LoRa MAC Specification

N. Sornin (Semtech), M. Luis (Semtech), T. Eirich (IBM), and T. Kramp (IBM)
Release V1.0

1 Introduction

LoRa™ is a wireless modulation for long-range low-power low-data-rate applications developed by Semtech. This document describes the LoRa MAC layer for potentially battery-powered end devices that may be either mobile or mounted at a fixed location. It further motivates the design decisions underlying the LoRa MAC layer to be considered by either network operators, end-device manufacturers, or both to achieve the best performance for a given usage scenario.

LoRa 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** in the backend. Gateways are connected to the network server via standard IP connections while end devices use single-hop LoRa communication to one or many gateways. All communication is generally bi-directional, although uplink communication from an end device to the network server is strongly favored.

Communication between end devices and gateways is spread out among different **frequency channels** and so-called **spreading factors**. In essence, selecting a spreading factor is a trade-off between communication range and data rate whereby communication with different spreading factors does not interfere with each other. Depending on the spreading factor used, LoRa data rates range from 0.3kbps to 40kbps. To maximize both battery life of end devices and overall network capacity, the LoRa network infrastructure and each end device cooperatively manage the spreading factor (i.e., data rate) and RF output for the end device individually by means of an **adaptive data rate** (ADR) scheme.

For time synchronization and the propagation of channel information, gateways periodically broadcast so-called **beacons**. Each beacon minimally contains a set of channels available for random access within a given network and the current GPS time. The broadcasting of beacons is done time-synchronously by all gateways of a network with no interference.

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

- Prior to transmission, the end device uses Listen Before Talk (LBT) to assess that the intended transmission channel is free. A channel is considered free if the instantaneous signal strength measured is smaller than RSSI_FREE_TH. If the channel is not free, the end device changes to another channel and repeats the LBT procedure.
- The end device changes channel in a pseudo-random fashion for every transmission since the resulting frequency diversity makes the system more robust to interferences.

1. Gateways are also known as **concentrators**.
2. End devices are also known as **motes**.

- The end device respects the maximum transmit duty cycle relative to the sub-band used.

  Note: The LoRa MAC enforces a per sub-band transmit duty-cycle limitation that guarantees compliance with the ETSI EN300.220 regulation. The precise way this limitation is implemented is described in Section 5.

A LoRa network distinguishes between two classes of end devices:

- **Bi-directional end devices (Class A):** End devices of class A allow bidirectional communication at the MAC and application layer whereby after each send operation two very short receive windows are opened. The time interval between the end of a send operation and the opening of the first and the second receive window are configurable but the same for all end devices of class A within a given network (CLS2_RECEIVE_DELAY1, CLS2_RECEIVE_DELAY2).
- **Bi-directional end devices with synchronized receive slots (Class B):** End devices of class B allow bidirectional communication at the MAC and application layer whereby both send and receive operations may be scheduled based on the time information contained in the beacons that are broadcasted by all gateways within a network. In addition, like end devices of class A, two receive windows are opened after the end of each send operation.[3]

  Note: In this document, the octet order for all multi-octet fields is little endian.

2 End-Device Activation

Prior to participating in a LoRa network, each end device first has to be personalized and then activated. During activation, the following information is stored within the end device:

- **DevAddr:** A device ID of 32 bits that uniquely identifies the end device.
- **AppEUI:** A globally unique application ID in EUI64 format that uniquely identifies the application provider (i.e., owner) of the end device.
- **NwkSKey:** A device-specific **network session key** used by both the network server and the end device to calculate and verify the MIC of all data messages to ensure data integrity. It is further used to encrypt and decrypt the payload field of MAC-only data messages.
- **AppSKey:** A device-specific **application session key** used by both the network server and the end device to encrypt and decrypt the payload field of application-specific data messages. It may also be used to calculate and verify an application-level MIC to be optionally included in the payload of application-specific data messages.

The DevAddr must be unique in the network where the device operates. Its format is as follows:

Bit#	31..25	24..0
DevAddr	NwkID	NwkAddr

3. The current specification does not describe the scheduling mechanism required for end devices of class B.

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

Activation of an end device can be achieved in two ways. Firstly, over-the-air activation (OTAA) when an end device is deployed or reset; secondly, activation by personalization (ABP) in which the two steps of end-device personalization and activation are done as one step.

2.1 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 the join procedure anew every time it is powered up or reset.

The join procedure required the end device to be personalized with the following information:

- **DevEUI:** a globally unique device ID in EUI64 format
- **AppEUI:** a globally unique application ID in EUI64 format as described before
- **AppKey:** a device-specific AES-128 application key

The **device ID** (DevEUI) identifies each end device globally uniquely. In addition to the AppEUI, the application provider further assigns a **device-specific application key** (AppKey) to the device, most likely by deriving the device-specific key from an application-specific root key exclusively known to and under the control of the application provider.[4] Whenever an end device joins a network via over-the-air activation, the AppKey is used to derive the device-specific sessions key NwkSKey and AppSKey to encrypt and verify network communication and application data.

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

From an end device's point of view, the join procedure consists of two messages exchanged, namely a **join request** and and a **join accept**. The join procedure is always initiated from the end device by sending a join-request message:

Octets	8	8	2
Join Request	AppEUI	DevEUI	DevNonce

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

4. Since all end devices end up with unrelated device-specific application keys, extracting the AppKey from a device only compromises this one device.

sequence of RSSI measurements under the assumption that the quality of randomness fulfills the criteria of true randomness. For each end device, the network server keeps track of the DevNonce values used by the end device in the past and ignores join requests with a DevNonce values seen before from that end device.

> **Note:** This mechanism prevents replay attacks by sending previously recorded join-request messages with the intention of disconnecting the respective end device from the network.

Prior to sending a join request, the end device must listen for a beacon. Beacons are sent out regularly by all gateways and each beacon contains channel information specifying those channels available for random access within a given network. Since join requests are not subject to a specific sending schedule and as such are always sent at random, a join-request message must be sent on one of the network's random-access channels.

> **Note:** An end device may cache the channel information distributed by the beacons of a given network in case it has to rejoin that same network after some loss of connectivity. In this case, the end device need not wait for a beacon before rejoining the network but can rely on the channel information cached.

The MIC value for a join-request message is calculated as follows [RFC4493]:

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

The join-request message is not encrypted.

If an end device is permitted to join a network, the network server will respond to the join request of the end device with a join-accept message. The join-accept message is sent a fixed time after the end of transmitting the join-request message (JOIN_ACCEPT_DELAY). If the join request is not accepted, no response is given to the end device.

Octets	6	4	2
Join Accept	AppNonce	DevAddr	RFU

The join-accept message contains an **application nonce** (AppNonce) of 6 octets and a **device address** (DevAddr) as described above. The AppNonce is a random value or some form of unique ID provided by the network server and used by the end device to derive the two session keys NwkSKey and AppSKey as follows:

NwkSKey = aes128_encrypt(AppKey, AppNonce | DevNonce | pad_{16})
AppSKey = aes128_encrypt(AppKey, DevNonce | AppNonce | pad_{16})

The MIC value for a join-accept message is calculated as follows [RFC4493]:

cmac = aes128_cmac(AppKey, MHDR | AppNonce | DevAddr | RFU)
MIC = *cmac*[0..3]

The join-accept message itself is encrypted with the AppKey as follows:

aes128_decrypt(AppKey, AppNonce | DevAddr | RFU | MIC)

The network server uses an AES decrypt operation to encrypt the message so that the end

device can use an AES encrypt operation to decrypt the message. This way an end device only has to implement AES encrypt but not AES decrypt.

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

2.2 Activation by Personalization

For certain types of networks (e.g., private networks) the over-the-air activation procedure may be considered too complex, though. For such networks, activation by personalization may be an appropriate shortcut at the cost of slightly reduced security and a potentially more complex personalization from an organizational point of view. It also ties an end device to a specific network already at personalization and may prevent roaming. As such, activation by personalization should be used only for end devices staying within the proximity of a specific LoRa network.

Basically, activating an end device during personalization means that the DevAddr and the two session keys NwkSKey and AppSKey are directly personalized into the end device instead of the DevEUI and the AppKey. That is, the end device is equipped with the required information for participating in a specific LoRa network right from the beginning.

> **Note:** The end device still shall await the reception of a first beacon message from the network to learn about the channels available for random access. Only if the set of channels available for random access within the respective network is known beforehand and guaranteed never to change, this information may also be personalized into the end device.

3 MAC Communication

In general, the LoRa terminology distinguishes between uplink and downlink messages. **Uplink messages** are sent by end devices to the network server, relayed by one or many gateways. All uplink messages use the radio packet explicit mode, that is, the LoRa physical header (PHDR) plus a header CRC are always included. The integrity of the payload is further protected by a CRC appended by the radio.

PHY	Preamble	PHDR	CRC	PHYPayload	CRC

Downlink messages are sent by the network server to one or many end-devices, relayed by one or many gateways.[5] All downlink messages use the radio packet explicit mode, that is, the LoRa **physical header** (PHDR) plus a header CRC. No payload integrity check is done at this level to keep messages as short as possible with minimum impact on any duty-

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

cycle limitations of the ISM bands used.

PHY	Preamble	PHDR	CRC	PHYPayload

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

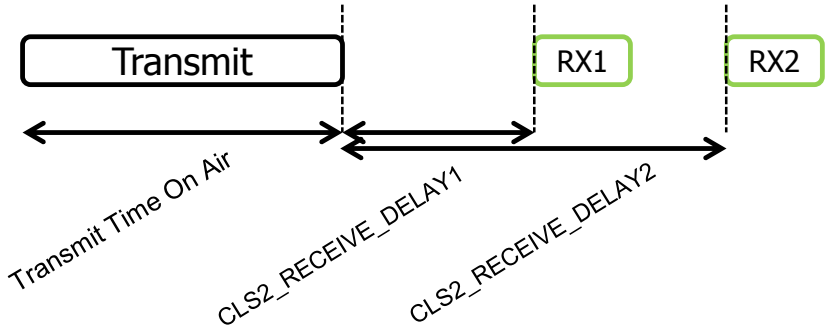


Figure 1: End device receive slot timing.

The first receive window RX1 comes exactly CLS2_RECEIVE_DELAY1 seconds after the end of the uplink modulation. The second slot RX2 comes exactly CLS2_RECEIVE_DELAY2 seconds after the end of the uplink modulation. The length of the receive window is five symbols long and therefore depends on the data rate being used. It is the minimum time required to effectively detect a downlink preamble. If a preamble is detected during one of the RX slot, the radio receiver stays active until the downlink frame is demodulated. If a frame was detected and subsequently demodulated during the first slot, the end-device does not open the second slot.

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

> **Note:** An end device is not allowed to transmit another message before the second receive window of the previous transmission is expired.

> **Note:** For end devices of class 3, scheduled receive windows are not served by the end device if they overlap with the time range starting with uplink send operation and ending at the end of the second receive window following that send operation.

3.1 MAC Payload Formats

All LoRa uplink and downlink messages carry a payload starting with a single-octet **MAC header** (MHDR), followed by a **MAC payload** (MACPayload) of up to 54 octets, and ending with a 4-octet **message integrity code** (MIC).[6]

6. Due to the low data rates of LoRa modulation (0.3kbps up to 40kbps) and the associated long time on air, the physical layer payload size is limited to 64 octets. This way, the maximum air time of a message is less than 2 seconds.

Octets	1	1..59	4
PHYPayload	MHDR	MACPayload	MIC

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

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

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

The LoRa MAC distinguishes between four different MAC message types: **join request**, **join accept**, **unconfirmed data** messages, and **confirmed data** messages. The join-request and join-accept messages are used by the over-the-air activation procedure described in Section 2. Data messages are used to transfer both MAC commands and application data, possibly even together in a single message. A confirmed data message has to be acknowledged by the receiving end of the radio link, whereas an unconfirmed data message does not require an acknowledgment.[7] **Proprietary** messages finally can be used to implement non-standard message formats that are not interoperable and must only be used among devices that have a common understanding of the proprietary extensions.

Major	Description
00	Version 1
01..11	RFU

Note: The Major specifies the format of the messages exchanged in the join procedure (see Section **Error! Reference source not found.**) and the first four bytes of data messages described in Section **Error! Reference source not found.**. 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 out of band (e.g., as part of the device personalization information).

Message integrity is ensured in different ways for different message types and described per message type below and in Section **Error! Reference source not found.**.

The PHY payload of data messages, a so-called **data frame**, contains a **frame header** (FHDR) followed by an optional **port field** (FPort) and an optional **frame payload** (FRMPayload). If present, an FPort value of 0 indicates that the FRMPayload contains MAC commands only; see Section 3.2 for a list of valid MAC commands. FPort values 1..255 are

[7] A detailed timing diagram of the acknowledge mechanism is given in Annex 7.

application-specific.

Octets	7..23	0..1	0..(N-1)
Data Frame	FHDR	FPort	FRMPayload

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

Octets	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

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

One of the main features of a LoRa network is that it allows the end-devices to individually use any of the possible data rates. This feature is used by the LoRa MAC to optimize the data rate of fixed end-devices. Mobile end-devices should use their default data rate because data rate management in a fast moving radio channel is not practical. If the ADR bit is set, the network will control the data rate of the end device through the appropriate MAC commands. If the ADR bit is cleared, the network will not attempt to control the data rate of the end device independently of the received signal quality. The ADR bit may be set and unset by the end device on demand. However, whenever possible, the ADR scheme should be enabled to increase the battery life of the end device and maximize the network capacity.

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

If an end device whose data rate is optimized by the network uses a data rate higher than its default applicative data rate, it periodically needs to validate the network still receives the uplink frames. After each uplink, the device increments an ADR_ACK_CNT counter. After ADR_ACK_LIMIT uplinks (ADR_ACK_CNT == ADR_ACK_LIMIT) without any kind of downlink response, it sets the **ADR acknowledgment request** bit (ADRACKReq). The gateway is required to respond with a downlink frame within the next ADR_ACK_DELAY uplinks, whereby any received downlink following an uplink resets the ADRACK counter. The downlink ACK bit does not need to be set, since any response during the receive slot of the end device indicates that the gateway still receives the uplinks from this device. If no reply is received within the next ADR_ACK_DELAY uplinks (i.e., after a total of ADR_ACK_LIMIT + ADR_ACK_DELAY), the end-device may try to regain connectivity by switching back to its default spreading factor providing longer reach. The ADRACKReq shall not be set if the device uses its default spreading factor 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.

Each end device maintains exactly two frame counters to keep track of the number of data frames sent uplink to the network server (FCntUp) and received by the end device downlink from the network server (FCntDown). The network server in turn maintains the same frame counters for each end device. At initialization, the frame counters on the end device and the

frame counters on the network server for that end device are reset to 0. Subsequently FCntUp and FCntDown are incremented at the sender side by 1 for each data frame sent in the respective direction. At the receiver side, the corresponding counter is kept in sync with the value received provided the value received is larger than the current counter value but in close proximity (considering rollovers), that is, at most MAX_FCNT_GAP data frames got lost. Data frames with a frame counter out of this range shall be silently discarded.

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

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

For a data frame to acknowledge the receipt of a previous data frame, the **acknowledgment** bit (ACK) shall be set.

> **Note:** For confirmed data messages, the receiver shall respond with an acknowledgment. If the sender is an end device, the network will send the acknowledgement exactly CLS2_RECEIVE_DELAY1 or CLS2_RECEIVE_DELAY2 seconds later, using one of the receive windows opened by the end device after a send operation. If the sender is a gateway, the end device transmits an acknowledgment at its own discretion.
>
> End devices that want to keep as little state as possible may transmit an explicit (possibly empty) acknowledgement data message immediately after the reception of a data message requiring a confirmation. Alternatively, an end device may defer the transmission of an acknowledgement to piggyback it with its next data message.
>
> A detailed timing diagram of the acknowledge mechanism is given in Annex 7.

The **frame pending** bit (FPending) is only used in downlink communication, indicating that the gateway has more data pending to be sent and therefore asking the end device to open another receive window as soon as possible. The exact use of FPending bit is described in Annex 7.3.

The **frame-options length** field (FOptsLen) denotes the actual length of FOpts included in the frame. FOpts are MAC commands of a maximum length of 15 octets that are piggybacked onto data frames; see Section 3.2 for a list of valid MAC commands. If FOptsLen is 0, the FOpts field is absent.

The MIC is calculated over the fields

msg = MHDR | FHDR | FPort | FRMPayload

whereby len(*msg*) denotes the length of the message in octets. The MIC is now calculated

as follows [RFC4493]:

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

whereby the block B_0 is defined as follows:

Octets	1	4	1	4	4	1	1
B_0	0x49	4 x 0x00	Dir	DevAddr	FCnt [Up|Down]	0x00	len(*msg*)

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

If a data frame carries a payload, that payload has to be encrypted before message integrity is calculated. The encryption scheme used is based on the generic algorithm described in IEEE 802.15.4/2006 Annex B [IEEE802154] using AES with a key length of 128 bits. The key *K* used depends on the FPort of the data message:

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

The fields encrypted are

pld = FRMPayload

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

Octets	1	5	1	4	4	1	1
A_i	0x01	4 x 0x00	Dir	DevAddr	FCnt [Up|Down]	0x00	*i*

Again, the **direction** field (Dir) is 0 for uplink frames and 1 for downlink frames.

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

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

Encryption and decryption of the payload finally is done by truncating

(*pld* | pad_{16}) xor S

to the first len(*pld*) octets.

3.2 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 exposed to the application, neither to the application server nor onto 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 command sequences are always sent in the clear (not encrypted) and may not exceed 15 octets. MAC commands sent as FRMPayload are always encrypted and may not exceed the maximum FRMPayload length. That is, MAC commands whose content shall not be disclosed to an eavesdropper must be sent in the FRMPayload of a separate data message.

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

CID	Command	Transmitted by		Short Description
		End Device	Gateway	
0x02	LinkCheckReq	x		Used by an end device to validate its connectivity to a network.
0x02	LinkCheckAns		x	Answer to LinkCheckReq command. Contains the received signal power estimation indicating to the end device the quality of reception (link margin).
0x03	LinkADRReq		x	Requests the end device to change data rate, transmit power, or channel.
0x03	LinkADRAns	x		Acknowledges the LinkRateReq.
0x06	DevStatusReq		x	Requests the status of the end device.
0x06	DevStatusAns	x		Returns the status of the end device, namely its battery level, RF duty cycle, RF power, and number of packets transmit.

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

3.2.1 Link Check

With the LinkCheckReq command an end device may validate its connectivity with the network. The command has no payload. If a LinkCheckReq is received by the network server via one or multiple gateways, it responds with a LinkCheckAns command.

Octets	1	1
LinkCheckAns Payload	Margin	GwCnt

The **demodulation margin** (Margin) is an 8-bit unsigned integer in the range of 0..254 indicating the link margin in dB of the last successfully received LinkCheckReq command. A value of 0 means that the packet was demodulated with no margin while a value of 20, for example, means that packet reached the nearest gateway 20dB above the demodulation floor.

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

3.2.2 Link ADR

With the LinkADRReq command, a network server may request an end device to perform a rate adaptation.

Bits	4	4	16
LinkADRReq Payload	DataRate	TXPower	ChMask

The requested **date rate** (DataRate) and **TX output power** (TXPower) are encoded as follows:

DataRate	Configuration
0	LoRa: SF12 / 125kHz
1	LoRa: SF11 / 125kHz
2	LoRa: SF10 / 125kHz
3	LoRa: SF9 / 125kHz
4	LoRa: SF8 / 125kHz
5	LoRa: SF7 / 125kHz
6	LoRa: SF7 / 250kHz
7	FSK: 100kbps
8..15	RFU

TXPower	Configuration
0	20 dBm (if supported)
1	14 dBm
2	11 dBm
3	8 dBm
4	5 dBm
5	2 dBm
6..15	RFU

The **channel mask** (ChMask) encodes the available center frequencies as follows with bit 0 corresponding to the LSB:

Bit#	Center Frequency [Mhz]
0	868.1
1	868.3
2	868.5
3	868.85
4	869.05
5	869.525
6	868.95
7..15	RFU

An end device answers to a LinkADRReq with a LinkADRAns command.

Octets	1
LinkADRAns Payload	Margin

The **margin** (Margin) is the demodulation signal-to-noise ratio in dB for the last successfully received LinkADRReq command. This is a signed integer of 8 bits with a minimum value of -127 and a maximum value of 127. The value -128 is reserved and means the demodulation margin could not be computed.[8]

3.2.3 End-Device Status

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

Octets	1	1
DevStatusAns Payload	Battery	Margin

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

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

The **margin** (Margin) is the demodulation signal-to-noise ratio in dB for the last successfully received DevStatusReq command. This is a signed integer of 8 bits with a minimum value of -127 and a maximum value of 127.

$$\text{Margin} = \text{round}(SNR_{dB} \times 4)$$

The value -128 is reserved and means the demodulation margin could not be computed.[9]

4 Beaconing

Besides relaying messages between end devices and network servers, all gateways participate in providing a time-synchronization mechanisms by sending beacons in regular fixed intervals configurable per network (BEACON_INTERVAL). All beacons are transmitted using SF9/CR1 in radio packet implicit mode, that is, without a LoRa physical header and with no CRC being appended by the radio.

PHY	Preamble	BCNPayload

8. This field can be fetched directly from the RegPktSnrValue register of the SX1272 chip.
9. This field can be fetched directly from the RegPktSnrValue register of the SX1272 chip.

The **beacon payload** (BCNPayload) consists of a mandatory part of 15 octets length and an optional gateway-specific part of 9 octets.

Octets	4	3	2	4	2	7	2
BCNPayload	RFU	NetID	ChMask	Time	CRC	GwID	CRC

The mandatory part contains a **network identifier** (NetID) to uniquely identify the network for which the beacon is sent, a **channel mask** (ChMask) identifying those channels that are available for random access with the network, and a GPS **timestamp** (Time) in seconds. The integrity of the beacon's mandatory part is protected by a CRC.

The optional part provides additional information regarding the gateway sending a beacon and therefore differs for each gateway.

Bits	4	4	24	24
GwID	InfoDesc	Info	Lat	Lng

The **information descriptor** (InfoDesc) describes how the **information** field (Info) shall be interpreted. An InfoDesc value of 0 indicates that the Info field contains the gateway's antenna broadcasting the beacon. For a single-antenna gateway that field is always 0. All other InfoDesc values are RFU. The **latitude** and **longitude** fields (Lat and Lng, respectively) encode the geographical location of the gateway as follows:

- The north-south latitude is encoded using a signed 24 bit word where -2^{23} corresponds to 90° south (the South Pole) and $2^{23}-1$ corresponds to 90° north (the North Pole). The equator corresponds to 0.
- The east-west longitude is encoded using a signed 24 bit word where -2^{23} corresponds to 180° west and $2^{23}-1$ corresponds to 180° east. The Greenwich meridian corresponds to 0.

> **Note:** As mentioned before, all gateways send their beacon at exactly the same point in time (i.e., time-synchronously) so that for the first 15 octets there are no visible on-air collisions for an end device listening to beacons even if the end device simultaneously receives beacons from several gateways. With respect to the optional part, collision occurs but an end device within the proximity of more than one gateway will still be able to decode the strongest beacon with high probability. This way, the end device can gather some basic understanding of its geographical location.

5 ETSI Applicable Bands

According to ETSI, the following bands and sub-bands for 868MHz band operation support wide-band modulation.

Band	Edge Frequencies [Mhz]		Field / Power	Spectrum Access	Bandwidth [Mhz]
g(Note7)	865	868	+6.2dBm / 100kHz	1% or LBT AFA	3Mhz

g(Note7)	865	870	-0.8dBm / 100kHz	0.1% or LBT AFA	5Mhz
g1	868	868.6	14dBm	1% or LBT AFA	600kHz
g2	868.7	869.2	14dBm	0.1% or LBT AFA	500kHz
g3	869.4	869.65	27dBm	10% or LBT AFA	250kHz
g4	869.7	870	7dBm	No req.	300kHz
g4	869.7	870	14dBm	1% or LBT AFA	300kHz

Of these, only 8 channels can be currently allocated and those are spread on g1, g2, and g3 sub-bands as follows:

Modulation	Bandwidth [kHz]	Channel Frequency [Mhz]	FSK Bitrate or LoRa SF / Bitrate	Nb Channels	Subband	Comment
LoRa	250	868.30	SF7 / 10kbps	1	g1	[LC7] Bi
FSK	250	868.30	100kbps	1	g1	[FC1] Bi
LoRa	125	868.10 868.30 868.50	SF7-SF12 / 0.3-5kbps	3	g1	[LC1] Bi [LC2] Bi [LC3] Bi
LoRa	125	868.85 869.05	SF7-SF12 / 0.3-5kbps	2	g2	[LC4] Bi [LC5] Bi
LoRa	125	869.525	SF7-SF12 / 0.3-5kbps	1	g3	[LC6] Bi

In order to access the physical medium the ETSI regulations impose some restrictions such maximum time the transmitter can be on or the maximum time a transmitter can transmit per hour. The ETSI regulations allow the choice of using either a duty-cycle limitation or a so-called **Listen Before Talk Adaptive Frequency Agility** (LBT AFA) transmissions management. The current LoRa MAC specification exclusively uses duty-cycled limited transmissions to comply with the ETSI regulations.

The LoRa MAC enforces a per sub-band duty-cycle limitation. Each time a frame is transmitted in a given sub-band (g1, g2, or g3), the time of emission and the on-air duration of the frame are recorded for this sub-band. The same sub-band cannot be used again

during the next Toff seconds where:

$$Toff_{subband} = \frac{TimeOnAir}{DutyCycle_{subband}}$$

During the unavailable time of a given sub-band, the device may still be able to transmit on another sub-band. If all sub-bands are unavailable, the device has to wait before any further transmission. The device adapts its channel hoping sequence according to the sub-band availability.

Example: A device just transmitted a 0.5sec long frame on channel LC2. The LC2 channel is in sub-band g1 allowing 1% duty-cycle. Therefore the g1 sub-band (i.e., channels LC1, LC2, and LC3) will be unavailable for 50 seconds.

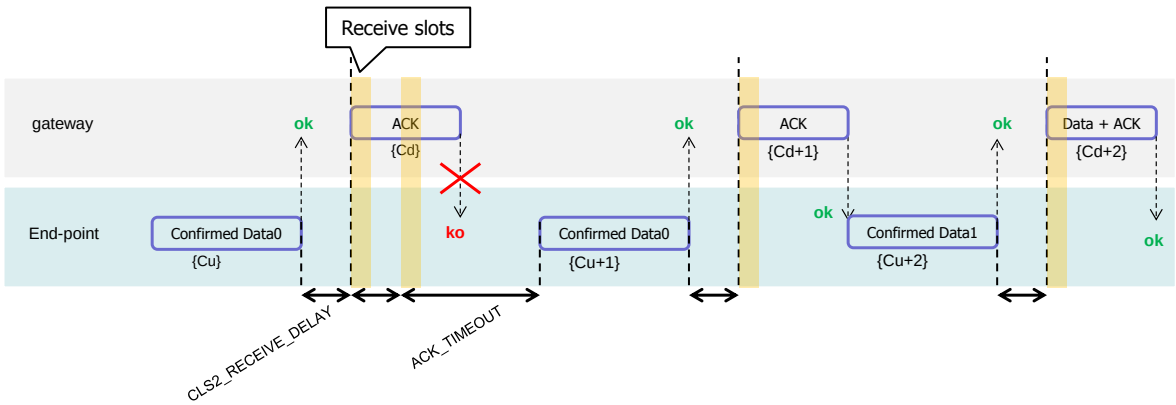
6 Default Settings

BEACON_INTERVAL	128 secs
CLS2_RECEIVE_DELAY1	1 sec
CLS2_RECEIVE_DELAY2	2 secs
JOIN_ACCEPT_DELAY	5 secs
MAX_FCNT_GAP	16384
ADR_ACK_LIMIT	16
ADR_ACK_DELAY	8
RSSI_FREE_TH	-90dBm

7 Annex

7.1 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)



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 frames and generates a downlink frame with the ACK bit set exactly CLS2_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. This downlink frame counter Cd is also derived by adding 1 to the last downlink towards the 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.

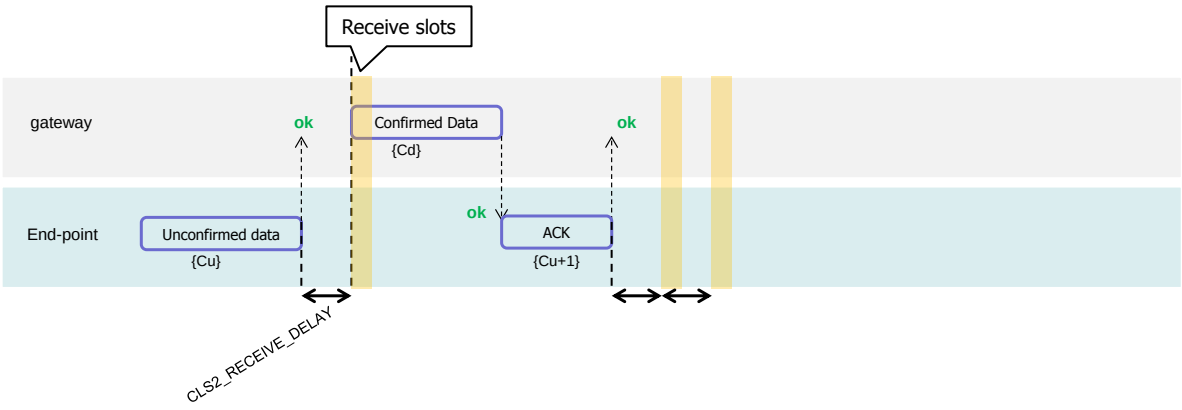
If an end-point does not receive a frame with the ACK bit set in one of the two receive windows immediately following the uplink transmission it can resent the same frame with the same payload and frame counter again. This resend must be done on another channel and must obey the duty cycle limitation as any other normal transmission. This time the network 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. After four failed attempts, there are two options:

- The end device was using a data rate greater than its default data rate, in which case the device switches to its default data rate and restarts.
- The end device was already using its default data rate. The MAC layer returns an error.

7.2 Downlink Diagram for Confirmed Data Messages

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



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 acknowledge, the end device performs a standard channel access procedure (LBT) and 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.

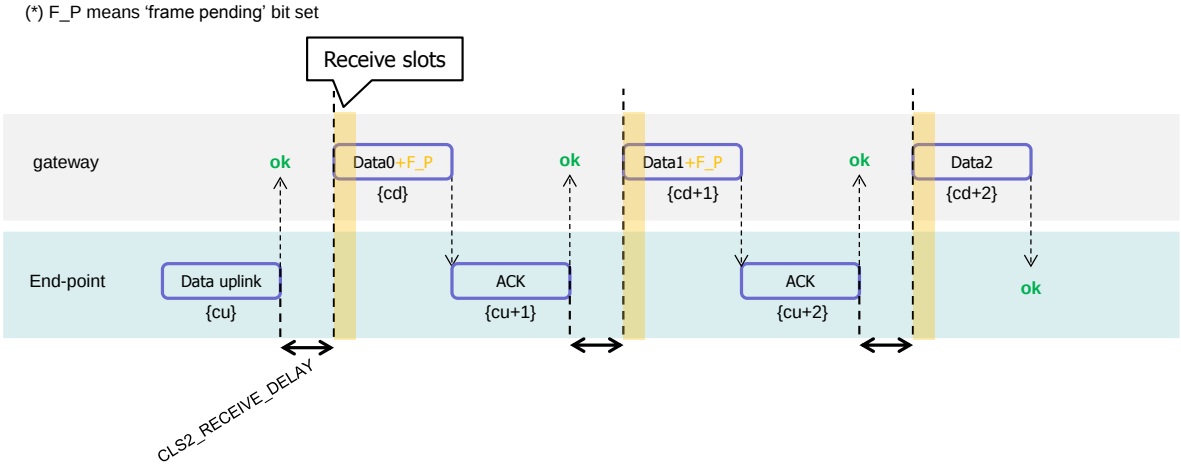
Note: End devices that want to keep as little state as possible may transmit an explicit (possibly empty) acknowledgement data message immediately after the reception of a data message requiring an acknowledgment. Alternatively, an end device may defer the transmission of an acknowledgement to piggyback it with its next data message.

7.3 Downlink Timing for Frame-Pending Messages

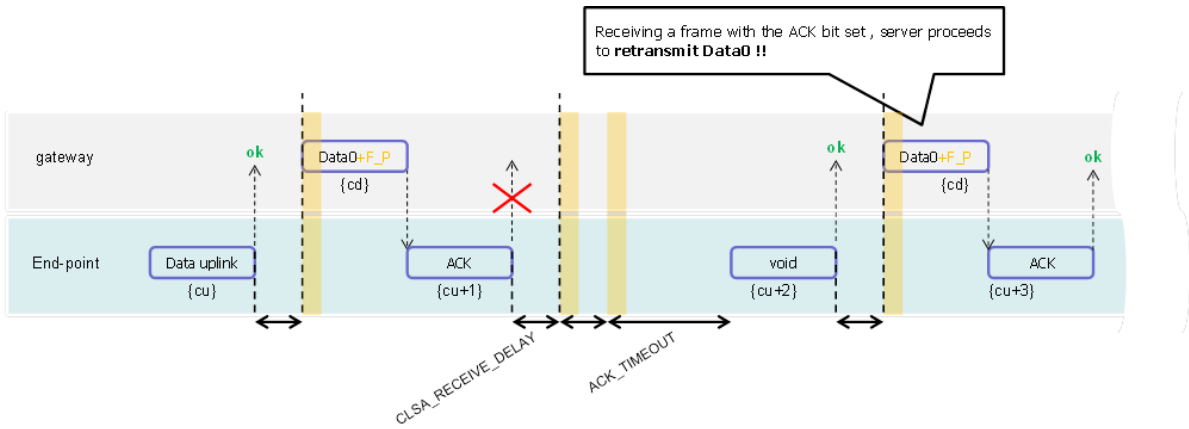
The next diagram illustrates the use of the **frame pending** (FPending) bit on a downlink. The FPending bit can only be set on a downlink frame and informs the end device that the network has several frames pending for him; the bit is ignored for all uplink frames.

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 orthogonal to the acknowledgment scheme.



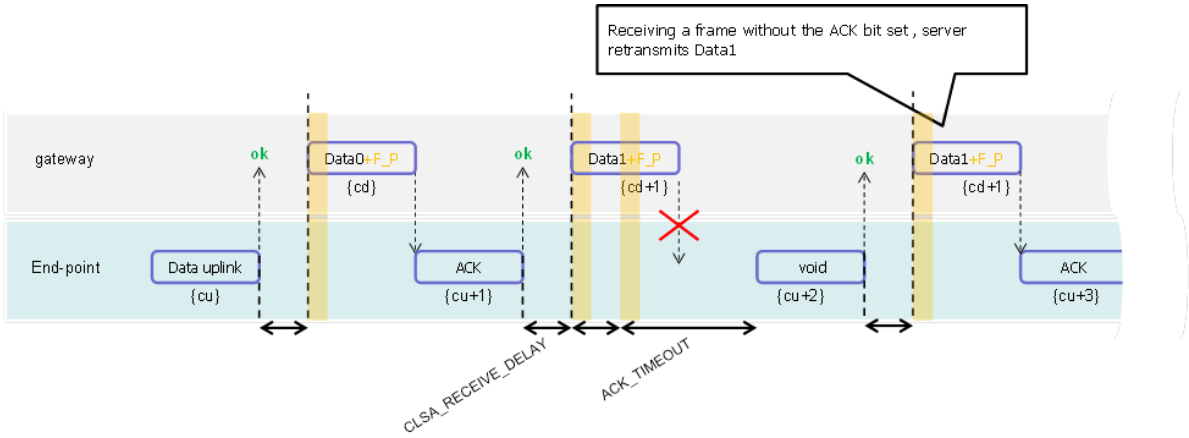
In this example the network has three data frames to transmit to the end device. The frame exchange is initiated by the end-device via a normal "unconfirmed" uplink message on channel A. The network uses the first receive window to transmit the Data0 with the bit FPending set as a confirmed data message. The device acknowledges the reception of the frame by transmitting back an empty frame with the ACK bit set on a new channel B. CLS2_RECEIVE_DELAY1 seconds later, the network transmits the second frame Data1 on channel B, again using a confirmed data message. The end device acknowledges on channel C. The last frame Data2 is transmitted as an unconfirmed data message with the FPending bit cleared. The end-device accordingly does not acknowledge reception of this last frame.



In this example, the first uplink ACK 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 an 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.



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

8 Revision History

8.1 From Version SI0.57 / SI0.60 to SI0.72

- tmerged all the changes from SI0.57 and SI0.60
- changed ACK and FPending diagrams
- introduced CLS2_RECEIVE_DELAY2

8.2 From Version SI0.56 to SI0.57

- swapped Fctrl and Addr field
- added “confirmed data” frame type
- removed the ACKREq bit , obsolete now replaced by confirmed frame
- moved ADR bit from MHDR to Fctrl field
- moved the “Join procedure” to a dedicated section
- changed the value of MAX_FCNT_GAP to 16384

8.3 From Version SI.055 to SI0.56

- changed MIC calculation scheme to CMAC
- changed counter fields in B_0 and A_i to one octet

8.4 From Version SI0.54 to SI0.55

- re-wrote the ADRACKReq bit description and associated mechanisms in **Error! eference source not found.**
- added the ACK_TIMEOUT , ADR_ACK_LIMIT and ADR_ACK_DELAY constants
- added timing diagram annex
- defined exact timing of the acknowledge mechanism
- described the “frame pending” mechanism

8.5 From Version SI0.53 to SI0.54

- in the MIC calculation of the join request, xor is done before encrypting
  From Version SI0.52 to SI0.53
- in data messages, the FPort is not encrypted
- fixed reference to 802.15.4/2006

8.6 From Version SI0.51 to SI0.52

- fixed a couple of typos

8.7 From Version SI0.5 to SI0.51

- moved from UTC to GPS time in seconds to avoid leap seconds in UTC time

8.8 From Version S1.6 to SI0.5

- moved the ADR bit from FHDR to MHDR
- moved the ACK bit from MHDR to FHDR
- changed acknowledgment scheme to use one additional (optional) octet
- slightly changed the format of short addresses
- changed meaning of the More bit
- explicitly allow sequences of MAC commands
- require end devices to listen for a beacon before joining
- removed all references to multicast frames for now
- removed all references to scheduled operation for now
- specified class 1 devices as send-only on the application-level
- some cleanup of terminology

9 Glossary

ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
AFA	Adaptive Frequency Agility
AR	Acknowledgement Request
CBC	Cipher Block Chaining
CMAC	Cipher-based Message Authentication Code
CR	Coding Rate
CRC	Cyclic Redundancy Check
ETSI	European Telecommunications Standards Institute

GPRS	General Packet Radio Service
HAL	Hardware Abstraction Layer
IP	Internet Protocol
LBT	Listen Before Talk
LoRa	Long Range
MAC	Medium Access Control
MIC	Message Integrity Code
RF	Radio Frequency
Rx	Receiver
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal Noise Ratio
SPI	Serial Peripheral Interface
SSL	Secure Socket Layer
Tx	Transmitter
USB	Universal Serial Bus

10 Bibilography

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

[SX1272ETSI]: SX1272/3 ETSI Compliance, Revision 1.0, April 2013, (C) 2013 Semtech Corp.

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