downlink) SHALL NEVER be reset in ABP
880 devices.

881 Note: This command is meant for ABP devices whose power might be
882 interrupted at some point (example, battery replacement). The device
883 might lose the MAC layer context stored in RAM (except the Frame
884 Counters that must be stored in an NVM). In that case the device
885 needs a way to convey that context loss to the Network Server. In
886 future versions of the LoRaWAN protocol, that command may also be
887 used to negotiate some protocol options between the device and the
888 Network Server.

889 The **ResetInd** command includes the minor of the LoRaWAN version supported by the end
890 device.

891

Size (bytes)	1
ResetInd Payload	Dev LoRaWAN version

892

Figure 21 : ResetInd payload format

Size (bytes)	7:4	3:0
Dev LoRaWAN version	RFU	Minor=1

893

894

895 The minor field indicates the minor of the LoRaWAN version supported by the end-device.

Minor version	Minor
RFU	0
1 (LoRaWAN x.1)	1
RFU	2:15

When a **ResetInd** is received by the Network Server, it responds with a **ResetConf** command.

The ResetConf command contains a single byte payload encoding the LoRaWAN version supported by the Network Server using the same format than “dev LoRaWAN version”.

Size (bytes)	1
ResetConf Payload	Serv LoRaWAN version

Figure 22 : ResetConf payload format

The server’s version carried by the **ResetConf** must be the same than the device’s version. Any other value is invalid.

If the server’s version is invalid the device SHALL discard the **ResetConf** command and retransmit the **ResetInd** in the next uplink frame

5.2 Link Check commands (LinkCheckReq, LinkCheckAns)

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

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

Size (bytes)	1	1
LinkCheckAns Payload	Margin	GwCnt

Figure 23: LinkCheckAns payload format

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

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

5.3 Link ADR commands (LinkADRReq, LinkADRAAns)

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

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

Figure 24 : LinkADRReq payload format

Bits	[7:4]	[3:0]
------	-------	-------

DataRate_TXPower	DataRate	TXPower
-------------------------	----------	---------

929

930 The requested data rate (**DataRate**) and TX output power (**TXPower**) are region-specific
 931 and are encoded as indicated in [PHY]. The TX output power indicated in the command is to
 932 be considered the maximum transmit power the device may operate at. An end-device will
 933 acknowledge as successful a command which specifies a higher transmit power than it is
 934 capable of using and MUST, in that case, operate at its maximum possible power. A value
 935 0xF (15 in decimal format) of either DataRate or TXPower means that the device MUST
 936 ignore that field, and keep the current parameter value. The channel mask (**ChMask**)
 937 encodes the channels usable for uplink access as follows with bit 0 corresponding to the
 938 LSB:

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

939

Table 5: Channel state table

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

943

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

944 In the Redundancy bits the **NbTrans** field is the number of transmissions for each uplink
 945 message. This applies to “confirmed” and “unconfirmed” uplink frames. The default value is
 946 1 corresponding to a single transmission of each frame. The valid range is [1:15]. If
 947 **NbTrans**==0 is received the end-device SHALL keep the current NbTrans value unchanged.

948 The channel mask control (**ChMaskCntl**) field controls the interpretation of the previously
 949 defined **ChMask** bit mask. It controls the block of 16 channels to which the **ChMask** applies.
 950 It can also be used to globally turn on or off all channels using specific modulation. This field
 951 usage is region specific and is defined in [PHY].

952 The Network Server may include multiple contiguous LinkADRReq commands within a
 953 single downlink message. For the purpose of configuring the end-device channel mask, the
 954 end-device MUST process all contiguous LinkADRReq messages, in the order present in
 955 the downlink message, as a single atomic block command. The Network Server MUST NOT
 956 include more than one such atomic block command in a downlink message. The end-device
 957 MUST send a single LinkADRAns command to accept or reject an entire ADR atomic
 958 command block. If the downlink message carries more than one ADR atomic command
 959 block, the end-device SHALL process only the first one and send a NACK (a LinkADRAns
 960 command with all Status bits set to 0) in response to all other ADR command block. The
 961 device MUST only process the DataRate, TXPower and NbTrans from the last LinkADRReq
 962 command in the contiguous ADR command block, as these settings govern the end-device
 963 global state for these values. The Channel mask ACK bit of the response MUST reflect the
 964 acceptance/rejection of the final channel plan after in-order-processing of **all** the Channel
 965 Mask Controls in the contiguous ADR command block.

966 The channel frequencies are region-specific and they are defined [PHY]. An end-device
 967 answers to a **LinkADRReq** with a **LinkADRAns** command.

968

969

Size (bytes)	1
---------------------	---

LinkADRAAns Payload Status

Figure 25 : LinkADRAAns payload format

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

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

	Bit = 0	Bit = 1
Channel mask ACK	The channel mask sent enables a yet undefined channel or the channel mask required all channels to be disabled. The command was discarded and the end-device state was not changed.	The channel mask sent was successfully interpreted. All currently defined channel states were set according to the mask.
Data rate ACK	The data rate requested is unknown to the end-device or is not possible given the channel mask provided (not supported by any of the enabled channels). The command was discarded and the end-device state was not changed.	The data rate was successfully set or the DataRate field of the request was set to 15, meaning it was ignored
Power ACK	The device is unable to operate at or below the requested power level. The command was discarded and the end-device state was not changed.	The device is able to operate at or below the requested power level, or the TXPower field of the request was set to 15, meaning it shall be ignored

Table 6: LinkADRAAns status bits signification

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

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

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

Size (bytes)	1
DutyCycleReq Payload	DutyCyclePL

Figure 26 : DutyCycleReq payload format

Bits	7:4	3:0
DutyCyclePL	RFU	MaxDCycle

The maximum end-device transmit duty cycle allowed is:

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

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

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

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

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

Size (bytes)	1	3
RXParamSetupReq Payload	DLsettings	Frequency

Figure 27 : RXParamSetupReq payload 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 reception slot (RX1). As a default this offset is 0. The offset is used to take into account maximum power density constraints for base stations in some regions and to balance the uplink and downlink radio link margins.

The data rate (**RX2DataRate**) field defines the data rate of a downlink using the second receive window following the same convention as the **LinkADRReq** command (0 means DR0/125kHz for example). The frequency (**Frequency**) field corresponds to the frequency of the channel used for the second receive window, whereby the frequency is coded following the convention defined in the **NewChannelReq** command.

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

1017 The payload contains a single status byte.

Size (bytes)	1
RXParamSetupAns Payload	Status

1018 **Figure 28 : RXParamSetupAns payload format**

1019 The status (**Status**) bits have the following meaning.

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

	Bit = 0	Bit = 1
Channel ACK	The frequency requested is not usable by the end-device.	RX2 slot channel was successfully set
RX2DataRate ACK	The data rate requested is unknown to the end-device.	RX2 slot data rate was successfully set
RX1DRoffset ACK	the uplink/downlink data rate offset for RX1 slot is not in the allowed range	RX1DRoffset was successfully set

1021 **Table 7: RXParamSetupAns status bits signification**

1022 If either of the 3 bits is equal to 0, the command did not succeed and the previous
1023 parameters MUST be kept.

1024

1025 5.6 End-Device Status (DevStatusReq, DevStatusAns)

1026 With the **DevStatusReq** command a Network Server may request status information from
1027 an end-device. The command has no payload. If a **DevStatusReq** is received by an end-
1028 device, it MUST respond with a **DevStatusAns** command.

1029

Size (bytes)	1	1
DevStatusAns Payload	Battery	Margin

1030 **Figure 29 : DevStatusAns payload format**

1031 The battery level (**Battery**) reported is encoded as follows:

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

1032 **Table 8: Battery level decoding**

1033 The margin (**Margin**) is the demodulation signal-to-noise ratio in dB rounded to the nearest
1034 integer value for the last successfully received **DevStatusReq** command. It is a signed
1035 integer of 6 bits with a minimum value of -32 and a maximum value of 31.

Bits	7:6	5:0
Status	RFU	Margin

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

Devices operating in region where a fixed channel plan is defined shall not implement these MAC commands. The commands SHALL not be answered by the device. Please refer to [PHY] for applicable regions.

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

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

Figure 30 : **NewChannelReq** payload format

The channel index (**ChIndex**) is the index of the channel being created or modified. Depending on the region and frequency band used, in certain regions ([PHY]) the LoRaWAN specification imposes default channels which must be common to all devices and cannot 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. A device must be able to handle at least 16 different channel definitions. In certain regions the device may have to store more than 16 channel definitions.

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

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

Bits	7:4	3:0
DrRange	MaxDR	MinDR

Following the convention defined in Section 5.3 the minimum data rate (**MinDR**) subfield designate the lowest uplink data rate allowed on this channel. For example using European regional parameters, 0 designates DR0 / 125 kHz. Similarly, the maximum data rate (**MaxDR**) designates the highest uplink data rate. For example, **DrRange** = 0x77 means that only 50 kbps GFSK is allowed on a channel and **DrRange** = 0x50 means that DR0 / 125 kHz to DR5 / 125 kHz are supported.

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

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

Size (bytes)	1
NewChannelAns Payload	Status

Figure 31 : **NewChannelAns** payload format

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

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

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

Table 9: NewChannelAns status bits signification

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

The **DIChannelReq** command allows the network to associate a different downlink frequency to the RX1 slot. This command is applicable for all the physical layer specifications supporting the **NewChannelReq** command (for example EU and China physical layers, but not for US or Australia).

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

Size (bytes)	1	3
DIChannelReq Payload	ChIndex	Freq

Figure 32 : DLChannelReq payload format

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

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

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

The payload of this message contains the following information:

Size (bytes)	1
DIChannelAns Payload	Status

Figure 33 : DLChannelAns payload format

1110 The status (**Status**) bits have the following meaning:

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

1111

	Bit = 0	Bit = 1
Channel frequency ok	The device cannot use this frequency	The device is able to use this frequency.
Uplink frequency exists	The uplink frequency is not defined for this channel , the downlink frequency can only be set for a channel that already has a valid uplink frequency	The uplink frequency of the channel is valid

1112

Table 10: DIChannelAns status bits signification

1113

1114 5.8 Setting delay between TX and RX (*RXTimingSetupReq*, 1115 *RXTimingSetupAns*)

1116 The *RXTimingSetupReq* command allows configuring the delay between the end of the TX
1117 uplink and the opening of the first reception slot. The second reception slot opens one
1118 second after the first reception slot.

1119

Size (bytes)	1
RXTimingSetupReq Payload	Settings

1120

Figure 34 : RXTimingSetupReq payload format

1121

1122 The delay (**Delay**) field specifies the delay. The field is split in two 4-bit indexes:

Bits	7:4	3:0
Settings	RFU	Del

1123

1124 The delay is expressed in seconds. **Del** 0 is mapped on 1 s.

1125

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

1126

Table 11: RXTimingSetup Delay mapping table

1127

1128 An end device answers *RXTimingSetupReq* with *RXTimingSetupAns* with no payload.

1129 The *RXTimingSetupAns* command should be added in the FOpt field of all uplinks until a
1130 class A downlink is received by the end-device. This guarantees that even in presence of

uplink packet loss, the network is always aware of the downlink parameters used by the end-device.

1133

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

This MAC command only needs to be implemented for compliance in certain regulatory regions. Please refer to [PHY].

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

1141

Size (bytes)	1
<i>TxParamSetupReq</i> payload	EIRP_DwellTime

1143

1144

Figure 35 : *TxParamSetupReq* payload format

The structure of EIRP_DwellTime field is described below:

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

1146

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

1150

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

1151

Table 12 : *TxParamSetup* EIRP encoding table

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

1154

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

1156

Coded Value	Dwell Time
0	No Limit
1	400 ms

1157

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

1158

1159

1160

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

1161

1162

1163

1164 **5.10 Rekey indication commands (*RekeyInd*, *RekeyConf*)**

1165 This MAC command is only available to OTA devices activated on a LoRaWAN1.1
 1166 compatible Network Server. LoRaWAN1.0 servers do not implement this MAC command.

1167 ABP devices MUST NOT implement this command. The Network Server SHALL ignore the
 1168 **RekeyInd** command coming from an ABP device.

1169 For OTA devices the **RekeyInd** MAC command is used to confirm security key update and
 1170 in future versions of LoRaWAN (>1.1) to negotiate the minor LoRaWAN protocol version
 1171 running between the end-device and the Network Server. The command does not signal a
 1172 reset of the MAC & radio parameters (see 6.2.3).

1173 The **RekeyInd** command includes the minor of the LoRaWAN version supported by the end
 1174 device.

1175

Size (bytes)	1
RekeyInd Payload	Dev LoRaWAN version

1176

Figure 36 : RekeyInd payload format

1177

Size (bytes)	7:4	3:0
Dev LoRaWAN version	RFU	Minor=1

1178

1179

1180 The minor field indicates the minor of the LoRaWAN version supported by the end-device.

1181

Minor version	Minor
RFU	0
1 (LoRaWAN x.1)	1
RFU	2:15

1182

1183 OTA devices SHALL send the **RekeyInd** in all confirmed & unconfirmed uplink frames
 1184 following the successful processing of a Join-accept (new session keys have been derived)
 1185 until a **RekeyConf** is received. If the device has not received a **RekeyConf** within the first
 1186 ADR_ACK_LIMIT uplinks it SHALL revert to the Join state. **RekeyInd** commands sent by
 1187 such devices at any later time SHALL be discarded by the Network Server. The Network
 1188 Server SHALL discard any uplink frames protected with the new security context that are
 1189 received after the transmission of the **Join-accept** and before the first uplink frame that
 1190 carries a **RekeyInd** command.

1191 When a **RekeyInd** is received by the Network Server, it responds with a **RekeyConf**
 1192 command.

1193 The RekeyConf command contains a single byte payload encoding the LoRaWAN version
 1194 supported by the Network Server using the same format than “dev LoRaWAN version”.

1195

1196

Size (bytes)	1
RekeyConf Payload	Serv LoRaWAN version

1197

Figure 37 : RekeyConf payload format

1198 The server version must be greater than 0 (0 is not allowed), and smaller or equal (\leq) to the
1199 device's LoRaWAN version. Therefore for a LoRaWAN1.1 device the only valid value is 1. If
1200 the server's version is invalid the device SHALL discard the **RekeyConf** command and
1201 retransmit the **RekeyInd** in the next uplink frame
1202

1203 5.11 ADR parameters (ADRParamSetupReq, ADRParamSetupAns)

1204 The **ADRParamSetupReq** command allows changing the ADR_ACK_LIMIT and
1205 ADR_ACK_DELAY parameters defining the ADR back-off algorithm. The
1206 ADRParamSetupReq command has a single byte payload.
1207

Size (bytes)	1
ADRParamSetupReq Payload	ADRparam

1208

Figure 38 : ADRParamSetupReq payload format

Bits	7:4	3:0
ADRparam	Limit_exp	Delay_exp

1209

1210 The Limit_exp field sets the ADR_ACK_LIMIT parameter value:
1211 $ADR_ACK_LIMIT = 2^{Limit_exp}$
1212

1213 The Limit_exp valid range is 0 to 15, corresponding to a range of 1 to 32768 for
1214 ADR_ACK_LIMIT
1215

1216 The Delay_exp field sets the ADR_ACK_DELAY parameter value.
1217 $ADR_ACK_DELAY = 2^{Delay_exp}$
1218

1219 The Delay_exp valid range is 0 to 15, corresponding to a range of 1 to 32768 for
1220 ADR_ACK_DELAY
1221

1222 The **ADRParamSetupAns** command is used by the end-device to acknowledge the
1223 reception of **ADRParamSetupReq** command. The **ADRParamSetupAns** command has no
1224 payload field.
1225

5.12 DeviceTime commands (*DeviceTimeReq*, *DeviceTimeAns*)

This MAC command is only available if the device is activated on a LoRaWAN1.1 compatible Network Server. LoRaWAN1.0 servers do not implement this MAC command.

With the ***DeviceTimeReq*** command, an end-device may request from the network the current network date and time. The request has no payload.

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

Size (bytes)	4	1
DeviceTimeAns Payload	32-bit unsigned integer : Seconds since epoch*	8bits unsigned integer: fractional-second in $\frac{1}{2^8}$ second steps

Figure 39 : *DeviceTimeAns* payload format

The time provided by the network MUST have a worst case accuracy of +/-100mSec.

(*) The GPS epoch (i.e Sunday January the 6th 1980 at midnight) is used as origin. The “seconds” field is the number of seconds elapsed since the origin. This field is monotonically increasing by 1 every second. To convert this field to UTC time, the leap seconds must be taken into account.

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

5.13 Force Rejoin Command (*ForceRejoinReq*)

With the Force Rejoin command, the network asks a device to immediately transmit a Rejoin-Request Type 0 or type 2 message with a programmable number of retries, periodicity and data rate. This RejoinReq uplink may be used by the network to immediately rekey a device or initiate a handover roaming procedure.

The command has two bytes of payload.

Bits	15:14	13:11	10:8	7	6:4	3:0
ForceRejoinReq bits	RFU	Period	Max_Retries	RFU	RejoinType	DR

Figure 40 : *ForceRejoinReq* payload format

1255

1256 The parameters are encoded as follow:

1257 Period: The delay between retransmissions SHALL be equal to 32 seconds $\times 2^{\text{Period}}$ +
1258 Rand32, where Rand32 is a pseudo-random number in the [0:32] range.

1259 Max_Retries: The total number of times the device will retry the Rejoin-request.

- 1260
- 0 : the Rejoin is sent only once (no retry)
- 1261
- 1 : the Rejoin MUST be sent 2 times in total (1 + 1 retry)
- 1262
- ...
- 1263
- 7: the Rejoin MUST be sent 8 times (1 + 7 retries)

1264 RejoinType: This field specifies the type of Rejoin-request that shall be transmitted by the
1265 device.

- 1266
- 0 or 1 : A Rejoin-request type 0 shall be transmitted
- 1267
- 2 : A Rejoin-request type 2 shall be transmitted
- 1268
- 3 to 7 : RFU

1269 DR: The Rejoin-request frame SHALL be transmitted using the data rate DR. The
1270 correspondence between the actual physical modulation data rate and the DR value follows
1271 the same convention as the **LinkADRReq** command and is defined for each region in [PHY]

1272 The command has no answer, as the device MUST send a Rejoin-Request when receiving
1273 the command. The first transmission of a RejoinReq message SHALL be done immediately
1274 after the reception of the command (but the network may not receive it). If the device
1275 receives a new **ForceRejoinReq** command before it has reached the number of
1276 transmission retries, the device SHALL resume transmission of RejoinReq with the new
1277 parameters.
1278

1279 5.14 RejoinParamSetupReq (RejoinParamSetupAns)

1280 With the RejoinParamSetupReq command, the network may request the device to
1281 periodically send a RejoinReq Type 0 message with a programmable periodicity defined as
1282 a time or a number of uplinks.

1283 Both time and count are proposed to cope with devices which may not have time
1284 measurement capability. The periodicity specified sets the maximum time or number of
1285 uplink between two RejoinReq transmissions. The device MAY send RejoinReq more
1286 frequently.
1287

1288 The command has a single byte payload.

Bits	7:4	3:0
RejoinParamSetupReq bits	MaxTimeN	MaxCountN

1289

Figure 41 : RejoinParamSetupReq payload format

1290 The parameters are defined as follow:

1291

1292 MaxCountN = C = 0 to 15. The device MUST send a Rejoin-request type 0 at least every
1293 2^{C+4} uplink messages.

1294 MaxTimeN = T = 0 to 15; the device MUST send a Rejoin-request type 0 at least every 2^{T+10}
1295 seconds.

1296 • T = 0 corresponds to roughly 17 minutes

1297 • T = 15 is about 1 year

1298

1299 A RejoinReq packet is sent every time one of the 2 conditions (frame Count or Time) is met.

1300 The device MUST implement the uplink count periodicity. Time based periodicity is
1301 OPTIONAL. A device that cannot implement time limitation MUST signal it in the answer

1302 The answer has a single byte payload.

Bits	Bits 7:1	Bit 0
Status bits	RFU	TimeOK

1303 **Figure 42 : RejoinParamSetupAns payload format**

1304 If Bit 0 = 1, the device has accepted Time and Count limitations, otherwise it only accepts
1305 the count limitation.

1306

1307 Note: For devices that have a very low message rate and no time
1308 measurement capability, the mechanism to agree on the optimal count
1309 limitation is not specified in LoRaWAN.

1310 6 End-Device Activation

1311 To participate in a LoRaWAN network, each end-device has to be personalized and
1312 activated.

1313 Activation of an end-device can be achieved in two ways, either via **Over-The-Air**
1314 **Activation** (OTAA) or via **Activation By Personalization** (ABP)

1315 6.1 Data Stored in the End-device

1316 6.1.1 Before Activation

1317 6.1.1.1 JoinEUI

1318 The **JoinEUI** is a global application ID in IEEE EUI64 address space that uniquely identifies
1319 the Join Server that is able to assist in the processing of the Join procedure and the session
1320 keys derivation.

1321 For OTAA devices, the **JoinEUI** MUST be stored in the end-device before the Join
1322 procedure is executed. The **JoinEUI** is not required for ABP only end-devices

1323 6.1.1.2 DevEUI

1324 The **DevEUI** is a global end-device ID in IEEE EUI64 address space that uniquely identifies
1325 the end-device.

1326 DevEUI is the recommended unique device identifier by Network Server(s), whatever
1327 activation procedure is used, to identify a device roaming across networks.

1328 For OTAA devices, the **DevEUI** MUST be stored in the end-device before the Join
1329 procedure is executed. ABP devices do not need the DevEUI to be stored in the device
1330 itself, but it is RECOMMENDED to do so.

1331 | **Note:** It is a recommended practice that the DevEUI should also be
1332 | available on a device label, for device administration.

1333 6.1.1.3 Device root keys (AppKey & NwkKey)

1334 The NwkKey and AppKey are AES-128 root keys specific to the end-device that are
 1335 assigned to the end-device during fabrication.¹ Whenever an end-device joins a network via
 1336 over-the-air activation, the NwkKey is used to derive the FNwkSIntKey , SNwkSIntKey and
 1337 NwkSEncKey session keys, and AppKey is used to derive the AppSKey session key

1338

1339 **Note:** When working with a v1.1 Network Server, the application
 1340 session key is derived only from the AppKey, therefore the NwkKey
 1341 may be surrendered to the network operator to manage the JOIN
 1342 procedure without enabling the operator to eavesdrop on the
 1343 application payload data.

1344 Secure provisioning, storage, and usage of root keys NwkKey and AppKey on the end-
 1345 device and the backend are intrinsic to the overall security of the solution. These are left to
 1346 implementation and out of scope of this document. However, elements of this solution may
 1347 include SE (Secure Elements) and HSM (Hardware Security Modules).

1348 To ensure backward compatibility with LoRaWAN 1.0 and earlier Network Servers that do not
 1349 support two root keys, the end-device MUST default back to the single root key scheme
 1350 when joining such a network. In that case only the root NwkKey is used. This condition is
 1351 signaled to the end-device by the “OptNeg” bit (bit 7) of the DLsetting field of the Join-accept
 1352 message being zero. The end-device in this case MUST

- 1353 • Use the NwkKey to derive both the AppSKey and the FNwkSIntKey session keys as
 1354 in LoRaWAN1.0 specification.
- 1355 • Set the SNwkSIntKey & NwkSEncKey equal to FNwkSIntKey, the same network
 1356 session key is effectively used for both uplink and downlink MIC calculation and
 1357 encryption of MAC payloads according to the LoRaWAN1.0 specification.

1358

1359 A NwkKey MUST be stored on an end-device intending to use the OTAA procedure.

1360 A NwkKey is not required for ABP only end-devices.

1361 An AppKey MUST be stored on an end-device intending to use the OTAA procedure.

1362 An Appkey is not required for ABP only end-devices.

1363 Both the NwkKey and AppKey SHOULD be stored in a way that prevents extraction and re-
 1364 use by malicious actors.

1365

1366 6.1.1.4 JSIntKey and JSEncKey derivation

1367 For OTA devices two specific lifetime keys are derived from the NwkKey root key:

- 1368 • JSIntKey is used to MIC Rejoin-Request type 1 messages and Join-Accept answers
- 1369 • JSEncKey is used to encrypt the Join-Accept triggered by a Rejoin-Request

1. Since all end-devices are equipped with unique application and network root keys specific for each end-device, extracting the AppKey/NwkKey from an end-device only compromises this one end-device.

1370
1371
1372 JSIntKey = aes128_encrypt(NwkKey, 0x06 | DevEUI | pad₁₆)
1373 JSEncKey = aes128_encrypt(NwkKey, 0x05 | DevEUI | pad₁₆)
1374

1375 6.1.2 After Activation

1376 After activation, the following additional informations are stored in the end-device: a device
1377 address (**DevAddr**), a triplet of network session key (**NwkSEncKey**/ **SNwkSIntKey**/
1378 **FNwkSIntKey**), and an application session key (**AppSKey**).

1379 6.1.2.1 End-device address (DevAddr)

1380 The **DevAddr** consists of 32 bits and identifies the end-device within the current network.
1381 The DevAddr is allocated by the Network Server of the end-device.
1382 Its format is as follows:

1383

Bit#	[31..32-N]	[31-N..0]
DevAddr bits	AddrPrefix	NwkAddr

1384

Figure 43 : DevAddr fields

1385

1386 Where N is an integer in the [7:24] range.

1387

1388 The LoRaWAN protocol supports various network address types with different network
1389 address space sizes. The variable size AddrPrefix field is derived from the Network Server's
1390 unique identifier **NetID** (see 6.2.3) allocated by the LoRa Alliance with the exception of the
1391 AddrPrefix values reserved for experimental/private network. The AddrPrefix field enables
1392 the discovery of the Network Server currently managing the end-device during roaming.
1393 Devices that do not respect this rule cannot roam between two networks because their home
1394 Network Server cannot be found.

1395 The least significant (32-N) bits, the network address (NwkAddr) of the end-device, can be
1396 arbitrarily assigned by the network manager.

1397 The following AddrPrefix values may be used by any private/experimental network and will
1398 not be allocated by the LoRa Alliance.

1399

Private/experimental network reserved AddrPrefix
N = 7
AddrPrefix = 7'b0000000 or AddrPrefix = 7'b0000001
NwkAddr = 25bits freely allocated by the network manager

1400

1401 Please refer to [BACKEND] for the exact construction of the AddrPrefix field and the
1402 definition of the various address classes.

1403

1404 6.1.2.2 Forwarding Network session integrity key (FNwkSIntKey)

1405 The FNwkSIntKey is a network session key specific for the end-device. It is used by the end-
1406 device to calculate the MIC or part of the MIC (message integrity code) of all uplink data
1407 messages to ensure data integrity as specified in 4.4.

1408 The FNwkSIntKey SHOULD be stored in a way that prevents extraction and re-use by
1409 malicious actors.

1410

1411 6.1.2.3 Serving Network session integrity key (SNwkSIntKey)

1412 The **SNwkSIntKey** is a network session key specific for the end-device. It is used by the
1413 end-device to verify the **MIC** (message integrity code) of all downlink data messages to
1414 ensure data integrity and to compute half of the uplink messages MIC.

1415 Note: The uplink MIC calculation relies on two keys (FNwkSIntKey and
1416 SNwkSIntKey) in order to allow a forwarding Network Server in a
1417 roaming setup to be able to verify only half of the MIC field

1418 When a device connects to a LoRaWAN1.0 Network Server the same key is used for both
1419 uplink & downlink MIC calculation as specified in 4.4. In that case **SNwkSIntKey** takes the
1420 same value than **FNwkSIntKey**.

1421 The **SNwkSIntKey** SHOULD be stored in a way that prevents extraction and re-use by
1422 malicious actors.

1423

1424 6.1.2.4 Network session encryption key (NwkSEncKey)

1425 The NwkSEncKey is a network session key specific to the end-device. It is used to encrypt &
1426 decrypt uplink & downlink MAC commands transmitted as payload on port 0 or in the FOpt
1427 field. When a device connects to a LoRaWAN1.0 Network Server the same key is used for
1428 both MAC payload encryption and MIC calculation. In that case **NwkSEncKey** takes the
1429 same value than **FNwkSIntKey**.

1430 The NwkSEncKey SHOULD be stored in a way that prevents extraction and re-use by
1431 malicious actors.

1432 6.1.2.5 Application session key (AppSKey)

1433 The **AppSKey** is an **application session key** specific for the end-device. It is used by both
1434 the application server and the end-device to encrypt and decrypt the payload field of
1435 application-specific data messages. Application payloads are end-to-end encrypted between
1436 the end-device and the application server, but they are integrity protected only in a hop-by-
1437 hop fashion: one hop between the end-device and the Network Server, and the other hop
1438 between the Network Server and the application server. That means, a malicious Network
1439 Server may be able to alter the content of the data messages in transit, which may even
1440 help the Network Server to infer some information about the data by observing the reaction
1441 of the application end-points to the altered data. Network Servers are considered as trusted,
1442 but applications wishing to implement end-to-end confidentiality and integrity protection MAY
1443 use additional end-to-end security solutions, which are beyond the scope of this
1444 specification.

1445 The **AppSKey** SHOULD be stored in a way that prevents extraction and re-use by malicious
1446 actors.

1447

1448 6.1.2.6 Session Context

1449 Session Context contains Network Session and Application Session.

1450

1451 The Network Session consists of the following state:

1452

- 1453 • F/SNwkSIntKey
- 1454 • NwkSEncKey
- 1455 • FCntUp
- 1456 • FCntDwn (LW 1.0) or NFCntDwn (LW 1.1)
- 1457 • DevAddr

1458

1459 The Application Session consists of the following state:

1460

- 1461 • AppSKey
- 1462 • FCntUp
- 1463 • FCntDown (LW 1.0) or AFCntDwn (LW 1.1)

1464

1465 Network Session state is maintained by the NS and the end-device. Application Session
1466 state is maintained by the AS and the end-device.

1467

1468 Upon completion of either the OTAA or ABP procedure, a new security session context has
1469 been established between the NS/AS and the end-device. Keys and the end-device address
1470 are fixed for the duration of a session (FNwkSIntKey, SNwkSIntKey, AppSKey, DevAddr).
1471 Frame counters increment as frame traffic is exchanged during the session (FCntUp,
1472 FCntDwn, NFCntDwn, AFCntDwn).

1473

For OTAA devices, Frame counters MUST NOT be re-used for a given key, therefore new Session Context MUST be established well before saturation of a frame counter.

It is RECOMMENDED that session state be maintained across power cycling of an end-device. Failure to do so for OTAA devices means the activation procedure will need to be executed on each power cycling of a device.

6.2 Over-the-Air Activation

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

As discussed above, the join procedure requires the end-device to be personalized with the following information before it starts the join procedure: a DevEUI, JoinEUI, NwkKey and AppKey.

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

6.2.1 Join procedure

From an end-device's point of view, the join procedure consists of either a **join or rejoin-request** and a **Join-accept** exchange.

6.2.2 Join-request message

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

Size (bytes)	8	8	2
Join-request	JoinEUI	DevEUI	DevNonce

Figure 44 : Join-request message fields

The join-request message 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 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 (stored in a non-volatile memory). Resetting DevNonce without changing JoinEUI will cause the Network Server to discard the Join-requests of the device. For each end-device, the

1512 Network Server keeps track of the last **DevNonce** value used by the end-device, and
1513 ignores Join-requests if DevNonce is not incremented.

1514

1515 **Note:** This mechanism prevents replay attacks by sending previously
1516 recorded join-request messages with the intention of disconnecting the
1517 respective end-device from the network. Any time the Network Server
1518 processes a Join-Request and generates a Join-accept frame, it shall
1519 maintain both the old security context (keys and counters, if any) and
1520 the new one until it receives the first successful uplink frame containing
1521 the **RekeyInd** command using the new context, after which the old
1522 context can be safely removed.

1523 The message integrity code (**MIC**) value (see Chapter 4 for MAC message description) for a
1524 join-request message is calculated as follows:¹

1525

1526 $cmac = \text{aes128_cmac}(\text{NwkKey}, \text{MHDR} \mid \text{JoinEUI} \mid \text{DevEUI} \mid \text{DevNonce})$

1527 $\text{MIC} = cmac[0..3]$

1528 The join-request message is not encrypted. The join-request message can be transmitted
1529 using any data rate and following a random frequency hopping sequence across the
1530 specified join channels. It is RECOMMENDED to use a plurality of data rates. The intervals
1531 between transmissions of **Join-Requests** SHALL respect the condition described in chapter
1532 7. For each transmission of a Join-request, the end-device SHALL increment the DevNonce
1533 value.

1534 6.2.3 Join-accept message

1535 The Network Server will respond to the **join** or **rejoin-request** message with a **join-accept**
1536 message if the end-device is permitted to join a network. The join-accept message is sent
1537 like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or
1538 JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2,
1539 respectively). The channel frequency and data rate used for these two receive windows are
1540 identical to the one used for the RX1 and RX2 receive windows described in the “receive
1541 windows” section of [PHY]

1542 No response is given to the end-device if the Join-request is not accepted.

1543 The join-accept message contains a server nonce (**JoinNonce**) of 3 octets, a network
1544 identifier (**NetID**), an end-device address (**DevAddr**), a (**DLSettings**) field providing some of
1545 the downlink parameters, the delay between TX and RX (**RxDelay**) and an optional list of
1546 network parameters (**CFList**) for the network the end-device is joining. The optional CFList
1547 field is region specific and is defined in [PHY].

1548

Size (bytes)	3	3	4	1	1	(16) Optional
Join-accept	JoinNonce	Home_NetID	DevAddr	DLSettings	RxDelay	CFList

1549

Figure 45 : Join-accept message fields

1550 The **JoinNonce** is a device specific counter value (that never repeats itself) provided by the
1551 Join Server and used by the end-device to derive the session keys **FNwkSIntKey**,

¹ [RFC4493]

1552 **SNwkSIntKey**, **NwkSEncKey** and **AppSKey**. JoinNonce is incremented with every Join-
1553 accept message.

1554 The device keeps track of the JoinNonce value used in the last successfully processed Join-
1555 accept (corresponding to the last successful key derivation). The device SHALL accept the
1556 Join-accept only if the MIC field is correct and the JoinNonce is strictly greater than the
1557 recorded one. In that case the new JoinNonce value replaces the previously stored one.

1558 If the device is susceptible of being power cycled the JoinNonce SHALL be persistent
1559 (stored in a non-volatile memory).

1560 The LoRa Alliance allocates a 24bits unique network identifier (**NetID**) to all networks with
1561 the exception of the following **NetID** values reserved for experimental/private networks that
1562 are left unmanaged.

1563 There are 2^{15} Private /Experimental network reserved NetID values built as follow:

Nb bits	3	14	7
	3'b000	XXXXXXXXXXXXXXXX Arbitrary 14bit value	7'b00000000 Or 7'b00000001

1564

1565 The **home_NetID** field of the Join-accept frame corresponds to the **NetId** of the device's
1566 home network.

1567 The network that assigns the devAddr and the home network may be different in a roaming
1568 scenario. For more precision please refer to [BACKEND].

1569 The **DLsettings** field contains the downlink configuration:

1570

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

1571

1572 The OptNeg bit indicates whether the Network Server implements the LoRaWAN1.0 protocol
1573 version (unset) or 1.1 and later (set). When the OptNeg bit is set

- 1574 • The protocol version is further (1.1 or later) negotiated between the end-device and
1575 the Network Server through the *RekeyInd/RekeyConf* MAC command exchange.
- 1576 • The device derives **FNwkSIntKey** & **SNwkSIntKey** & **NwkSEncKey** from the
1577 **NwkKey**
- 1578 • The device derives **AppSKey** from the **AppKey**

1579

1580 When the OptNeg bit is not set

- 1581 • The device reverts to LoRaWAN1.0 , no options can be negotiated
- 1582 • The **RekeyInd** command is not sent by the device
- 1583 • The device derives **FNwkSIntKey** & **AppSKey** from the **NwkKey**
- 1584 • The device sets **SNwkSIntKey** & **NwkSEncKey** equal to **FNwkSIntKey**

1585

1586 The 4 session keys **FNwkSIntKey**, **SNwkSIntKey**, **NwkSEncKey** and **AppSKey** are
1587 derived as follows:
1588

1589 If the OptNeg is unset, the session keys are derived from the NwkKey as follow:
1590 AppSKey = aes128_encrypt(NwkKey, 0x02 | JoinNonce | NetID | DevNonce | pad₁₆)¹
1591 FNwkSIntKey = aes128_encrypt(NwkKey, 0x01 | JoinNonce | NetID | DevNonce | pad₁₆)
1592 SNwkSIntKey = NwkSEncKey = FNwkSIntKey.

1593
1594 The MIC value of the join-accept message is calculated as follows:²
1595 $cmac = \text{aes128_cmac}(\mathbf{NwkKey}, \text{MHDR} | \text{JoinNonce} | \text{NetID} | \text{DevAddr} | \text{DLSettings} |$
1596 $\text{RxDelay} | \text{CFList})$
1597 $\text{MIC} = cmac[0..3]$

1598
1599
1600 Else if the OptNeg is set, the AppSKey is derived from AppKey as follow:
1601 AppSKey = aes128_encrypt(AppKey, 0x02 | JoinNonce | JoinEUI | DevNonce | pad₁₆)
1602
1603 And the network session keys are derived from the NwkKey:
1604 FNwkSIntKey = aes128_encrypt(NwkKey, 0x01 | JoinNonce | JoinEUI | DevNonce | pad₁₆)
1605 SNwkSIntKey = aes128_encrypt(NwkKey, 0x03 | JoinNonce | JoinEUI | DevNonce | pad₁₆)
1606 NwkSEncKey = aes128_encrypt(NwkKey, 0x04 | JoinNonce | JoinEUI | DevNonce | pad₁₆)

1607
1608 In this case the MIC value is calculated as follows:³
1609 $cmac = \text{aes128_cmac}(\mathbf{JSIntKey},$
1610 $\text{JoinReqType} | \text{JoinEUI} | \text{DevNonce} | \text{MHDR} | \text{JoinNonce} | \text{NetID} | \text{DevAddr} |$
1611 $\text{DLSettings} | \text{RxDelay} | \text{CFList})$
1612 $\text{MIC} = cmac[0..3]$
1613
1614 JoinReqType is a single byte field encoding the type of Join-request or Rejoin-request that
1615 triggered the Join-accept response.

Join-request or Rejoin-request type	JoinReqType value
Join-request	0xFF
Rejoin-request type 0	0x00
Rejoin-request type 1	0x01
Rejoin-request type 2	0x02

Table 13 : JoinReqType values

1616
1617 The key used to encrypt the Join-Accept message is a function of the Join or ReJoin-
1618 Request message that triggered it.
1619

Triggering Join-request or Rejoin-request type	Join-accept Encryption Key
Join-request	NwkKey
Rejoin-request type 0 or 1 or 2	JSEncKey

Table 14 : Join-Accept encryption key

1620
1621 The Join-Accept message is encrypted as follows:
1622 aes128_decrypt(**NwkKey** or **JSEncKey**, JoinNonce | NetID | DevAddr | DLSettings |
1623 RxDelay | CFList | MIC).
1624

¹ The pad₁₆ function appends zero octets so that the length of the data is a multiple of 16

² [RFC4493]

³ [RFC4493]

1625 The message is either 16 or 32 bytes long.

1626 **Note:** AES decrypt operation in ECB mode is used to encrypt the join-
1627 accept message so that the end-device can use an AES encrypt
1628 operation to decrypt the message. This way an end-device only has to
1629 implement AES encrypt but not AES decrypt.

1630 **Note:** Establishing these four session keys allows for a federated
1631 Network Server infrastructure in which network operators are not able
1632 to eavesdrop on application data. The application provider commits to
1633 the network operator that it will take the charges for any traffic incurred
1634 by the end-device and retains full control over the AppSKey used for
1635 protecting its application data.

1636 **Note:** The device's protocol version (1.0 or 1.1) is registered on the
1637 backend side out-of-band at the same time than the DevEUI and the
1638 device's NwkKey and possibly AppKey
1639

1640 The RX1DROffset field sets the offset between the uplink data rate and the downlink data
1641 rate used to communicate with the end-device on the first reception slot (RX1). By default
1642 this offset is 0. The offset is used to take into account maximum power density constraints
1643 for base stations in some regions and to balance the uplink and downlink radio link margins.

1644 The actual relationship between the uplink and downlink data rate is region specific and
1645 detailed in [PHY]

1646 The delay **RxDelay** follows the same convention as the **Delay** field in the
1647 **RXTimingSetupReq** command.

1648 If the Join-accept message is received following the transmission of:

- 1649 • A Join-Request or a Rejoin-request Type 0 or 1 and if the CList field is absent, the
1650 device SHALL revert to its default channel definition. If the CList is present, it
1651 overrides **all** currently defined channels. The MAC layer parameters (except
1652 RXdelay1, RX2 data rate, and RX1 DR Offset that are transported by the join-accept
1653 message) SHALL all be reset to their default values.
- 1654 • A Rejoin-request Type 2 and if the CList field is absent, the device SHALL keep its
1655 current channels definition unchanged. If the CList is present, it overrides all
1656 currently defined channels. All other MAC parameters (except frame counters which
1657 are reset) are kept unchanged.

1658 In all cases following the successful processing of a Join-accept message the device SHALL
1659 transmit the **RekeyInd** MAC command until it receives the **RekeyConf** command (see 5.9).
1660 The reception of the **RekeyInd** uplink command is used by the Network Server as a signal to
1661 switch to the new security context.
1662

1663 6.2.4 ReJoin-request message

1664 Once activated a device MAY periodically transmit a Rejoin-request message on top of its
 1665 normal applicative traffic. This Rejoin-request message periodically gives the backend the
 1666 opportunity to initialize a new session context for the end-device. For this purpose the
 1667 network replies with a Join-Accept message. This may be used to hand-over a device
 1668 between two networks or to rekey and/or change devAddr of a device on a given network.

1669 The Network Server may also use the Rejoin-request RX1/RX2 windows to transmit a
 1670 normal confirmed or unconfirmed downlink frame optionally carrying MAC commands. This
 1671 possibility is useful to reset the device's reception parameters in case there is a MAC layer
 1672 state de-synchronization between the device and the Network Server.

1673 Example: This mechanism might be used to change the RX2 window data rate and the RX1
 1674 window data rate offset for a device that isn't reachable any more in downlink using the
 1675 current downlink configuration.

1676 The Rejoin procedure is always initiated from the end-device by sending a Rejoin-request
 1677 message.

1678 Note: Any time the network backend processes a ReJoin-Request
 1679 (type 0,1 or 2) and generates a Join-accept message, it shall maintain
 1680 both the old security context (keys and counters, if any) and the new
 1681 one until it receives the first successful uplink frame using the new
 1682 context, after which the old context may be safely discarded. In all
 1683 cases, the processing of the ReJoin-request message by the network
 1684 backend is similar to the processing of a standard Join-request
 1685 message, in that the Network Server initially processing the message
 1686 determines if it should be forwarded to a Join Server to create a Join-
 1687 accept message in response.

1688
 1689 There are three types of Rejoin-request messages that can be transmitted by an end device
 1690 and corresponds to three different purposes. The first byte of the Rejoin-request message is
 1691 called Rejoin Type and is used to encode the type of Rejoin-request. The following table
 1692 describes the purpose of each Rejoin-Request message type.
 1693

1694

RejoinReq type	Content & Purpose
0	Contains NetID+DevEUI. Used to reset a device context including all radio parameters (devAddr, session keys, frame counters, radio parameters, ..). This message can only be routed to the device's home Network Server by the receiving Network Server, not to the device's JoinServer. The MIC of this message can only be verified by the serving or home Network Server.
1	Contains JoinEUI+DevEUI. Exactly equivalent to the initial Join-Request message but may be transmitted on top of normal applicative traffic without disconnecting the device. Can only be routed to the device's JoinServer by the receiving Network Server. Used to restore a lost session context (Example, Network Server has lost the session keys and cannot associate the device to a JoinServer). Only the JoinServer is able to check the MIC of this message.
2	Contains NetID+DevEUI. Used to rekey a device or change its DevAddr (DevAddr, session keys, frame counters). Radio parameters are kept unchanged. This message can only be routed to the device's home Network Server by visited networks, not to the device's Join Server. The MIC of this message can only be verified by the serving or home Network Server.

1695

Table 15 : summary of RejoinReq messages

1696 6.2.4.1 ReJoin-request Type 0 or 2 message

1697

Size (bytes)	1	3	8	2
Rejoin-request	Rejoin Type = 0 or 2	NetID	DevEUI	RJcount0

1698

Figure 46: Rejoin-request type 0&2 message fields

1699 The Rejoin-request type 0 or 2 message contains the **NetID** (identifier of the device's home
1700 network) and **DevEUI** of the end-device followed by a 16 bits counter (**RJcount0**).

1701 **RJcount0** is a counter incremented with every Type 0 or 2 Rejoin frame transmitted.
1702 RJcount0 is initialized to 0 each time a Join-Accept is successfully processed by the end-
1703 device. For each end-device, the Network Server **MUST** keep track of the last **RJcount0**
1704 value (called RJcount0_last) used by the end-device. It ignores Rejoin-requests if (RJcount0
1705 <= RJcount0_last)

1706 RJcount0 SHALL never wrap around. If RJcount0 reaches $2^{16}-1$ the device SHALL stop
1707 transmitting ReJoin-request type 0 or 2 frames. The device MAY go back to Join state.

1708

1709 **Note:** This mechanism prevents replay attacks by sending previously
1710 recorded Rejoin-request messages

1711 The message integrity code (**MIC**) value (see Chapter 4 for MAC message description) for a
1712 Rejoin-request message is calculated as follows:¹

1713

1714 $cmac = \text{aes128_cmac}(\text{SNwkSIntKey}, \text{MHDR} \mid \text{Rejoin Type} \mid \text{NetID} \mid \text{DevEUI} \mid$
1715 $\text{RJcount0})$
1716 $\text{MIC} = cmac[0..3]$

¹ [RFC4493]

1717 The Rejoin-request message is not encrypted.
 1718 The device's **Rejoin-Req** type 0 or 2 transmissions duty-cycle SHALL always be <0.1%

1719 Note: The Rejoin-Request type 0 message is meant to be transmitted
 1720 from once per hour to once every few days depending on the device's
 1721 use case. This message can also be transmitted following a
 1722 ForceRejoinReq MAC command. This message may be used to
 1723 reconnect mobile device to a visited network in roaming situations. It
 1724 can also be used to rekey or change the devAddr of a static device.
 1725 Mobile devices expected to roam between networks should transmit
 1726 this message more frequently than static devices.

1727

1728 Note: The Rejoin-Request type 2 message is only meant to enable
 1729 rekeying of an end-device. This message can only be transmitted
 1730 following a ForceRejoinReq MAC command.

1731 6.2.4.2 ReJoin-request Type 1 message

1732 Similarly to the Join-Request, the Rejoin-Request type 1 message contains the JoinEUI and
 1733 the DevEUI of the end-device. The Rejoin-Request type 1 message can therefore be routed
 1734 to the Join Server of the end-device by any Network Server receiving it. The Rejoin-request
 1735 Type 1 may be used to restore connectivity with an end-device in case of complete state
 1736 loss of the Network Server. It is recommended to transmit a Rejoin-Request type 1
 1737 message a least once per month.

1738

1739

Size (bytes)	1	8	8	2
Rejoin-request	ReJoin Type = 1	JoinEUI	DevEUI	RJcount1

1740 **Figure 47: Rejoin-request type 1 message fields**

1741 The RJcount1 for Rejoin-request Type 1 is a different counter from the RJCount0 used for
 1742 Rejoin-request type 0.

1743 **RJcount1** is a counter incremented with every Rejoin-request Type 1 frame transmitted.
 1744 For each end-device, the Join Server keeps track of the last **RJcount1** value (called
 1745 RJcount1_last) used by the end-device. It ignores Rejoin-requests if (RJcount1 <= RJcount1_last).

1747 RJcount1 SHALL never wrap around for a given JoinEUI. The transmission periodicity of
 1748 Rejoin-Request type 1 shall be such that this wrap around cannot happen for the lifetime of
 1749 the device for a given JoinEUI value.

1750

1751 **Note:** This mechanism prevents replay attacks by sending previously
 1752 recorded Rejoin-request messages

1753 The message integrity code (**MIC**) value (see Chapter 4 for MAC message description) for a
 1754 Rejoin-request-Type1 message is calculated as follows:¹

1755

1756 $cmac = \text{aes128_cmac}(\text{JSIntKey}, \text{MHDR} \parallel \text{RejoinType} \parallel \text{JoinEUI} \parallel \text{DevEUI} \parallel \text{RJcount1})$

¹ [RFC4493]

1757 MIC = *cmac*[0..3]

1758 The Rejoin-request-type 1 message is not encrypted.

1759

1760 The device's **Rejoin-Req** type 1 transmissions duty-cycle shall always be **<0.01%**

1761 Note: The Rejoin-Request type 1 message is meant to be transmitted
 1762 from once a day to once a week. This message is only used in the
 1763 case of a complete loss of context of the server side. This event being
 1764 very unlikely a latency of 1 day to 1 week to reconnect the device is
 1765 considered as appropriate

1766 6.2.4.3 Rejoin-Request transmissions

1767

1768 The following table summarizes the possible conditions for transmission of each Rejoin-
 1769 request type message.

1770

RejoinReq type	Transmitted autonomously & periodically by the end-device	Transmitted following a ForceRejoinReq MAC command
0	x	x
1	x	
2		x

1771

Table 16 : transmission conditions for RejoinReq messages

1772 Rejoin-Request type 0&1 messages SHALL be transmitted on any of the defined Join
 1773 channels (see [PHY]) following a random frequency hopping sequence.

1774 Rejoin-Request type 2 SHALL be transmitted on any of the currently enabled channels
 1775 following a random frequency hopping sequence.

1776 Rejoin-Request type 0 or type 2 transmitted following a **ForceRejoinReq** command SHALL
 1777 use the data rate specified in the MAC command.

1778 Rejoin-Request type 0 transmitted periodically and autonomously by the end-device (with a
 1779 maximum periodicity set by the RejoinParamSetupReq command) and Rejoin-Request type
 1780 1 SHALL use:

- 1781 • The data rate & TX power currently used to transmit application payloads if ADR is
 1782 enabled
- 1783 • Any data rate allowed on the Join Channels and default TX power if ADR is disabled.
 1784 In that case it is RECOMMENDED to use a plurality of data rates.

1785 6.2.4.4 Rejoin-Request message processing

1786 For all 3 Rejoin-Request types the Network Server may respond with:

- 1787 • A **join-accept** message (as defined in 6.2.3) if it wants to modify the device's
1788 network identity (roaming or re-keying). In that case RJcount (0 or 1) replaces
1789 DevNonce in the key derivation process
- 1790 • A normal downlink frame optionally containing MAC commands. This downlink
1791 SHALL be sent on the same channel, with the same data rate and the same delay
1792 that the Join-accept message it replaces.
1793

1794 In most cases following a ReJoin-Request type 0 or 1 the network will not respond.
1795

1796 6.2.5 Key derivation diagram

1797 The following diagrams summarize the key derivation schemes for the cases where a device
1798 connects to a LoRaWAN1.0 or 1.1 Network Server.

LoRaWAN1.0 network backend:

When a LoRaWAN1.1 device is provisioned with a LoRaWAN1.0.X network backend, all keys are derived from the **NwkKey** root key. The device's **AppKey** is not used.

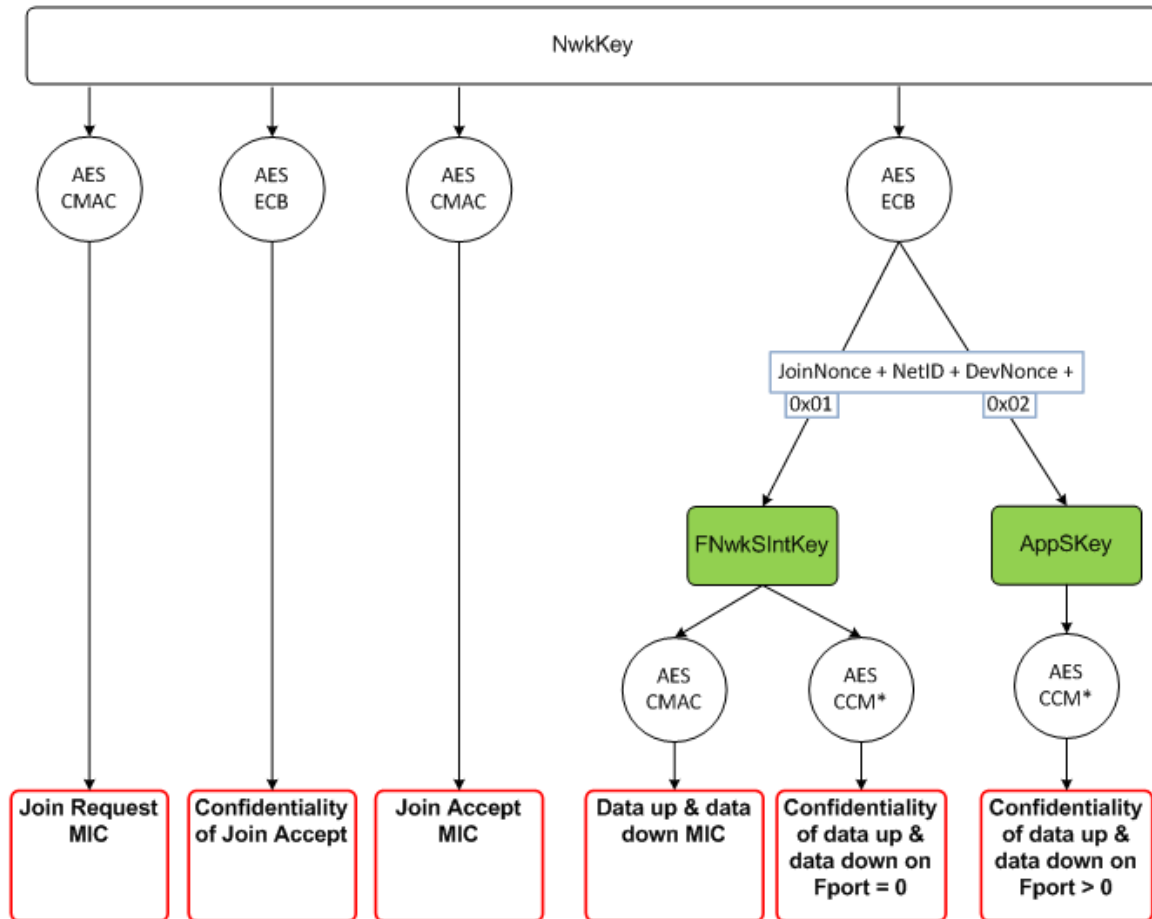


Figure 48 : LoRaWAN1.0 key derivation scheme

LoRaWAN1.1 network backend:

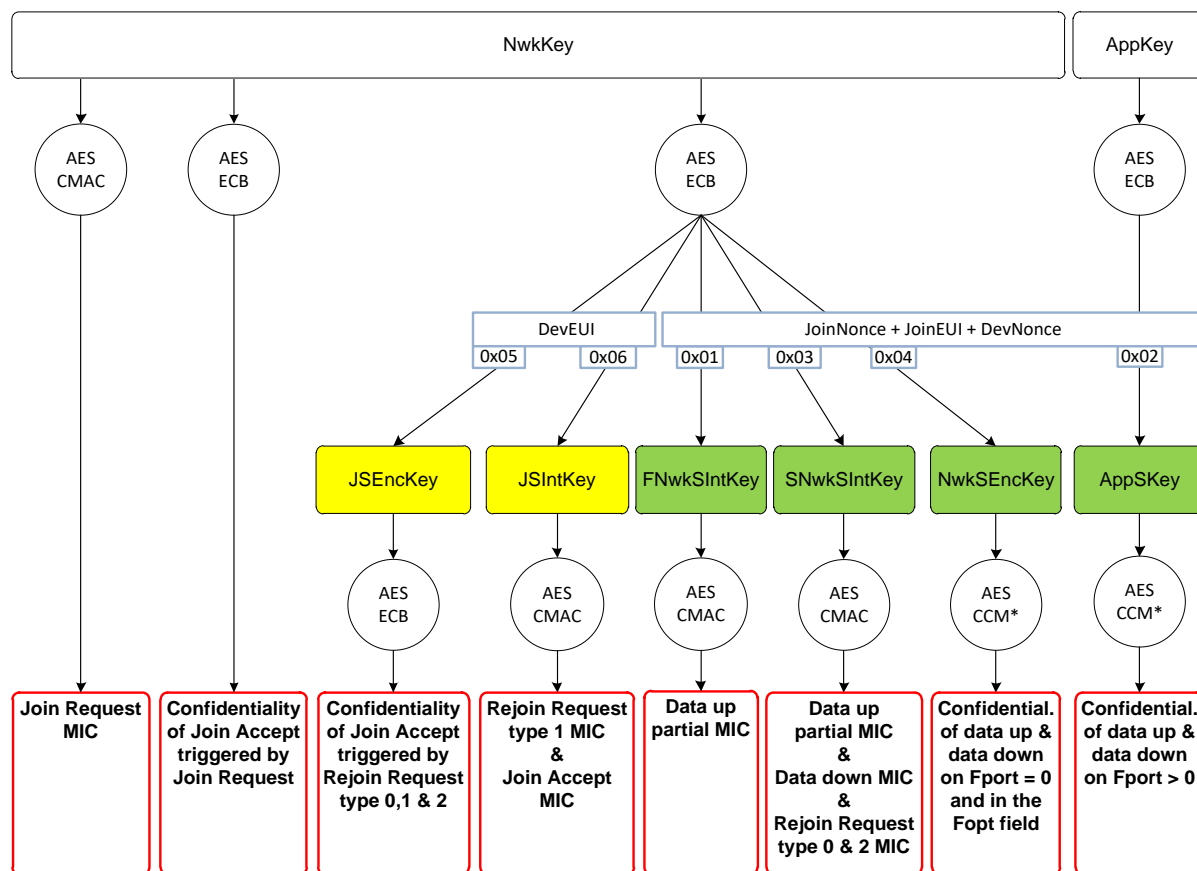


Figure 49 : LoRaWAN1.1 key derivation scheme

1810 6.3 Activation by Personalization

1811 Activation by personalization directly ties an end-device to a specific network by-passing the
1812 **Join-request - Join-accept** procedure.

1813 Activating an end-device by personalization means that the **DevAddr** and the four session
1814 keys **FNwkSIntKey**, **SNwkSIntKey**, **NwkSEncKey** and **AppSKey** are directly stored into
1815 the end-device instead of being derived from the **DevEUI**, **JoinEUI**, **AppKey** and **NwkKey**
1816 during the join procedure. The end-device is equipped with the required information for
1817 participating in a specific LoRa network as soon as it is started.

1818 Each device SHALL have a unique set of **F/SNwkSIntKey**, **NwkSEncKey** and **AppSKey**.
1819 Compromising the keys of one device SHALL NOT compromise the security of the
1820 communications of other devices. The process to build those keys SHALL be such that the
1821 keys cannot be derived in any way from publicly available information (like the node address
1822 or the end-device's devEUI for example).

1823 When a personalized end-device accesses the network for the first time or after a re-
1824 initialization, it SHALL transmit the ResetInd MAC command in the FOpt field of all uplink
1825 messages until it receives a ResetConf command from the network. After a re-initialization
1826 the end-device MUST use its default configuration (id the configuration that was used when
1827 the device was first connected to the network).

1828 **Note:** Frame counter values SHALL only be used once in all
1829 invocations of a same key with the CCM* mode of operation.
1830 Therefore, re-initialization of an ABP end-device frame counters is
1831 forbidden. ABP devices MUST use a non-volatile memory to store the
1832 frame counters.

1833 ABP devices use the same session keys throughout their lifetime (i.e.,
1834 no rekeying is possible. Therefore, it is recommended that OTAA
1835 devices are used for higher security applications.
1836

7 Retransmissions back-off

Uplink frames that:

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

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

Note: An example of such uplink frame is typically the 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 whole group of end-device will start broadcasting Join-request uplinks and will only stops when receiving a JoinResponse from the network.

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

Aggregated during the first hour following power-up or reset	$T_0 < t < T_0 + 1h$	Transmit time < 36Sec
Aggregated during the next 10 hours	$T_0 + 1 < t < T_0 + 11h$	Transmit time < 36Sec
After the first 11 hours , aggregated over 24h	$T_0 + 11 + N < t < T_0 + 35 + N$ $N \geq 0$	Transmit time < 8.7Sec per 24h

Table 17 : Join-request dutycycle limitations

1863

1864

CLASS B – BEACON

8 Introduction to Class B

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

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

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

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

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

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

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

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

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

1905

1906 All end-devices start and join the network as end-devices of Class A. The end-device
1907 application can then decide to switch to Class B. This is done through the following process:

1908 • The end-device application requests the LoRaWAN layer to switch to Class B mode.
1909 The LoRaWAN layer in the end-device searches for a beacon and returns either a
1910 BEACON_LOCKED service primitive to the application if a network beacon was
1911 found and locked or a BEACON_NOT_FOUND service primitive. To accelerate the
1912 beacon discovery the LoRaWAN layer may use the “DeviceTimeReq” MAC
1913 command.

1914 • Once in Class B mode, the MAC layer sets to 1 the *Class B* bit of the FCTRL field of
1915 every uplink frame transmitted. This bit signals to the server that the device has
1916 switched to Class B. The MAC layer will autonomously schedule a reception slot for
1917 each beacon and each ping slot. When the beacon reception is successful the end-
1918 device LoRaWAN layer forwards the beacon content to the application together with
1919 the measured radio signal strength. The end-device LoRaWAN layer takes into
1920 account the maximum possible clock drift in the scheduling of the beacon reception
1921 slot and ping slots. When a downlink is successfully demodulated during a ping slot,
1922 it is processed similarly to a downlink as described in the LoRaWAN Class A
1923 specification.

1924 • A mobile end-device must periodically inform the Network Server of its location to
1925 update the downlink route. This is done by transmitting a normal (possibly empty)
1926 “unconfirmed” or “confirmed” uplink. The end-device LoRaWAN layer will
1927 appropriately set the *Class B* bit to 1 in the frame’s FCtrl field. Optimally this can be
1928 done more efficiently if the application detects that the node is moving by analyzing
1929 the beacon content. In that case the end-device must apply a random delay (as
1930 defined in Section 15.5 between the beacon reception and the uplink transmission to
1931 avoid systematic uplink collisions.

1932 • At any time the Network Server may change the device’s ping-slot downlink
1933 frequency or data rate by sending a PingSlotChannelReq MAC command.

1934 • The device may change the periodicity of its ping-slots at any time. To do so, it
1935 MUST temporarily stop class B operation (unset classB bit in its uplink frames) and
1936 send a PingSlotInfoReq to the Network Server. Once this command is acknowledged
1937 the device may restart classB operation with the new ping-slot periodicity

1938 • If no beacon has been received for a given period (as defined in Section 12.2), the
1939 synchronization with the network is lost. The MAC layer must inform the application
1940 layer that it has switched back to Class A. As a consequence the end-device
1941 LoRaWAN layer stops setting the *Class B* bit in all uplinks and this informs the
1942 Network Server that the end-device is no longer in Class B mode. The end-device
1943 application can try to switch back to Class B periodically. This will restart this process
1944 starting with a beacon search.

1945 The following diagram illustrates the concept of beacon reception slots and ping slots.

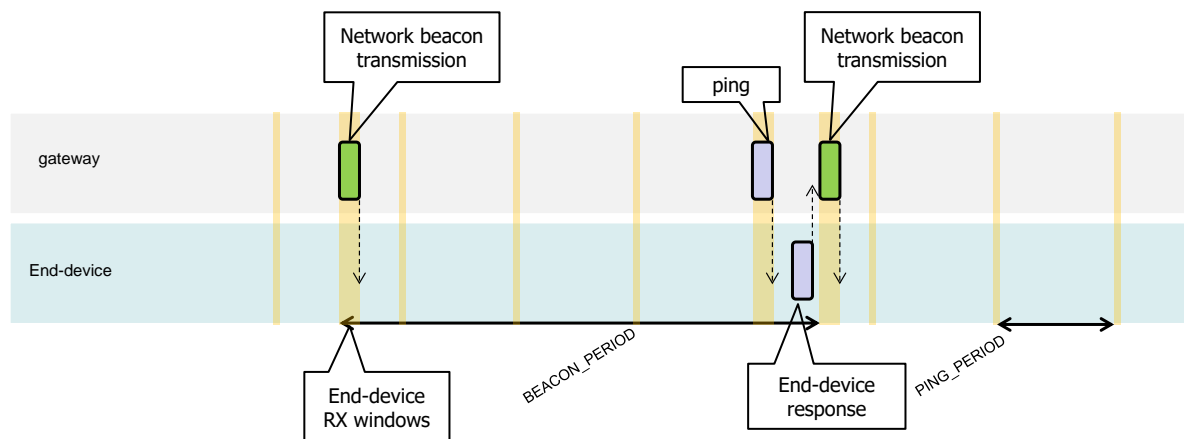


Figure 50: Beacon reception slot and ping slots

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

10 Uplink frame in Class B mode

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

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

Figure 51 : classB FCtrl fields

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

The signification of the FPending bit for downlink is unaltered and still signals that one or more downlink frames are queued for this device in the server and that the device should keep is receiver on as described in the Class A specification.

11 Downlink Ping frame format (Class B option)

11.1 Physical frame format

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

11.2 Unicast & Multicast MAC messages

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

11.2.1 Unicast MAC message format

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

11.2.2 Multicast MAC message format

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

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

12 Beacon acquisition and tracking

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

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

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

Note: For example, a device which internal clock is defined with a ± 10 ppm precision may drift by ± 1.3 ms every beacon period.

12.1 Minimal beacon-less operation time

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

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

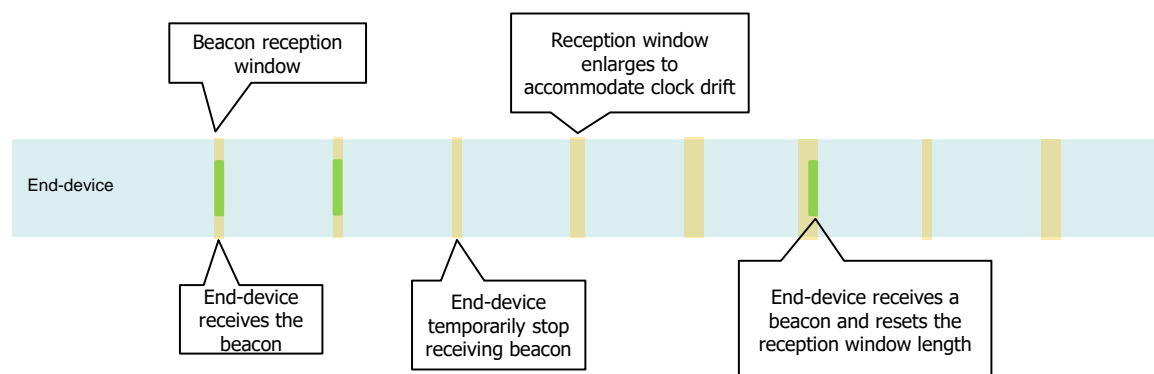


Figure 52 : beacon-less temporary operation

12.2 Extension of beacon-less operation upon reception

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

12.3 Minimizing timing drift

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

13 Class B Downlink slot timing

13.1 Definitions

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

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

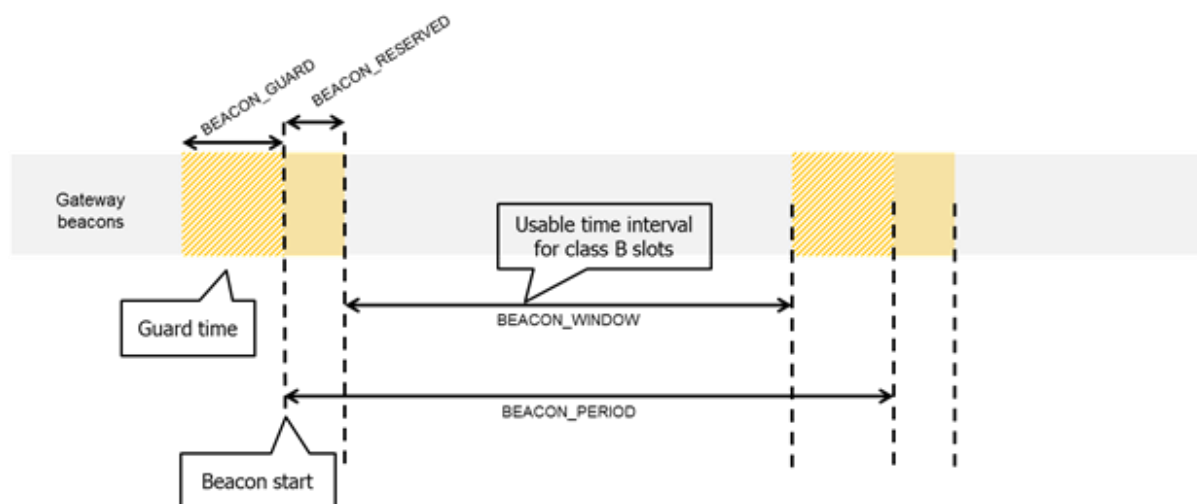


Figure 53: Beacon timing

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

Table 18: Beacon timing

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

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

An end-device using the slot number N must turn on its receiver exactly T_{on} seconds after the start of the beacon where:

$$T_{on} = \text{beacon_reserved} + N * 30 \text{ ms}$$

2048 N is called the *slot index*.

2049 The latest ping slot starts at *beacon_reserved* + 4095 * 30 ms = 124 970 ms after the
2050 beacon start or 3030 ms before the beginning of the next beacon.

2051 13.2 Slot randomization

2052 To avoid systematic collisions or over-hearing problems the slot index is randomized and
2053 changed at every beacon period.

2054 The following parameters are used:

2055

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

2056 **Table 19 : classB slot randomization algorithm parameters**

2057 At each beacon period the end-device and the server compute a new pseudo-random offset
2058 to align the reception slots. An AES encryption with a fixed key of all zeros is used to
2059 randomize:

2060 $Key = 16 \times 0x00$

2061 $Rand = aes128_encrypt(Key, beaconTime \parallel DevAddr \parallel pad16)$

2062 $pingOffset = (Rand[0] + Rand[1] \times 256) \text{ modulo } pingPeriod$

2063 The slots used for this beacon period will be:

2064 $pingOffset + N \times pingPeriod$ with $N=[0:pingNb-1]$

2065 The node therefore opens receive slots starting at :

First slot	$Beacon_reserved + pingOffset \times slotLen$
Slot 2	$Beacon_reserved + (pingOffset + pingPeriod) \times slotLen$
Slot 3	$Beacon_reserved + (pingOffset + 2 \times pingPeriod) \times slotLen$
...	...

2066 If the end-device serves simultaneously a unicast and one or more multicast slots this
2067 computation is performed multiple times at the beginning of a new beacon period. Once for
2068 the unicast address (the node network address) and once for each multicast group address.

2069 In the case where a multicast ping slot and a unicast ping slot collide and cannot be served
2070 by the end-device receiver then the end-device should preferentially listen to the multicast
2071 slot. If there is a collision between multicast reception slots the FPending bit of the previous
2072 multicast frame can be used to set a preference.

2073 The randomization scheme prevents a systematic collision between unicast and multicast
2074 slots. If collisions happen during a beacon period then it is unlikely to occur again during the
2075 next beacon period.

14 Class B MAC commands

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

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

Table 20 : classB MAC command table

14.1 PingSlotInfoReq

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

Size (bytes)	1
PingSlotInfoReq Payload	PingSlotParam

Figure 54 : PingSlotInfoReq payload format

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

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

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

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

2092 • **Periodicity** = 0 means that the end-device opens a ping slot approximately every
2093 second during the beacon_window interval

2094 • **Periodicity** = 7 , every 128 seconds which is the maximum ping period supported by
2095 the LoRaWAN Class B specification.

2096 To change its ping slot periodicity a device SHALL first revert to Class A , send the new
2097 periodicity through a **PingSlotInfoReq** command and get an acknowledge from the server
2098 through a **PingSlotInfoAns** . It MAY then switch back to Class B with the new periodicity.

2099 This command MAY be concatenated with any other MAC command in the **FHDRFOpt** field
2100 as described in the Class A specification frame format.

2101 14.2 BeaconFreqReq

2102 This command is sent by the server to the end-device to modify the frequency on which this
2103 end-device expects the beacon.

2104

Octets	3
BeaconFreqReq payload	Frequency

Figure 55 : BeaconFreqReq payload format

2105

2106 The Frequency coding is identical to the **NewChannelReq** MAC command defined in the
2107 Class A.

2108 **Frequency** is a 24bits unsigned integer. The actual beacon channel frequency in Hz is 100
2109 x frequ. This allows defining the beacon channel anywhere between 100 MHz to 1.67 GHz
2110 by 100 Hz step. The end-device has to check that the frequency is actually allowed by its
2111 radio hardware and return an error otherwise.

2112 A valid non-zero Frequency will force the device to listen to the beacon on a fixed frequency
2113 channel even if the default behavior specifies a frequency hopping beacon (i.e US ISM
2114 band).

2115 A value of 0 instructs the end-device to use the default beacon frequency plan as defined in
2116 the “Beacon physical layer” section. Where applicable the device resumes frequency
2117 hopping beacon search.

2118 Upon reception of this command the end-device answers with a **BeaconFreqAns** message.
2119 The MAC payload of this message contains the following information:

Size (bytes)	1
BeaconFreqAns payload	Status

Figure 56 : BeaconFreqAns payload format

2120

2121 The **Status** bits have the following meaning:

Bits	7:1	0
Status	RFU	Beacon frequency ok

2122

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

2123

2124 **14.3 PingSlotChannelReq**

2125 This command is sent by the server to the end-device to modify the frequency and/or the
 2126 data rate on which the end-device expects the downlink pings.

2127 This command **can only be sent in a class A receive window** (following an uplink). The
 2128 command SHALL NOT be sent in a class B ping-slot. If the device receives it inside a class
 2129 B ping-slot, the MAC command SHALL NOT be processed.

2130

Octets	3	1
PingSlotChannelReq Payload	Frequency	DR

2131

Figure 57 : PingSlotChannelReq payload format

2132 The Frequency coding is identical to the **NewChannelReq** MAC command defined in the
 2133 Class A.

2134 **Frequency** is a 24bits unsigned integer. The actual ping channel frequency in Hz is 100 x
 2135 frequ. This allows defining the ping channel anywhere between 100MHz to 1.67GHz by
 2136 100Hz step. The end-device has to check that the frequency is actually allowed by its radio
 2137 hardware and return an error otherwise.

2138 A value of 0 instructs the end-device to use the default frequency plan.

2139 The DR byte contains the following fields:

2140

Bits	7:4	3:0
DR	RFU	data rate

2141

2142 The “data rate” subfield is the index of the Data Rate used for the ping-slot downlinks. The
 2143 relationship between the index and the physical data rate is defined in [PHY] for each region.

2144 Upon reception of this command the end-device answers with a **PingSlotFreqAns**
 2145 message. The MAC payload of this message contains the following information:

2146

Size (bytes)	1
pingSlotFreqAns Payload	Status

2147

Figure 58 : PingSlotFreqAns payload format

2148 The **Status** bits have the following meaning:

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

2149

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

2150

2151
2152 If either of those 2 bits equals 0, the command did not succeed and the ping-slot parameters
2153 have not been modified.
2154

2155 **14.4 BeaconTimingReq & BeaconTimingAns**

2156 These MAC commands are deprecated in the LoRaWAN1.1 version. The device may use
2157 DeviceTimeReq&Ans commands as a substitute.
2158

15 Beaconing (Class B option)

15.1 Beacon physical layer

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

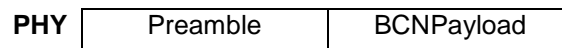


Figure 59 : beacon physical format

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

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

15.2 Beacon frame content

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

Size (bytes)	2/3	4	2	7	0/1	2
BCNPayload	RFU	Time	CRC	GwSpecific	RFU	CRC

Figure 60 : beacon frame content

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

For example: This is a valid EU868 beacon frame:

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

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

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

Figure 61 : example of beacon CRC calculation (1)

2190

2191 The gateway specific part provides additional information regarding the gateway sending a
2192 beacon and therefore may differ for each gateway. The RFU field when applicable (region
2193 specific) should be equal to 0. The optional part is protected by a CRC-16 computed on the
2194 GwSpecific+RFU fields. The CRC-16 definition is the same as for the mandatory part.

2195 For example: This is a valid US900 beacon:

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

2196

Figure 62 : example of beacon CRC calculation (2)

2197 Over the air the bytes are sent in the following order:

2198 00 00 00 | 00 00 02 CC | A2 7E | 00 | 01 20 00 | 00 81 03 | 00 | 50 D4

2199 Listening and synchronizing to the network common part is sufficient to operate a stationary
2200 end-device in Class B mode. A mobile end-device may also demodulate the gateway
2201 specific part of the beacon to be able to signal to the Network Server whenever he is moving
2202 from one cell to another.

2203 **Note:** As mentioned before, all gateways participating in the beaoning
2204 process send their beacon simultaneously so that for network common
2205 part there are no visible on-air collisions for a listening end-device even
2206 if the end-device simultaneously receives beacons from several
2207 gateways. Not all gateways are required to participate in the beaoning
2208 process. The participation of a gateway to a given beacon may be
2209 randomized. With respect to the gateway specific part, collision occurs
2210 but an end-device within the proximity of more than one gateway will
2211 still be able to decode the strongest beacon with high probability.

2212 15.3 Beacon GwSpecific field format

2213 The content of the **GwSpecific** field is as follow:

Size (bytes)	1	6
GwSpecific	InfoDesc	Info

2214

Figure 63 : beacon GwSpecific field format

2215 The information descriptor **InfoDesc** describes how the information field **Info** shall be
2216 interpreted.

2217

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
3:127	RFU
128:255	Reserved for custom network specific broadcasts

2218

Table 21 : beacon infoDesc index mapping

2219 For a single omnidirectional antenna gateway the **InfoDesc** value is 0 when broadcasting
2220 GPS coordinates. For a site featuring 3 sectorized antennas for example, the first antenna

2221 broadcasts the beacon with **InfoDesc** equals 0, the second antenna with **InfoDesc** field
2222 equals 1, etc.

2223 15.3.1 Gateway GPS coordinate:InfoDesc = 0, 1 or 2

2224 For **InfoDesc** = 0 ,1 or 2, the content of the **Info** field encodes the GPS coordinates of the
2225 antenna broadcasting the beacon

Size (bytes)	3	3
Info	Lat	Lng

Figure 64 : beacon Info field format

2227 The latitude and longitude fields (**Lat** and **Lng**, respectively) encode the geographical
2228 location of the gateway as follows:

- 2229 • The north-south latitude is encoded using a two's complement 24 bit word where -2^{23}
2230 corresponds to 90° south (the South Pole) and $2^{23}-1$ corresponds to ~90° north (the
2231 North Pole). The Equator corresponds to 0.
- 2232 • The east-west longitude is encoded using a two's complement 24 bit word where -
2233 2^{23} corresponds to 180° West and $2^{23}-1$ corresponds to ~180° East. The Greenwich
2234 meridian corresponds to 0.

2235 15.4 Beaconsing precise timing

2236 The beacon is sent every 128 seconds starting at 00:00:00, Sunday 5th – Monday 6th of
2237 January 1980 (start of the GPS epoch) plus TBeaconDelay. Therefore the beacon is sent at
2238 $B_T = k * 128 + T_{\text{BeaconDelay}}$

2239 seconds after the GPS epoch.

2240 whereby k is the smallest integer for which

$$2241 \quad k * 128 > T$$

2242 whereby

2243 T = seconds since 00:00:00, Sunday 5th of January 1980 (start of the GPS time).

2244 **Note:** T is GPS time and unlike Unix time, T is strictly monotonically
2245 increasing and is not influenced by leap seconds.

2246 Whereby TBeaconDelay is 1.5 mSec +/- 1uSec delay.

2248 TBeaconDelay is meant to allow a slight transmission delay of the gateways required by the
2249 radio system to switch from receive to transmit mode.

2250 All end-devices ping slots use the beacon transmission start time as a timing reference,
2251 therefore the Network Server as to take TBeaconDelay into account when scheduling the
2252 class B downlinks.

2254 15.5 Network downlink route update requirements

2255 When the network attempts to communicate with an end-device using a Class B downlink
2256 slot, it transmits the downlink from the gateway which was closest to the end-device when

2257 the last uplink was received. Therefore the Network Server needs to keep track of the rough
2258 position of every Class B device.

2259 Whenever a Class B device moves and changes cell, it needs to communicate with the
2260 Network Server in order to update its downlink route. This update can be performed simply
2261 by sending a “confirmed” or “unconfirmed” uplink, possibly without applicative payload.

2262 The end-device has the choice between 2 basic strategies:

- 2263 • Systematic periodic uplink: simplest method that doesn’t require demodulation of the
2264 “gateway specific” field of the beacon. Only applicable to slowly moving or stationery
2265 end-devices. There are no requirements on those periodic uplinks.
- 2266 • Uplink on cell change: The end-device demodulates the “gateway specific” field of
2267 the beacon, detects that the ID of the gateway broadcasting the beacon it
2268 demodulates has changed, and sends an uplink. In that case the device SHALL
2269 respect a pseudo random delay in the [0:120] seconds range between the beacon
2270 demodulation and the uplink transmission. This is required to insure that the uplinks
2271 of multiple Class B devices entering or leaving a cell during the same beacon period
2272 will not systematically occur at the same time immediately after the beacon
2273 broadcast.

2274 Failure to report cell change will result in Class B downlink being temporary not operational.
2275 The Network Server may have to wait for the next end-device uplink to transmit downlink
2276 traffic.

2277
2278

2279 16 Class B unicast & multicast downlink channel frequencies

2280 The class B downlink channel selection mechanism depends on the way the class B beacon
2281 is being broadcasted.

2282 16.1 Single channel beacon transmission

2283 In certain regions (ex EU868) the beacon is transmitted on a single channel. In that case, all
2284 unicast&multicast Class B downlinks use a single frequency channel defined by the
2285 “**PingSlotChannelReq**” MAC command. The default frequency is defined in [PHY].

2286 16.2 Frequency-hopping beacon transmission

2287 In certain regions (ex US902-928 or CN470-510) the class B beacon is transmitted following
2288 a frequency hopping pattern.

2289 In that case, by default Class B downlinks use a channel which is a function of the Time field
2290 of the last beacon (see Beacon Frame content) and the DevAddr.

2291 Class B downlink channel = $\left[\text{DevAddr} + \text{floor} \left(\frac{\text{Beacon_Time}}{\text{Beacon_period}} \right) \right] \text{ modulo NbChannel}$

- 2292 • Whereby Beacon_Time is the 32 bit Time field of the current beacon period
- 2293 • Beacon_period is the length of the beacon period (defined as 128sec in the
2294 specification)
- 2295 • Floor designates rounding to the immediately lower integer value
- 2296 • DevAddr is the 32 bits network address of the device
- 2297 • NbChannel is the number of channel over which the beacon is frequency hopping

2298 Class B downlinks therefore hop across NbChannel channels (identical to the beacon
2299 transmission channels) in the ISM band and all Class B end-devices are equally spread
2300 amongst the NbChannel downlink channels.

2301 If the “**PingSlotChannelReq**” command with a valid non-zero argument is used to set the
2302 Class B downlink frequency then all subsequent ping slots should be opened on this single
2303 frequency independently of the last beacon frequency.

2304 If the “**PingSlotChannelReq**” command with a zero argument is sent, the end-device
2305 should resume the default frequency plan, id Class B ping slots hoping across 8 channels.

2306 The underlying idea is to allow network operators to configure end-devices to use a single
2307 proprietary dedicated frequency band for the Class B downlinks if available, and to keep as
2308 much frequency diversity as possible when the ISM band is used.

2309

2310

CLASS C – CONTINUOUSLY LISTENING

2311 17 Class C: Continuously listening end-device

2312 The end-devices implementing the Class C option are used for applications that have sufficient
2313 power available and thus do not need to minimize reception time.

2314 Class C end-devices SHALL NOT implement Class B option.

2315 The Class C end-device will listen with RX2 windows parameters as often as possible. The
2316 end-device SHALL listen on RX2 when it is not either (a) sending or (b) receiving on RX1,
2317 according to Class A definition. To do so, it MUST open a short window using RX2
2318 parameters between the end of the uplink transmission and the beginning of the RX1
2319 reception window and MUST switch to RX2 reception parameters as soon as the RX1
2320 reception window is closed; the RX2 reception window MUST remain open until the end-
2321 device has to send another message.

Note: If the device is in the process of demodulating a downlink using the RX2 parameters when the RX1 window should be opened, it shall drop the demodulation and switch to the RX1 receive window

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

2329 In case a message is received by a device in Class C mode requiring an uplink transmission
2330 (DL MAC command request or DL message in confirmed mode), the device SHALL answer
2331 within a time period known by both the end-device and the Network Server (out-of-band
2332 provisioning information).

2333 Before this timeout expires, the network SHALL not send any new confirmed message or
2334 MAC command to the device. Once this timeout expires or after reception of any uplink
2335 message, the network is allowed to send a new DL message.

2336 17.1 Second receive window duration for Class C

2337 Class C devices implement the same two receive windows as Class A devices, but they do
2338 not close RX2 window until they need to send again. Therefore they may receive a downlink
2339 in the RX2 window at nearly any time, including downlinks sent for the purpose of MAC
2340 command or ACK transmission. A short listening window on RX2 frequency and data rate is
2341 also opened between the end of the transmission and the beginning of the RX1 receive
2342 window.

2343
2344

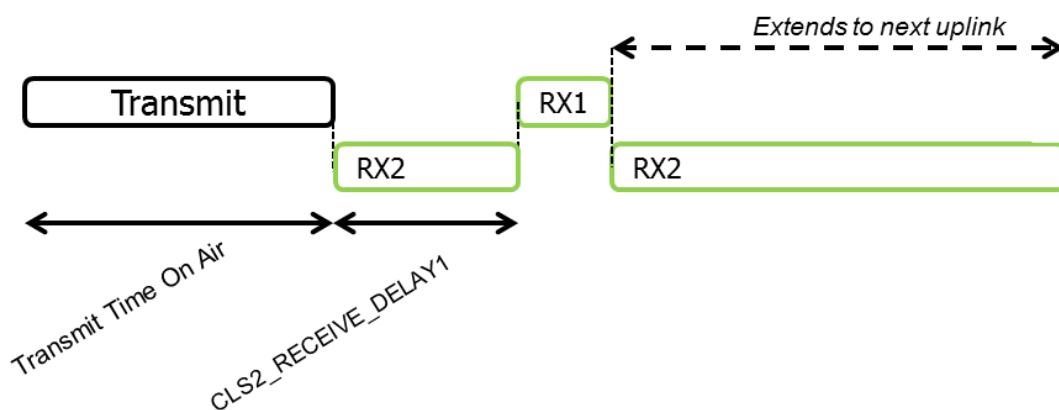


Figure 65: Class C end-device reception slot timing.

17.2 Class C Multicast downlinks

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

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

18 Class C MAC command

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

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x20	DeviceModelInd	x		Used by the end-device to indicate its current operating mode (Class A or C)
0x20	DeviceModeConf		x	Used by the network to acknowledge a DeviceModelInd command

Table 22 : Class C MAC command table

18.1 Device Mode (DeviceModelInd, DeviceModeConf)

With the **DeviceModelInd** command, an end-device indicates to the network that it wants to operate either in class A or C. The command has a one byte payload defined as follows:

Size (bytes)	1
DeviceModelInd Payload	Class

Figure 66 : DeviceModelInd payload format

With the classes defined for the above commands as:

Class	Value
Class A	0x00
RFU	0x01
Class C	0x02

Table 23 : DeviceModInd class mapping

When a **DeviceModelInd** command is received by the Network Server, it responds with a **DeviceModeConf** command. The device SHALL include the **DeviceModelInd** command in all uplinks until the **DeviceModeConf** command is received.

The device SHALL switch mode as soon as the first **DeviceModelInd** command is transmitted.

Note: When transitioning from class A to class C, It is recommended for battery powered devices to implement a time-out mechanism in the application layer to guarantee that it does not stay indefinitely in class C mode if no connection is possible with the network.

The **DeviceModeConf** command has a 1 byte payload.

Size (bytes)	1
DeviceModeConf Payload	Class

With the class parameter defined as for the **DeviceModelInd** command

SUPPORT INFORMATION

2387

2388 This sub-section is only a recommendation.

2389

19 Examples and Application Information

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

19.1 Uplink Timing Diagram for Confirmed Data Messages

The following diagram illustrates the steps followed by an end-device trying to transmit two confirmed data frames (Data0 and Data1). This device's NbTrans parameter must be greater or equal to 2 for this example to be valid (because the first confirmed frame is transmitted twice)

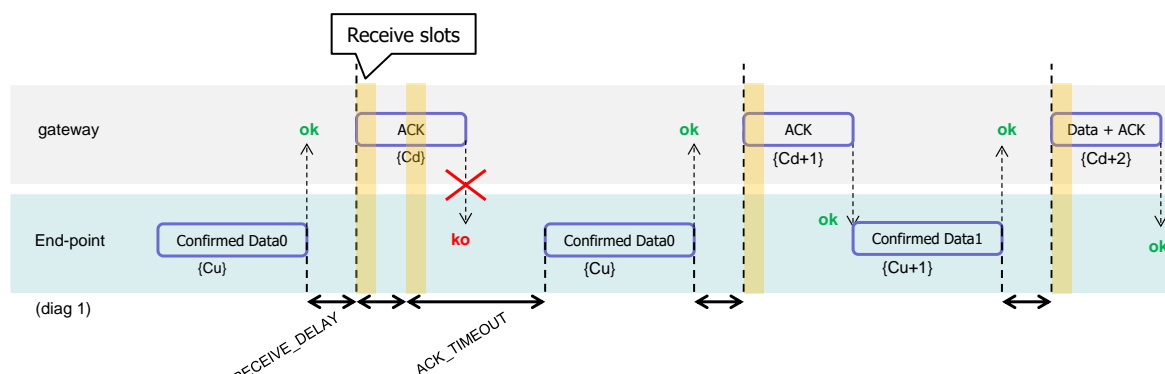


Figure 67: Uplink timing diagram for confirmed data messages

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

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

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

19.2 Downlink Diagram for Confirmed Data Messages

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

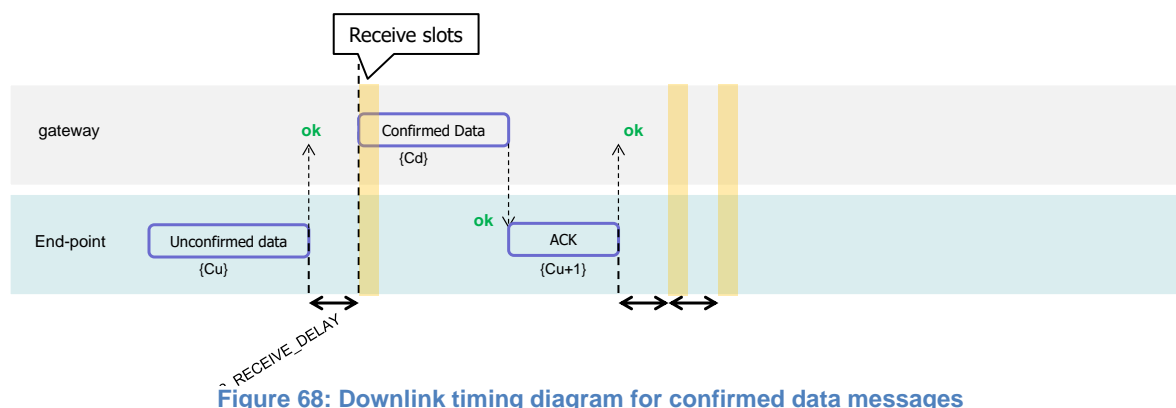


Figure 68: Downlink timing diagram for confirmed data messages

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

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

19.3 Downlink Timing for Frame-Pending Messages

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

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

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

(*) F_P means ‘frame pending’ bit set

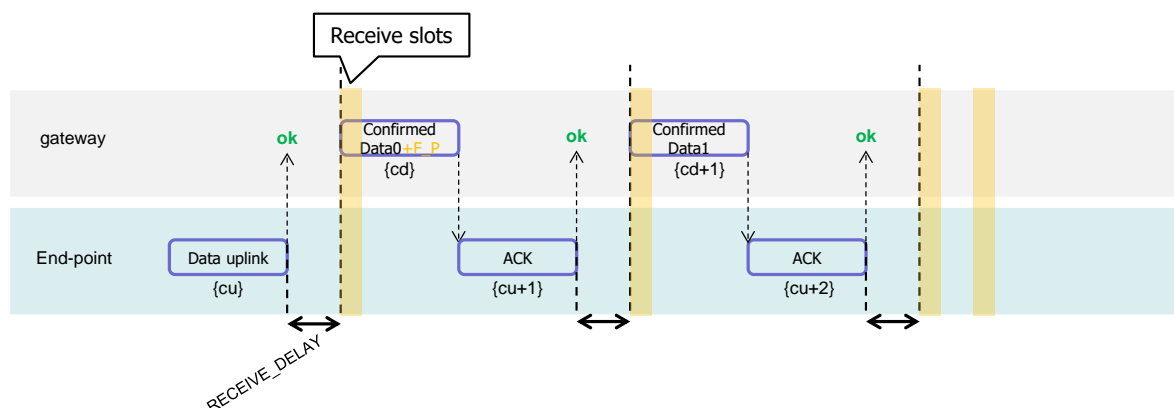


Figure 69: Downlink timing diagram for frame-pending messages, example 1

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

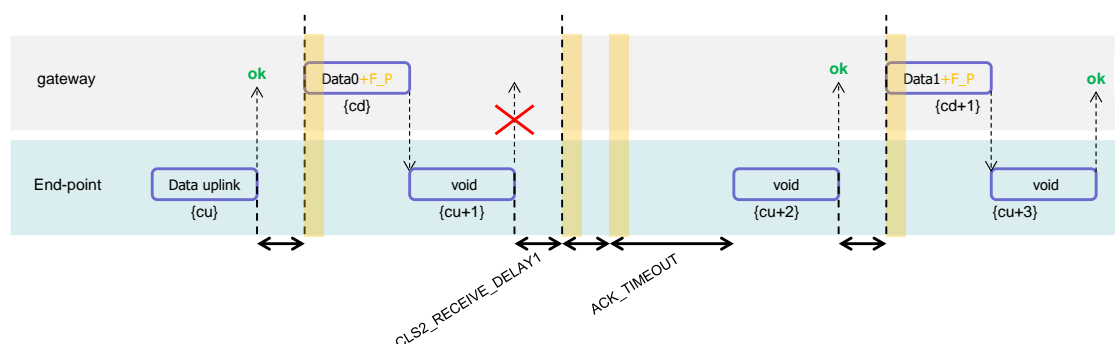


Figure 70: Downlink timing diagram for frame-pending messages, example 2

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

Note: An acknowledgement is never sent twice.

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

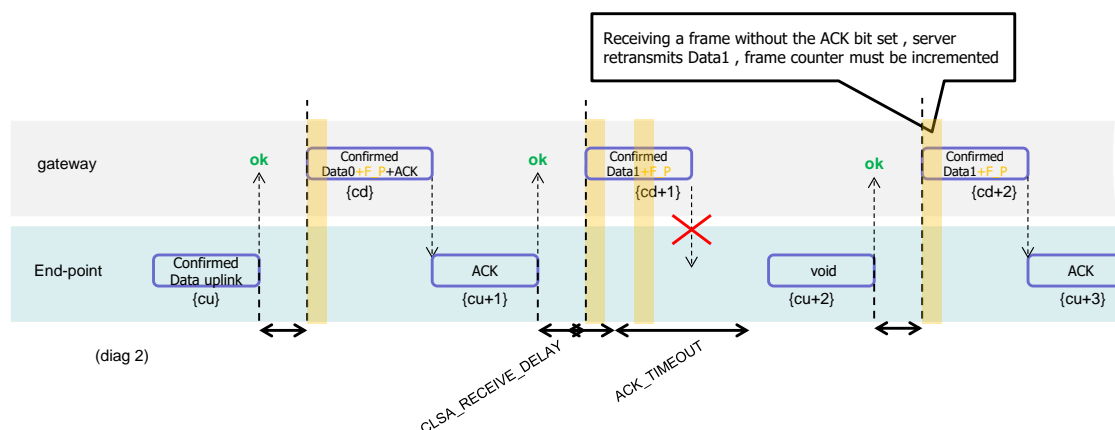


Figure 71: Downlink timing diagram for frame-pending messages, example 3

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

2478 **20 Recommendation on contract to be provided to the Network**
2479 **Server by the end-device provider at the time of provisioning**

2480 Configuration data related to the end-device and its characteristics must be known by the
2481 Network Server at the time of provisioning. –This provisioned data is called the “contract”.
2482 This contract cannot be provided by the end-device and must be supplied by the end-device
2483 provider using another channel (out-of-band communication).

2484 This end-device contract is stored in the Network Server. It can be used by the Application
2485 Server and the network controller to adapt the algorithms.

2486 This data will include:

- 2487 • End-device specific radio parameters (device frequency range, device maximal
2488 output power, device communication settings - RECEIVE_DELAY1,
2489 RECEIVE_DELAY2)
- 2490 • Application type (Alarm, Metering, Asset Tracking, Supervision, Network Control)

21 Recommendation on finding the locally used channels

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

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

2503 22 Revisions

2504 22.1 Revision 1.0

- 2505 • Approved version of LoRaWAN1.0

2506 22.2 Revision 1.0.1

- 2507 • Clarified the RX window start time definition
- 2508 • Corrected the maximum payload size for DR2 in the NA section
- 2509 • Corrected the typo on the downlink data rate range in 7.2.2
- 2510 • Introduced a requirement for using coding rate 4/5 in 7.2.2 to guarantee a maximum
- 2511 time on air < 400mSec
- 2512 • Corrected the Join-accept MIC calculation in 6.2.5
- 2513 • Clarified the NbRep field and renamed it to NbTrans in 5.2
- 2514 • Removed the possibility to not encrypt the Applicative payload in the MAC layer ,
- 2515 removed the paragraph 4.3.3.2. If further security is required by the application , the
- 2516 payload will be encrypted, using any method, at the application layer then re-
- 2517 encrypted at the MAC layer using the specified default LoRaWAN encryption
- 2518 • Corrected FHDR field size typo
- 2519 • Corrected the channels impacted by ChMask when chMaskCntl equals 6 or 7 in
- 2520 7.2.5
- 2521 • Clarified 6.2.5 sentence describing the RX1 slot data rate offset in the JoinResp
- 2522 message
- 2523 • Removed the second half of the DROffset table in 7.2.7 , as DR>4 will never be used
- 2524 for uplinks by definition
- 2525 • Removed explicit duty cycle limitation implementation in the EU868Mhz ISM band
- 2526 (chapter7.1)
- 2527 • Made the RXtimingSetupAns and RXParamSetupAns sticky MAC commands to
- 2528 avoid end-device's hidden state problem. (in 5.4 and 5.7)
- 2529 • Added a frequency plan for the Chinese 470-510MHz metering band
- 2530 • Added a frequency plan for the Australian 915-928MHz ISM band
- 2531

2532 22.3 Revision 1.0.2

- 2533 • Extracted section 7 “Physical layer” that will now be a separated document
- 2534 “LoRaWAN regional physical layers definition”
- 2535 • corrected the ADR_backoff sequence description (ADR_ACK_LIMT was written
- 2536 instead of ADR_ACK_DELAY) paragraph 4.3.1.1
- 2537 • Corrected a formatting issue in the title of section 18.2 (previously section 19.2 in the
- 2538 1.0.1 version)
- 2539 • Added the DICHannelRec MAC command, this command is used to modify the
- 2540 frequency at which an end-device expects a downlink.
- 2541 • Added the Tx ParamSetupRec MAC command. This command enables to remotely
- 2542 modify the maximum TX dwell time and the maximum radio transmit power of a
- 2543 device in certain regions

- 2544 • Added the ability for the end-device to process several ADRreq commands in a
- 2545 single block in 5.2
- 2546 • Clarified AppKey definitionIntroduced the ResetInd / ResetConf MAC commands
- 2547 • Split Data rate and txpower table in 7.1.3 for clarity
- 2548 • Added DeviceTimeReq/Ans MAC command to class A
- 2549 • Changed Class B time origin to GPS epoch, added BeaconTimingAns description
- 2550 • Aligned all beacons of class B to the same time slot. Class B beacon is now common
- 2551 to all networks.
- 2552 • Separated AppKey and NwkKey to independently derive AppSKeys and NetSKeys.
- 2553 • Separated NetSKeyUp and NetSKeyDnw for roaming
- 2554 •

2555 22.4 Revision 1.1

2556 This section provides an overview of the main changes happening between LoRaWAN1.1
2557 and LoRaWAN1.0.2.

2558 22.4.1 Clarifications

- 2559 ○ Grammatical
- 2560 ○ Normative text used consistently
- 2561 ○ ADR behavior,
 - 2562 • Introduced the concept of ADR command block processing
 - 2563 • TXPower handling
 - 2564 • Default channel re-enabling
 - 2565 • ADR Backoff behavior
- 2566 ○ Default TXPower definition
- 2567 ○ FCnt shall never be reused with the same session keys
- 2568 ○ MAC Commands are discarded if present in both FOpts and Payload
- 2569 ○ Retransmission backoff clarification

2570 22.4.2 Functional modifications

- 2571 ○ FCnt changes
 - 2572 • All counters are 32bits wide , 16bits not supported any more
 - 2573 • Separation of FCntDown into AFCntDown and NFCntDown
 - 2574 ▪ Remove state synchronization requirement from NS/AS
 - 2575 • Remove requirement to discard frames if FCnt gap is greater than MAX_FCNT_GAP
 - 2576 ▪ Unnecessary with 32bit counters
 - 2577 • End-device Frame counters are reset upon the successful processing of a Join-Accept
 - 2578 • ABP device must never reset frame counters
- 2579 ○ Retransmission (transmission without incrementing the FCnt)
 - 2580 • Downlink frames are never retransmitted
 - 2581 • Subsequent receptions of a frame with the same FCnt are ignored
 - 2582 • Uplink retransmissions are controlled by NbTrans (this includes both confirmed and
 - 2583 unconfirmed frames)

- 2584 • A retransmission may not occur until both RX1 and RX2 receive windows have
- 2585 expired
- 2586 • Class B/C devices cease retransmitting a frame upon the reception of a frame in the
- 2587 RX1 window
- 2588 • Class A device cease retransmitting a frame upon the reception of a frame in either
- 2589 the RX1 or RX2 window
- 2590 ○ Key changes
 - 2591 • Added one new root key (separation of cipher function)
 - 2592 ▪ NwkKey and AppKey
 - 2593 • Added new session keys
 - 2594 ▪ NwkSEncKey encrypts payloads where Fport = 0 (MAC command payload)
 - 2595 ▪ AppSKey encrypts payloads where Fport != 0 (Application payloads)
 - 2596 ▪ NwkSIntKey is used to MIC downlink frames
 - 2597 • For downlinks with the ACK bit set, the 2 LSBs of the AFCntUp of the
 - 2598 confirmed uplink which generated the ACK are added to the MIC
 - 2599 calculation
 - 2600 ▪ SNwkSIntKey and FNwkSIntKey are used to MIC uplink frames
 - 2601 • Each is used to calculate 2 separate 16 bit MICs which are combined to a
 - 2602 single 32 bit MIC
 - 2603 • The SNwkSIntKey portion is considered "private" and not shared with a
 - 2604 roaming fNs
 - 2605 • The FNwkSIntKey portion is considered "public" and may be shared with
 - 2606 a roaming fNs
 - 2607 • The private MIC portion now uses the TxDr, TxCh
 - 2608 • For uplinks with the ACK bit set, the 2 LSBs of the FCntDown of the
 - 2609 confirmed downlink which generated the ACK are added to the private
 - 2610 MIC calculation
 - 2611 ▪ Keys fully defined later (section 6)
 - 2612 • Associated MIC and Encrypt changes using new keys
 - 2613 ○ MAC Commands introduced
 - 2614 • TxParamSetupReq/Ans
 - 2615 • DlChannelReq/Ans
 - 2616 • ResetInd/Conf
 - 2617 • ADRParamSetupReq/Ans
 - 2618 • DeviceTimeReq/Ans
 - 2619 • ForceRejoinReq
 - 2620 • RejoinParamSetupReq/Ans
 - 2621 • For the linkADRRReq command
 - 2622 ▪ Value of 0xF is to be ignored for DR or TXPower
 - 2623 ▪ Value of 0 is to be ignored for NbTrans
 - 2624 ○ Activation
 - 2625 • JoinEUI replaces AppEUI (clarification)
 - 2626 • EUI's fully defined
 - 2627 • Root keys defined
 - 2628 ▪ NwkKey
 - 2629 ▪ AppKey
 - 2630 • Additional session keys added (split MIC/Encrypt keys)
 - 2631 ▪ SNwkSIntKeyUp and FNwkSIntKeyUp (split-MIC uplink)
 - 2632 ▪ NwkSIntKeyDown (MIC downlink)
 - 2633 ▪ NwkSEncKey (Encrypt up/down)

- ### 2661 22.4.3 Examples

- 2663

23 Glossary

2664		
2665		
2666	ADR	Adaptive Data Rate
2667	AES	Advanced Encryption Standard
2668	AFA	Adaptive Frequency Agility
2669	AR	Acknowledgement Request
2670	CBC	Cipher Block Chaining
2671	CMAC	Cipher-based Message Authentication Code
2672	CR	Coding Rate
2673	CRC	Cyclic Redundancy Check
2674	DR	Data Rate
2675	ECB	Electronic Code Book
2676	ETSI	European Telecommunications Standards Institute
2677	EIRP	Equivalent Isotropically Radiated Power
2678	FSK	Frequency Shift Keying modulation technique
2679	GPRS	General Packet Radio Service
2680	HAL	Hardware Abstraction Layer
2681	IP	Internet Protocol
2682	LBT	Listen Before Talk
2683	LoRa™	Long Range modulation technique
2684	LoRaWAN™	Long Range Network protocol
2685	MAC	Medium Access Control
2686	MIC	Message Integrity Code
2687	RF	Radio Frequency
2688	RFU	Reserved for Future Usage
2689	Rx	Receiver
2690	RSSI	Received Signal Strength Indicator
2691	SF	Spreading Factor
2692	SNR	Signal Noise Ratio
2693	SPI	Serial Peripheral Interface
2694	SSL	Secure Socket Layer
2695	Tx	Transmitter
2696	USB	Universal Serial Bus

2697 **24 Bibliography**

2698 **24.1 References**

- 2699 [IEEE802154]: IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-
2700 Rate Wireless Personal Area Networks (LR-WPANs), IEEE Std 802.15.4TM-2011 (Revision
2701 of IEEE Std 802.15.4-2006), September 2011.
- 2702 [RFC4493]: The AES-CMAC Algorithm, June 2006.
- 2703 [PHY]: LoRaWAN Regional parameters v1.1, LoRa Alliance
- 2704 [BACKEND]: LoRaWAN backend Interfaces specification v1.0, LoRa Alliance

25 NOTICE OF USE AND DISCLOSURE

Copyright © LoRa Alliance, Inc. (2017). All Rights Reserved.

The information within this document is the property of the LoRa Alliance (“The Alliance”) and its use and disclosure are subject to LoRa Alliance Corporate Bylaws, Intellectual Property Rights (IPR) Policy and Membership Agreements.

Elements of LoRa Alliance specifications may be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights (such a third party may or may not be a member of LoRa Alliance). The Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

This document and the information contained herein are provided on an “AS IS” basis and THE ALLIANCE DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO (A) ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OF THIRD PARTIES (INCLUDING WITHOUT LIMITATION ANY INTELLECTUAL PROPERTY RIGHTS INCLUDING PATENT, COPYRIGHT OR TRADEMARK RIGHTS) OR (B) ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NONINFRINGEMENT.

IN NO EVENT WILL THE ALLIANCE BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF DATA, INTERRUPTION OF BUSINESS, OR FOR ANY OTHER DIRECT, INDIRECT, SPECIAL OR EXEMPLARY, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, IN CONTRACT OR IN TORT, IN CONNECTION WITH THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The above notice and this paragraph must be included on all copies of this document that are made.

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

Note: All Company, brand and product names may be trademarks that are the sole property of their respective owners.