# LoRaWAN协议说明书

# 第1章 介绍

本文档描述了LoRaWAN网络协议，是针对电池供电的终端设备(不管移动还是固定位置)进行优化的一套网络协议。

LoRaWAN网络通常采用星型拓扑结构，由拓扑中的网关来转发终端与后台网络服务器间的消息。网关通过标准IP连接来接入网络服务器，而终端则通过单跳的LoRa或者FSK来和一个或多个网关通讯。虽然主要传输方式是终端上行传输给网络服务器，但所有的传输通常都是双向的。

终端和网关间的通讯被分散到不同的信道频点和数据速率上。数据速率的选择需要权衡距离和消息时长两个因素，使用不同数据速率的设备互不影响。LoRa的数据速率范围可以从 0.3kbps 到 50kbps。为了最大程度地延长终端的电池寿命和扩大网络容量，LoRa网络使用速率自适应(ADR)机制来独立管理每个终端的速率和RF输出。

虽然每个设备可以在任意信道，任意时间，发送任意数据，但需要注意遵守如下规定：

·终端的每次传输都使用伪随机方式来改变信道。频率的多变使得系统具有更强的抗干扰能力。

·终端要遵守相应频段和本地区的无线电规定中的发射占空比要求。

·终端要遵守相应频段和本地区的无线电规定中的发射时长要求。

*注：发射占空比，意思是发射时长占总时长的比例。按照无线电规定，每个设备不能疯狂发射霸占信道，总得给别人一点机会。*

这份文档主要讲述协议细节，一些基于各地区规定的操作参数，例如：发射占空比和发射时长等，在另一份文档[LoRaWAN地区参数]中做具体描述。将这份文档分开，是为了加入新地区参数时不影响基础的协议规范。

## 1.1 LoRaWAN Classes

所有的LoRaWAN设备都必须至少实现本文档描述的 Class A 功能。另外也可以实现本文档中描述的 Class B 和 Class C 及后续将定义的可选功能。不管怎么样，设备都必须兼容 Class A。

## 1.2 文档约定

MAC命令的格式写作LinkCheckReq(粗斜体)，位和位域的格式写作 FRMPayload (粗体)，常量的格式写作 RECEIVE\_DELAY1，变量的格式写作 N。

在本文档中，所有多字节字段的字节序均采用小端模式。

EUI 是8字节字段，采用小端模式传输。

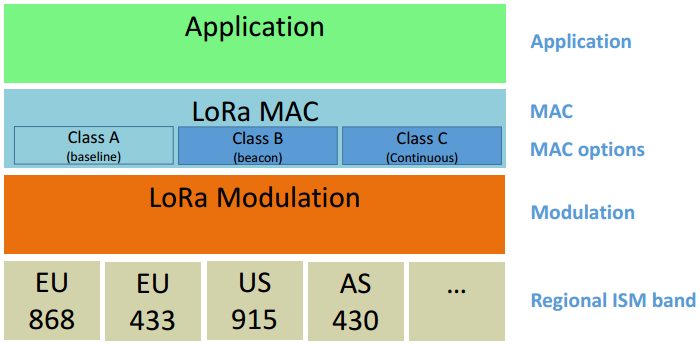
默认所有RFU保留位都设为0。

# **第2章 LoRaWAN Classes 类型介绍**

LoRa 是由Semtech面向长距离、低功耗、低速率应用而开发的无线调制技术。本文档中，将 Class A 基础上实现了更多功能的设备称为“更高 class 终端”。

## 2.1 LoRaWAN Classes

LoRa网络包含基础LoRaWAN（称之为Class A）和可选功能（Class B，Class C）:

图1.LoRaWAN Classes

双向传输终端(Class A)： Class A 的终端在每次上行后都会紧跟两个短暂的下行接收窗口，以此实现双向传输。传输时隙是由终端在有传输需要时安排，附加一定的随机延时(即ALOHA协议)。这种Class A 操作是最省电的，要求应用在终端上行传输后的很短时间内进行服务器的下行传输。服务器在其他任何时间进行的下行传输都得等终端的下一次上行。

划定接收时隙的双向传输终端(Class B)： Class B 的终端会有更多的接收时隙。除了Class A 的随机接收窗口，Class B 设备还会在指定时间打开别的接收窗口。为了让终端可以在指定时间打开接收窗口，终端需要从网关接收时间同步的信标 Beacon。这使得服务器可以知道终端正在监听。

最大化接收时隙的双向传输终端(Class C)：Class C 的终端基本是一直打开着接收窗口，只在发送时短暂关闭。Class C 的终端会比 Class A 和 Class B更加耗电，但同时从服务器下发给终端的时延也是最短的。

## 2.2 文档范围

这份LoRaWAN协议还描述了与 Class A 不同的其他 Class 的额外功能。更高 Class 的终端必须满足 Class A 定义的所有功能。

*注意：物理层帧格式，MAC帧格式，以及协议中更高 class 和 Class A 相同的内容都写在了 Class A 部分，避免内容重复。*

# **第3章 PHY 帧格式**

LoRa 有上行消息和下行消息。

## 3.1 上行消息

上行消息是由终端发出，经过一个或多个网关转发给网络服务器。

上行消息使用 LoRa 射频帧的严格模式，消息中含有 PHDR 和 PHDR\_CRC 。 载荷有CRC校验来保证完整性。

PHDR，PHDR\_CRC及载荷 CRC 域都通过射频收发器加入。

**上行 PHY:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Preamble** | **PHDR** | **PHDR\_CRC** | **PHYPayload** | **CRC** |

图2.上行PHY帧格式

## 3.2 下行消息

下行消息是由网络服务器发出，经过单个网关转发给单个终端。

下行消息使用射频帧的严格模式，消息中包含 PHDR 和 PHDR\_CRC。

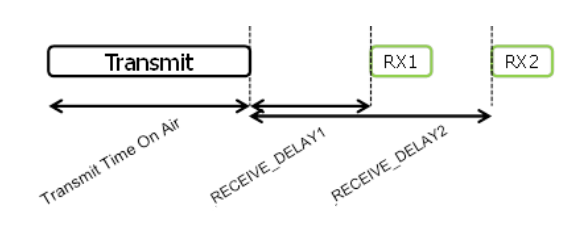
**下行 PHY:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Preamble** | **PHDR** | **PHDR\_CRC** | **PHYPayload** |

图3.下行PHY帧格式

## 3.3 接收窗口

每个上行传输后终端都要开两个短的接收窗口。接收窗口开始时间的规定，是以传输结束时间为参考。

 图4.终端接收时隙的时序图

### 3.3.1 第一接收窗口的信道，数据速率和启动。

第一接收窗口 RX1 使用的频率和上行频率有关，使用的速率和上行速率有关。RX1 是在上行调制结束后的 RECEIVE\_DELAY1 秒打开。上行和 RX1 时隙下行速率的关系是按区域规定，详细描述在[LoRaWAN地区参数]文件中。默认第一窗口的速率是和最后一次上行的速率相同。

### 3.3.2 第二接收窗口的信道，数据速率和启动。

第二接收窗口 RX2 使用一个固定可配置的频率和数据速率，在上行调制结束后的 RECEIVE\_DELAY2 秒打开。频率和数据速率可以通过 MAC 命令(见第5章)。默认的频率和速率是按区域规定，详细描述在[LoRaWAN地区参数]文件中。

### 3.3.3 接收窗口的持续时间

接收窗口的长度至少要让终端射频收发器有足够的时间来检测到下行的前导码。

### 3.3.4 接收方在接收窗口期间的处理

如果在任何一个接收窗口中检测到前导码，射频收发器需要继续激活，直到整个下行帧都解调完毕。如果在第一接收窗口检测到数据帧，且这个数据帧的地址和MIC校验通过确认是给这个终端，那终端就不必开启第二个接收窗口。

### 3.3.5 网络发送消息给终端

如果网络想要发一个下行消息给终端，它会精确地在两个接收窗口的起始点发起传输。

### 3.3.6 接收窗口的重要事项

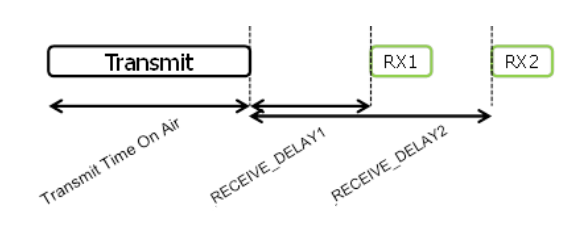
终端在第一或第二接收窗口收到下行消息后，或者在第二接收窗口阶段，不能再发起另一个上行消息。

### 3.3.7 其他协议的收发处理

节点在LoRaWAN收发窗口阶段可以收发其他协议，只要终端能满足当地要求以及兼容LoRaWAN协议。

## 3.4 梳理解析

LoRaWAN第3章，主要是讲了接收窗口这回事，只要记住张图就行。

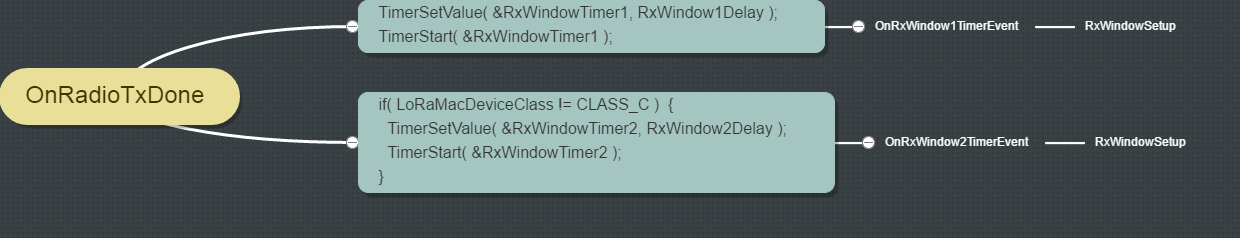


目前RX1一般是在上行后1秒开始，RX2是在上行后2秒开始。

## 3.5 源码分析

### 3.5.1 源码流程

在梳理这章节的对应代码时，自己手动做了张思维导图。有时是这样，代码再有层次感，也不及一个图。好，请收下。



### 3.5.2 发送完成就开始RX1和RX2延时

static void OnRadioTxDone( void )

{

...

// Setup timers

if( IsRxWindowsEnabled == true )

{

TimerSetValue( &RxWindowTimer1, RxWindow1Delay );

TimerStart( &RxWindowTimer1 );

if( LoRaMacDeviceClass != CLASS\_C )

{

TimerSetValue( &RxWindowTimer2, RxWindow2Delay );

TimerStart( &RxWindowTimer2 );

}

if( ( LoRaMacDeviceClass == CLASS\_C ) || ( NodeAckRequested == true ) )

{

TimerSetValue( &AckTimeoutTimer, RxWindow2Delay + ACK\_TIMEOUT + randr( -ACK\_TIMEOUT\_RND, ACK\_TIMEOUT\_RND ) );

TimerStart( &AckTimeoutTimer );

}

}

...

}

### 3.5.3 接收窗口的射频处理

从上面一步，我们已经清晰的知道，对应的处理肯定是在OnRxWindow1TimerEvent和OnRxWindow2TimerEvent中。   
 这两个接收窗口的处理，会对速率和信道进行设置，按照[LoRaWAN协议中文版\_配套文件地区参数(物理层)](http://blog.csdn.net/iotisan/article/details/55056092" \t "_blank)中对各地区的要求分别进行处理。

比如这个470的处理，对上行信道对48取余得到下行信道。

RxWindowSetup( LORAMAC\_FIRST\_RX1\_CHANNEL + ( Channel % 48 ) \* LORAMAC\_STEPWIDTH\_

# 第4章 MAC帧格式

LoRa所有上下行链路消息都会携带PHY载荷，PHY载荷以1字节MAC头(MHDR)开始，紧接着MAC载荷(MACPayload)，最后是4字节的MAC校验码(MIC)。

**射频PHY层：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Preamble** | **PHDR** | **PHDR\_CRC** | **PHYPayload** | **CRC** |

图5.射频PHY结构(注意 CRC只有上行链路消息中存在)

**PHYPayload：**

|  |  |  |
| --- | --- | --- |
| **MHDR** | **MACPayload** | **MIC** |

**或者**

|  |  |  |
| --- | --- | --- |
| **MHDR** | **Join-Request** | **MIC** |

**或者**

|  |  |  |
| --- | --- | --- |
| **MHDR** | **Join-Response** | **MIC** |

图6.PHY载荷结构

**MACPayload：**

|  |  |  |
| --- | --- | --- |
| **FHDR** | **FPort** | **FRMPayload** |

图7.MAC载荷结构

**FHDR：**

|  |  |  |  |
| --- | --- | --- | --- |
| **DevAddr** | **FCtrl** | **FCnt** | **FOpts** |

图8.帧头结构

图9.LoRa帧格式元素(即图5~8)

## 4.1 MAC层(PHYPayload)

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

MACPayload字段的最大长度M，在第6章有详细说明。

## 4.2 MAC头(MHDR字段)

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

MAC头中指定了消息类型(MType)和帧编码所遵循的LoRaWAN规范的主版本号(Major)。

### 4.2.1 消息类型(MType位字段)

LoRaWAN定义了六个不同的MAC消息类型：join request, join accept, unconfirmed data up/down, 以及 confirmed data up/down。

|  |  |
| --- | --- |
| **MType** | **描述** |
| 000 | Join Request |
| 001 | Join Accept |
| 010 | Unconfirmed Data Up |
| 011 | Unconfirmed Data Down |
| 100 | Confirmed Data Up |
| 101 | Confirmed Data Down |
| 110 | RFU |
| 111 | Proprietary |

表1.MAC消息类型

#### 4.2.1.1 Join-request and join-accept 消息

join-request和join-accept都是用在空中激活流程中，具体见章节6.2

#### 4.2.1.2 Data messages

Data messages 用来传输MAC命令和应用数据，这两种命令也可以放在单个消息中发送。   
Confirmed-data message 接收者需要应答。   
Unconfirmed-data message 接收者则不需要应答。   
Proprietary messages 用来处理非标准的消息格式，不能和标准消息互通，只能用来和具有相同拓展格式的消息进行通信。

不同消息类型用不同的方法保证消息一致性，下面会介绍每种消息类型的具体情况。

### 4.2.2 数据消息的主版本(Major位字段)

|  |  |
| --- | --- |
| **Major位字段** | **描述** |
| 00 | LoRaWAN R1 |
| 01..11 | RFU |

表2.Major列表

*注意：Major定义了激活过程中(join procedure)使用的消息格式（见章节6.2）和MAC Payload的前4字节（见第4章）。终端要根据不同的主版本号实现不同最小版本的消息格式。终端使用的最小版本应当提前通知网络服务器。*

## 4.3 MAC载荷(MACPayload)

MAC载荷，也就是所谓的“数据帧”，包含：帧头（FHDR）、端口（FPort）以及帧载荷(FRMPayload），其中端口和帧载荷是可选的。

### 4.3.1 帧头(FHDR)

FHDR是由终端短地址(DevAddr)、1字节帧控制字节(FCtrl)、2字节帧计数器(FCnt)和用来传输MAC命令的帧选项(FOpts，最多15个字节)组成。

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

FCtrl在上下行消息中有所不同，**下行消息**如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Bit#** | 7 | 6 | 5 | 4 | [3..0] |
| **FCtrl bits** | ADR | ADRACKReq | ACK | FPending | FOptsLen |

**上行消息**如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Bit#** | 7 | 6 | 5 | 4 | [3..0] |
| **FCtrl bits** | ADR | ADRACKReq | ACK | RFU | FOptsLen |

#### 4.3.1.1 帧头中自适应数据速率的控制(ADR, ADRACKReq in FCtrl)

LoRa网络允许终端采用任何可能的数据速率。LoRaWAN协议利用该特性来优化固定终端的数据速率。这就是自适应数据速率(Adaptive Data Rate (ADR))。当这个使能时，网络会优化使得尽可能使用最快的数据速率。

移动的终端由于射频环境的快速变化，数据速率管理就不再适用了，应当使用固定的数据速率。

如果ADR的位字段有置位，网络就会通过相应的MAC命令来控制终端设备的数据速率。如果ADR位没设置，网络则无视终端的接收信号强度，不再控制终端设备的数据速率。ADR位可以根据需要通过终端及网络来设置或取消。不管怎样，ADR机制都应该尽可能使能，帮助终端延长电池寿命和扩大网络容量。

*注意：即使是移动的终端，可能在大部分时间也是处于非移动状态。因此根据它的移动状态，终端也可以请求网络使用ADR来帮助优化数据速率。*

如果终端被网络优化过的数据速率高于自己默认的数据速率，它需要定期检查下网络仍能收到上行的数据。每次上行帧计数都会累加(是针对于每个新的上行包，重传包就不再增加计数)，终端增加 ADR\_ACK\_CNT 计数。如果直到ADR\_ACK\_LIMIT次上行(ADR\_ACK\_CNT >= ADR\_ACK\_LIMIT)都没有收到下行回复，它就得置高ADR应答请求位(ADRACKReq)。 网络必须在规定时间内回复一个下行帧，这个时间是通过ADR\_ACK\_DELAY来设置，上行之后收到任何下行帧就要把ADR\_ACK\_CNT的计数重置。当终端在接收时隙中的任何回复下行帧的ACK位字段不需要设置，表示网关仍在接收这个设备的上行帧。如果在下一个ADR\_ACK\_DELAY上行时间内都没收到回复(例如，在总时间ADR\_ACK\_LIMIT+ADR\_ACK\_DELAY之后)，终端必须切换到下一个更低速率，使得能够获得更远传输距离来重连网络。终端如果在每次ADR\_ACK\_LIMIT到了之后依旧连接不上，就需要每次逐步降低数据速率。如果终端用它的默认数据速率，那就不需要置位ADRACKReq，因为无法帮助提高链路距离。

*注意：不要ADRACKReq立刻回复，这样给网络预留一些余量，让它做出最好的下行调度处理。*

*注意：上行传输时，如果 ADR\_ACK\_CNT >= ADR\_ACK\_LIMIT 并且当前数据速率比设备的最小数据速率高，就要设置 ADRACKReq，其它情况下不需要。*

#### 4.3.1.2 消息应答位及应答流程(ACK in FCtrl)

收到confirmed类型的消息时，接收端要回复一条应答消息(应答位ACK要进行置位)。如果发送者是终端，网络就利用终端发送操作后打开的两个接收窗口之一进行回复。如果发送者是网关，终端就自行决定是否发送应答。   
 应答消息只会在收到消息后回复发送，并且不重发。

*注意：为了让终端尽可能简单，尽可能减少状态，在收到confirmation类型需要确认的数据帧，需要立即发送一个严格的应答数据帧。或者，终端会延迟发送应答，在它下一个数据帧中再携带。*

#### 4.3.1.3 重传流程

当需要应答却没收到应答时就会进行重发，重发的个数由终端自己定，可能每个终端都不一样，这个参数也可以由网络服务器来设置调整。

*注意：一些应答机制的示例时序图在第18章中有提供。*

*注意：如果终端设备重发次数到达了最大值，它可以降低数据速率来重连。至于后面是否再重发还是说丢弃不管，都取决于终端自己。*

*注意：如果网络服务器重发次数到达了最大值，它就认为该终端掉线了，直到它再收到终端的消息。一旦和终端设备的连接出现问题时，要不要重发都取决于网络服务器自己。*

*注意：在重传期间的数据速率回退的建议策略在章节18.4中有描述。*

#### 4.3.1.4 帧挂起位(FPending in FCtrl 只在下行有效)

帧挂起位(FPending)只在下行交互中使用，表示网关还有挂起数据等待下发，需要终端尽快发送上行消息来再打开一个接收窗口。

FPending的详细用法在章节18.3。

#### 4.3.1.5 帧计数器(FCnt)

每个终端有两个计数器跟踪数据帧的个数，一个是上行链路计数器（FCntUp），由终端在每次上行数据给网络服务器时累加；另一个是下行链路计数器（FCntDown），由服务器在每次下行数据给终端时累计。网络服务器为每个终端跟踪上行帧计数及产生下行帧计数。终端入网成功后，终端和服务端的上下行帧计数同时置0。每次发送消息后，发送端与之对应的 FCntUp 或 FCntDown 就会加1。*接收方会同步保存接收数据的帧计数，对比收到的计数值和当前保存的值，如果两者相差小于 MAX\_FCNT\_GAP （要考虑计数器滚动），接收方就按接收的帧计数更新对应值。如果两者相差大于 MAX\_FCNY\_GAP 就说明中间丢失了很多数据，这条以及后面的数据就被丢掉。*

LoRaWAN的帧计数器可以用16位和32位两种，节点上具体执行哪种计数，需要在带外通知网络侧，告知计数器的位数。   
 如果采用16位帧计数，FCnt字段的值可以使用帧计数器的值，此时有需要的话通过在前面填充0（值为0）字节来补足；如果采用32位帧计数，FCnt就对应计数器32位的16个低有效位(上行数据使用上行FCnt，下行数据使用下行FCnt)。

终端在相同应用和网络密钥下，不能重复用相同的FCntUp数值，除非是重传。

**4.3.1.6 帧可选项(FOptsLen in FCtrl, FOpts)** FCtrl字节中的FOptsLen位字段描述了整个帧可选项(FOpts)的字段长度。

FOpts字段存放MAC命令，最长15字节，详细的MAC命令见章节4.4。

如果FOptsLen为0，则FOpts为空。在FOptsLen非0时，则反之。如果MAC命令在FOpts字段中体现，port0不能用(FPort要么不体现，要么非0)。

MAC命令不能同时出现在FRMPayload和FOpts中，如果出现了，设备丢掉该组数据。

### 4.3.2 端口字段(FPort)

如果帧载荷字段不为空，端口字段必须体现出来。端口字段有体现时，若FPort的值为0表示FRMPayload只包含了MAC命令；具体见章节4.4中的MAC命令。FPort的数值从1到223(0x01..0xDF)都是由应用层使用。FPort的值从224到255(0xE0..0xFF)是保留用做未来的标准应用拓展。

|  |  |  |  |
| --- | --- | --- | --- |
| **Size(bytes)** | 7..23 | 0..1 | 0..N |
| **MACPayload** | FHDR | FPort | FRMPayload |

N是应用程序载荷的字节个数。N的有效范围具体在第7章有定义。

N应该小于等于：   
 N <= M - 1 - (FHDR长度)

*注：M是MAC载荷的最大长度。*

### 4.3.3 MAC帧载荷加密(FRMPayload)

如果数据帧携带了载荷，FRMPayload必须要在MIC计算前进行加密。   
加密机制是采用IEEE802.15.4/2006的AES128[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)。

默认的，加密和加密由LoRaWAN层来给所有的FPort来执行。如果加密/解密由应用层来做更方便的话，也可以在LoRaWAN层之上给特定FPorts来执行，除了端口0。具体哪个节点的哪个FPort在LoRaWAN层之外要做加解密，必须要和服务器通过out-of-band信道来交互(见第19章)。

#### 4.3.3.1 LoRaWAN的加密

密钥K根据不同的FPort来使用：

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

表3: FPort列表

具体加密是这样：   
 pld = FRMPayload   
 对于每个数据帧，算法定义了一个块序列Ai，i从1到k，

其中，k = ceil( len(pld) / 16 )：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Size(bytes)** | 1 | 4 | 1 | 4 | 4 | 1 | 1 |
| **Ai** | 0x01 | 4 x 0x00 | Dir | DevAddr | FCntUp or FCntDown | 0x00 | i |

方向字段(Dir)在上行帧时为0，在下行帧时为1.   
 块Ai通过加密，得到一个由块Si组成的序列S。

Si = aes128\_encrypt(K, Ai) for i = 1..k   
 S = S1 | S2 | .. | Sk

通过异或计算对payload进行加解密。

**4.3.3.2 LoRaWAN层之上的加密** 如果LoRaWAN之上的层级在已选的端口上(但不能是端口0，这是给MAC命令保留的)提供了预加密的FRMPayload给LoRaWAN，LoRaWAN则不再对FRMPayload进行修改，直接将FRMPayload从MACPayload传到应用层，以及从应用层传到MACPayload。

## 4.4 消息校验码(MIC)

消息检验码要计算消息中所有字段。   
 msg = MHDR **|** FHDR **|** FPort **|** FRMPayload

MIC是按照[RFC4493]来计算：

cmac = aes128\_cmac(NwkSKey, B0 ***|*** msg)   
 MIC = cmac[0..3]

块B0的定义如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Size(**  **bytes)** | 1 | 4 | 1 | 4 | 4 | 1 | 1 |
| **B0** | 0x49 | 4 x 0x00 | Dir | DevAddr | FCntUp or FCntDown | 0x00 | len(msg) |

方向字段(Dir)在上行帧时为0，在下行帧时为1.

LoRaWAN第4章，主要讲述了MAC帧格式，对所有涉及的字段都做了解释。

千言万语汇成一句话，哦不，汇成一个表。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **数据帧头** |  |  |  |  | DevAddr | FCtrl | FCnt | FOpts |  |  |  |  |
| **数据帧** | Preamble | PHDR | PHDR\_CRC | MHDR | FHDR | | | | FPort | FRMPayload | MI  C | CRC |
| **MA**  **C层** | Preamble | PHDR | PHDR\_CRC | MHDR | MACPayload | | | | | | MI  C | CRC |
| **P**  **H**  **Y层** | Preamble | PHDR | PHDR\_CRC | PHYPayload | | | | | | | | CRC |

好了，帧格式是大家随手都能看到的东西，本尊作为IoT小能手，如果不能提出一些稍有深度的信息增量，就对不起这个称号了。所以，有些协议设计层面的心得要分享下：

特别酷的ADR(速率自适应)机制   
 这个章节中最亮眼的莫过于速率自适应机制，简直是为LoRa网络量身定做的：*一旦使能了FCtrl中的ADR位，距离近信号好的节点用高速率，距离远信号弱的节点用低速率，不小心被调高了速率，则自动降下来。这样，尽可能地提高了传输速率，也有效提高了网络容量。*我已经见过不少厂家，拿这个协议的公知特点当产品卖点了。

可同时携带数据和命令的MAC帧。一般来说，应用除了数据，出于管理需要，肯定还会涉及命令。比如基站要查询节点状态，或者节点要请求变更信道等。所以LoRaWAN协议设计上利用FOpts把数据和命令揉在一个MAC帧里，这样可以提高交互效率，有效地降低功耗。这在寸土寸金，哦不，寸库仑(电量单位)寸金的物联网应用中，是一个很有必要的设计。

## 4.5 源码解析

这章的处理基本都在 \src\mac\LoRaMac.c 中，下面按照MAC帧格式的字段逐个解析下。

### 4.5.1 MAC层MHDR

在LoRaWAN的数据API中处理了MHDR，这个字段内容比较少，就按需选择了消息类型是confirm还是unconfirm。   
 另外在管理API中的Join-Req的消息类型。

具体可见 LoRaMacMcpsRequest() 和 LoRaMacMlmeRequest() 这两个函数。

### 4.5.2 MACPayload

MACPayload 的组帧都在 PrepareFrame() 这个函数中处理，将macHdr和MACPayload的FCtrl、FPort、FRMPayload都传递进去，完成整个MAC层的数据组帧。

LoRaMacBuffer就存放了MACPayload的数据，这个变量的组帧和协议字段定义是一一对应。MACPayload的组帧处理，在大流程上是对join和数据两种类型的帧分别处理，用两个case分开。

为了方便阅览，我把函数代码框架提炼了出来。

LoRaMacStatus\_t PrepareFrame( LoRaMacHeader\_t \*macHdr,

LoRaMacFrameCtrl\_t \*fCtrl, uint8\_t fPort,

void \*fBuffer, uint16\_t fBufferSize )

{

switch( macHdr->Bits.MType )

{

case FRAME\_TYPE\_JOIN\_REQ:

...// 省略

break;

case FRAME\_TYPE\_DATA\_CONFIRMED\_UP:

NodeAckRequested = true;

//Intentional falltrough

case FRAME\_TYPE\_DATA\_UNCONFIRMED\_UP:

...

fCtrl->Bits.AdrAckReq = AdrNextDr( fCtrl->Bits.Adr, true, &LoRaMacParams.ChannelsDatarate );

...

if( SrvAckRequested == true )

{

SrvAckRequested = false;

fCtrl->Bits.Ack = 1;

}

LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr ) & 0xFF;

LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 8 ) & 0xFF;

LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 16 ) & 0xFF;

LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 24 ) & 0xFF;

LoRaMacBuffer[pktHeaderLen++] = fCtrl->Value;

LoRaMacBuffer[pktHeaderLen++] = UpLinkCounter & 0xFF;

LoRaMacBuffer[pktHeaderLen++] = ( UpLinkCounter >> 8 ) & 0xFF;

// Copy the MAC commands which must be re-send into the MAC command buffer

memcpy1( &MacCommandsBuffer[MacCommandsBufferIndex], MacCommandsBufferToRepeat, MacCommandsBufferToRepeatIndex );

MacCommandsBufferIndex += MacCommandsBufferToRepeatIndex;

if( ( payload != NULL ) && ( payloadSize > 0 ) )

{

if( ( MacCommandsBufferIndex <= LORA\_MAC\_COMMAND\_MAX\_LENGTH ) && ( MacCommandsInNextTx == true ) )

{

fCtrl->Bits.FOptsLen += MacCommandsBufferIndex;

// Update FCtrl field with new value of OptionsLength

LoRaMacBuffer[0x05] = fCtrl->Value;

for( i = 0; i < MacCommandsBufferIndex; i++ )

{

LoRaMacBuffer[pktHeaderLen++] = MacCommandsBuffer[i];

}

}

}

else

{

if( ( MacCommandsBufferIndex > 0 ) && ( MacCommandsInNextTx ) )

{

payloadSize = MacCommandsBufferIndex;

payload = MacCommandsBuffer;

framePort = 0;

}

}

MacCommandsInNextTx = false;

// Store MAC commands which must be re-send in case the device does not receive a downlink anymore

MacCommandsBufferToRepeatIndex = ParseMacCommandsToRepeat( MacCommandsBuffer, MacCommandsBufferIndex, MacCommandsBufferToRepeat );

if( MacCommandsBufferToRepeatIndex > 0 )

{

MacCommandsInNextTx = true;

}

MacCommandsBufferIndex = 0;

if( ( payload != NULL ) && ( payloadSize > 0 ) )

{

LoRaMacBuffer[pktHeaderLen++] = framePort;

if( framePort == 0 )

{

LoRaMacPayloadEncrypt( (uint8\_t\* ) payload, payloadSize, LoRaMacNwkSKey, LoRaMacDevAddr, UP\_LINK, UpLinkCounter, LoRaMacPayload );

}

else

{

LoRaMacPayloadEncrypt( (uint8\_t\* ) payload, payloadSize, LoRaMacAppSKey, LoRaMacDevAddr, UP\_LINK, UpLinkCounter, LoRaMacPayload );

}

memcpy1( LoRaMacBuffer + pktHeaderLen, LoRaMacPayload, payloadSize );

}

LoRaMacBufferPktLen = pktHeaderLen + payloadSize;

LoRaMacComputeMic( LoRaMacBuffer, LoRaMacBufferPktLen, LoRaMacNwkSKey, LoRaMacDevAddr, UP\_LINK, UpLinkCounter, &mic );

LoRaMacBuffer[LoRaMacBufferPktLen + 0] = mic & 0xFF;

LoRaMacBuffer[LoRaMacBufferPktLen + 1] = ( mic >> 8 ) & 0xFF;

LoRaMacBuffer[LoRaMacBufferPktLen + 2] = ( mic >> 16 ) & 0xFF;

LoRaMacBuffer[LoRaMacBufferPktLen + 3] = ( mic >> 24 ) & 0xFF;

LoRaMacBufferPktLen += LORAMAC\_MFR\_LEN;

break;

case FRAME\_TYPE\_PROPRIETARY:

...// 省略

break;

default:

return LORAMAC\_STATUS\_SERVICE\_UNKNOWN;

}

return LORAMAC\_STATUS\_OK;

}

Join-request的组帧处理对应协议第6章 6.2.4 Join-request message。   
数据帧的组帧处理则稍微复杂些，尤其是FHDR，下面逐个字段讲解下FHDR。

#### 4.5.2.1 MACPayload中的FHDR

**1.FHDR中的DevAddr**

LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr ) & 0xFF;   
 LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 8 ) & 0xFF;   
 LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 16 ) & 0xFF;   
 LoRaMacBuffer[pktHeaderLen++] = ( LoRaMacDevAddr >> 24 ) & 0xFF;

**2.FHDR中的FCtrl**

首先 ADR 位段 是在传入 PrepareFrame() 之前，就做了处理。   
fCtrl.Bits.Adr = AdrCtrlOn;

接着 AdrAckReq 位段，在长期失联情况下会发送AdrAckReq确认链路。   
fCtrl->Bits.AdrAckReq=AdrNextDr(fCtrl->Bits.Adr,true, &LoRaMacParams.ChannelsDatarate );

最后 F0ptsLen 位段，会在下面计算完FOpts之后更新。

**3.FHDR中的FCnt**

LoRaMacBuffer[pktHeaderLen++] = UpLinkCounter & 0xFF;   
 LoRaMacBuffer[pktHeaderLen++] = ( UpLinkCounter >> 8 ) & 0xFF;

这个UpLinkCounter会在物理层发送完成后会按照协议进行累加。可以看到这是个32位计数器，按照协议规定，“如果采用32位帧计数，FCnt就对应计数器32位的16个低有效位”。

这是上行的，另外下行的也类似。

**4.FHDR中的FOpts**

把MAC命令放入F0pts中，并且更新F0ptsLen。MAC命令，要么使用非零的FPort来和数据一起传输，要么使用FPort0来单独传输。

// Copy the MAC commands which must be re-send into the MAC command buffer

memcpy1(&MacCommandsBuffer[MacCommandsBufferIndex], MacCommandsBufferToRepeat, MacCommandsBufferToRepeatIndex );

MacCommandsBufferIndex += MacCommandsBufferToRepeatIndex;

if( ( payload != NULL ) && ( payloadSize > 0 ) )

{

if((MacCommandsBufferIndex<=LORA\_MAC\_COMMAND\_MAX\_LENGTH ) && ( MacCommandsInNextTx == true ) )

{

fCtrl->Bits.FOptsLen += MacCommandsBufferIndex;

// Update FCtrl field with new value of OptionsLength

LoRaMacBuffer[0x05] = fCtrl->Value;

for( i = 0; i < MacCommandsBufferIndex; i++ )

{

LoRaMacBuffer[pktHeaderLen++] = MacCommandsBuffer[i];

}

}

}

else

{

if( ( MacCommandsBufferIndex > 0 ) && ( MacCommandsInNextTx ) )

{

payloadSize = MacCommandsBufferIndex;

payload = MacCommandsBuffer;

framePort = 0;

}

}

#### 4.5.2.2 MACPayload中的FPort

这个是在应用层一直传递进去的，协议栈默认是用了端口2。这个是后期大家在应用时要调整的，类似于IP端口，不同的端口对应不同的服务。

### 4.5.3 MIC解析

在函数 PrepareFrame()的最后是调用LoRaMacComputeMic() 计算出整个MAC层的校验码。应用层这边基本不用改这边就暂时不细究了。

# **第5章 MAC命令**

对网络管理者而言，有一套专门的MAC命令用来在服务器和终端MAC层之间交互。这套MAC命令对应用程序(不管是服务器端还是终端设备的应用程序)是不可见的。

单个数据帧中可以携带MAC命令，要么在FOpts字段中捎带，要么在独立帧中将FPort设成0后放在FRMPayload里。如果采用FOpts捎带的方式，MAC命令是不加密并且不长度超过15字节。如果采用独立帧放在FRMPayload的方式，那就必须采用加密方式，并且不超过FRMPayload的最大长度。

*注意：如果MAC命令不想被窃听，那就必须以独立帧形式放在FRMPayload中。*

每个MAC命令是由 1字节CID 跟着一段可能为空的字节序列组成的。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CID** | **Command** | **由谁发送** | | **描述** |
| 终端 | 网关 |
| 0x02 | LinkCheckReq | x |  | 终端利用这个命令来判断网络连接质量 |
| 0x02 | LinkCheckAns |  | x | LinkCheckReq的回复。包含接收信号强度，告知终端接收质量 |
| 0x03 | LinkADRReq |  | x | 向终端请求改变数据速率，发射功率，重传率以及信道 |
| 0x03 | LinkADRAns | x |  | LinkADRReq的回复。 |
| 0x04 | DutyCycleReq |  | x | 向终端设置发送的最大占空比。 |
| 0x04 | DutyCycleAns | x |  | DutyCycleReq的回复。 |
| 0x05 | RXParamSetupReq |  | x | 向终端设置接收时隙参数。 |
| 0x05 | RXParamSetupAns | x |  | RXParamSetupReq的回复。 |
| 0x06 | DevStatusReq |  | x | 向终端查询其状态。 |
| 0x06 | DevStatusAns | x |  | 返回终端设备的状态，即电池余量和链路解调预算。 |
| 0x07 | NewChannelReq |  | x | 创建或修改 1个射频信道 定义。 |
| 0x07 | NewChannelAns | x |  | NewChannelReq的回复。 |
| 0x08 | RXTimingSetupReq |  | x | 设置接收时隙的时间。 |
| 0x08 | RXTimingSetupAns | x |  | RXTimingSetupReq的回复。 |
| 0x80~0xFF | 私有 | x | x | 给私有网络命令拓展做预留。 |

表4：MAC命令表

*注意：MAC命令的长度虽然没有明确给出，但是MAC执行层必须要知道。因此未知的MAC命令无法被忽略，且前面未知的MAC命令会终止MAC命令的处理队列。所以建议按照LoRaWAN协议介绍的MAC命令来处理MAC命令。这样所有基于LoRaWAN协议的MAC命令都可以被处理，即使是更高版本的命令。*

## 5.1梳理解析

从LoRaWAN第4章的帧格式可以得到如下信息：MAC命令，要么使用FPort0来单独传输，要么使用非零的FPort来和数据一起传输。

LoRaWAN第5章，LoRaWAN出于网络管理需要，提出了9条MAC命令，这个章节是对9条命令进行具体的描述。

说个题外话，CLAA(中国LoRa应用联盟)在9条命令以外还扩充了一些MAC命令。现阶段协议还不能公开，所以我就不多说了。中兴目前作为LoRa联盟董事会成员，也许以后会把这些拓展MAC命令引入到LoRaWAN协议也说不准，大家暂且当个课外知识了解下就好。

## 5.2代码位置

### 5.2.1 MAC命令枚举

/\*!

\* LoRaMAC mote MAC commands

\*

\* LoRaWAN Specification V1.0.1, chapter 5, table 4

\*/

typedef enum eLoRaMacMoteCmd

{

/\*!

\* LinkCheckReq

\*/

MOTE\_MAC\_LINK\_CHECK\_REQ = 0x02,

/\*!

\* LinkADRAns

\*/

MOTE\_MAC\_LINK\_ADR\_ANS = 0x03,

/\*!

\* DutyCycleAns

\*/

MOTE\_MAC\_DUTY\_CYCLE\_ANS = 0x04,

/\*!

\* RXParamSetupAns

\*/

MOTE\_MAC\_RX\_PARAM\_SETUP\_ANS = 0x05,

/\*!

\* DevStatusAns

\*/

MOTE\_MAC\_DEV\_STATUS\_ANS = 0x06,

/\*!

\* NewChannelAns

\*/

MOTE\_MAC\_NEW\_CHANNEL\_ANS = 0x07,

/\*!

\* RXTimingSetupAns

\*/

MOTE\_MAC\_RX\_TIMING\_SETUP\_ANS = 0x08,

}LoRaMacMoteCmd\_t;

////////////////////////////////////////////////////////////////////////////////////////////////

/\*!

\* LoRaMAC server MAC commands

\*

\* LoRaWAN Specification V1.0.1 chapter 5, table 4

\*/

typedef enum eLoRaMacSrvCmd

{

/\*!

\* LinkCheckAns

\*/

SRV\_MAC\_LINK\_CHECK\_ANS = 0x02,

/\*!

\* LinkADRReq

\*/

SRV\_MAC\_LINK\_ADR\_REQ = 0x03,

/\*!

\* DutyCycleReq

\*/

SRV\_MAC\_DUTY\_CYCLE\_REQ = 0x04,

/\*!

\* RXParamSetupReq

\*/

SRV\_MAC\_RX\_PARAM\_SETUP\_REQ = 0x05,

/\*!

\* DevStatusReq

\*/

SRV\_MAC\_DEV\_STATUS\_REQ = 0x06,

/\*!

\* NewChannelReq

\*/

SRV\_MAC\_NEW\_CHANNEL\_REQ = 0x07,

/\*!

\* RXTimingSetupReq

\*/

SRV\_MAC\_RX\_TIMING\_SETUP\_REQ = 0x08,

}LoRaMacSrvCmd\_t;

### 5.2.2 MAC命令的接收处理

OnRadioRxDone()携带着MAC帧进来，经过层层筛选，最终到达ProcessMacCommands()来处理MAC命令。   
 这里代码中涉及的两种处理方式，可以跟协议对应起来：port = 0时，MAC命令放在FRMPayload中，需要先解密再处理；port非零时，MAC命令放在fopts中。

if( port == 0 )

{

if( fCtrl.Bits.FOptsLen == 0 )

{

LoRaMacPayloadDecrypt( payload + appPayloadStartIndex,

frameLen,

nwkSKey,

address,

DOWN\_LINK,

downLinkCounter,

LoRaMacRxPayload );

// Decode frame payload MAC commands

ProcessMacCommands( LoRaMacRxPayload, 0, frameLen, snr );

}

}

else

{

if( fCtrl.Bits.FOptsLen > 0 )

{

// Decode Options field MAC commands. Omit the fPort.

ProcessMacCommands( payload, 8, appPayloadStartIndex - 1, snr );

}

}

### 5.2.3 MAC命令的发送及回复

MAC命令的发送及回复处理都在这个函数中，AddMacCommand()。

协议栈对MAC命令发送的处理还是比较简单的，都是放在Fopts中来传输，都在这个15字节的MacCommandsBuffer中。

LinkADR是LoRaWAN网络管理中相当重要的一个MAC命令，其解析占用了183行。索性专门写篇源码解析，记录下。

阅读此文前，最好再把第五章的这个命令好好翻一翻，代码和协议才能对应上。

我正在陆续对协议的各个章节进行翻译，具体其他章节的译文，以及译文之外的代码解析，可点此查看帖子[LoRa学习笔记\_汇总](http://blog.csdn.net/iotisan/article/details/53930458" \t "_blank)。

## 5.3 LinkADRReq 的源码解析

按照代码思路走一遍。

### 5.3.1 解析 DataRate\_TXPower 字段

datarate = payload[macIndex++];

txPower = datarate & 0x0F;

datarate = ( datarate >> 4 ) & 0x0F;

if( ( AdrCtrlOn == false ) &&

( (LoRaMacParams.ChannelsDatarate != datarate ) ||

( LoRaMacParams.ChannelsTxPower != txPower ) )

)

{

// ADR disabled don't handle ADR requests if server tries to change datarate or txpower

// Answer the server with fail status

// Power ACK = 0

// Data rate ACK = 0

// Channel mask = 0

AddMacCommand( MOTE\_MAC\_LINK\_ADR\_ANS, 0, 0 );

macIndex += 3; // Skip over the remaining bytes of the request

break;

}

如果终端ADR没开，那么就立即丢弃本命令处理。这里的macIndex += 3是对应LinkADRReq的剩余命令长度3而言的。

### 5.3.2 解析 ChMask 字段

chMask = ( uint16\_t )payload[macIndex++];

chMask |= ( uint16\_t )payload[macIndex++] << 8;

### 5.3.3 解析 Redundancy 字段

nbRep = payload[macIndex++];

chMaskCntl = ( nbRep >> 4 ) & 0x07;

nbRep &= 0x0F;

if( nbRep == 0 )

{

nbRep = 1;

}

把字段中的 chMaskCntl 和 nbRep 都给解析了出来。

### 5.3.4 按地区规定处理 chMaskCntl ，及判断 ChMask 有效性

#elif defined( USE\_BAND\_470 )

if( chMaskCntl == 6 )

{

// Enable all 125 kHz channels

for( uint8\_t i = 0, k = 0; i < LORA\_MAX\_NB\_CHANNELS; i += 16, k++ )

{

for( uint8\_t j = 0; j < 16; j++ )

{

if( Channels[i + j].Frequency != 0 )

{

channelsMask[k] |= 1 << j;

}

}

}

}

else if( chMaskCntl == 7 )

{

status &= 0xFE; // Channel mask KO

}

else

{

for( uint8\_t i = 0; i < 16; i++ )

{

if( ( ( chMask & ( 1 << i ) ) != 0 ) &&

( Channels[chMaskCntl \* 16 + i].Frequency == 0 )

)

{

// Trying to enable an undefined channel

status &= 0xFE; // Channel mask KO

}

}

channelsMask[chMaskCntl] = chMask;

}

如果 chMaskCntl 为6，则所有信道都使能。如果 chMaskCntl 为7，则由于未定义返回失败。   
 其他有效 chMaskCntl 情况下，先检查是否有未定义的频点，如果没问题则更新对应的channelsMask。

### 5.3.5 判断速率有效性

if( ValidateDatarate( datarate, channelsMask ) == false )

{

// Datarate KO

status &= 0xFD;

}

### 5.3.6 判断发射功率有效性

if( ValueInRange(txPower,LORAMAC\_MAX\_TX\_POWER, LORAMAC\_MIN\_TX\_POWER ) == false )

{

status &= 0xFB; // TxPower KO

}

### 5.3.7 全部判断通过后更新参数

if( ( status & 0x07 ) == 0x07 )

{

LoRaMacParams.ChannelsDatarate = datarate;

LoRaMacParams.ChannelsTxPower = txPower;

memcpy1( (uint8\_t\* )LoRaMacParams.ChannelsMask,

( uint8\_t\* )channelsMask,

sizeof( LoRaMacParams.ChannelsMask ) );

LoRaMacParams.ChannelsNbRep = nbRep;

}

### 5.3.8 回复MAC命令 LinkADRAns

AddMacCommand( MOTE\_MAC\_LINK\_ADR\_ANS, status, 0 );

突然发现 AddMacCommand 的形参只有CID加2字节的回复，我是太无聊，把终端所有MAC命令都翻了一遍，确认所有payload确实是小于2字节。再次赞扬LoRaWAN协议的精简作风。

# 第6章 终端激活

为了加入LoRaWAN网络，每个终端需要初始化及激活。

终端的激活有两种方式，一种是空中激活 Over-The-Air Activation (OTAA)，当设备部署和重置时使用; 另一种是独立激活 Activation By Personalization (ABP)，此时初始化和激活这两步就在一个步骤内完成。

twowinter 备注： ABP 这个词不太好翻译，通常会翻成个性化激活，也就是通过独立配置参数的方式激活。但总感觉少点味道，与空中激活摆在一起，感觉独立激活这个词在语义上更有并列感。当然这是我的主观感觉，建议大家和同行交流时，还是说 ABP激活 吧。

## 6.1 终端激活后的数据存储

激活后，终端会存储如下信息：设备地址(DevAddr)，应用ID(AppEUI)，网络会话密钥(NwkSKey)，应用会话密钥(AppSKey)。

### 6.1.1 终端地址(DevAddr)

终端地址(DevAddr)由可标识当前网络设备的32位ID所组成，具体格式如下：

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

它的高7位是NwkId，用来区别同一区域内的不同网络，另外也保证防止节点窜到别的网络去。它的低25位是NwkAddr，是终端的网络地址，可以由网络管理者来分配。

### 6.1.2 应用ID(AppEUI)

AppEUI是一个类似IEEE EUI64的全球唯一ID，标识终端的应用提供者。   
 APPEUI在激活流程开始前就存储在终端中。

### 6.1.3 网络会话密钥(NwkSKey)

NwkSKey被终端和网络服务器用来计算和校验所有消息的MIC，以保证数据完整性。也用来对单独MAC的数据消息载荷进行加解密。

### 6.1.4 应用会话密钥(AppSKey)

AppSKey被终端和网络服务器用来对应用层消息进行加解密。当应用层消息载荷有MIC时，也可以用来计算和校验该应用层MIC。

## 6.2 空中激活 OTAA

针对空中激活，终端必须按照加网流程来和网络服务器进行数据交互。如果终端丢失会话消息，则每次必须重新进行一次加网流程。   
 加网流程需要终端准备好如下这三个参数：DevEUI，AppEUI，AppKey。

APPEUI在上面的6.1.2已经做了描述。

*注意：对于空中激活，终端不会初始化任何网络密钥。只有当终端加入网络后，才会被分配一个网络会话密钥，用来加密和校验网络层的传输。通过这样，使得终端在不同网络间的漫游处理变得方便。同时使用网络和应用会话密钥，使得网络服务器中的应用数据，不会被网络提供者读取或者篡改。*

**6.2.1 终端 ID (DevEUI)** DevEUI 是一个类似IEEE EUI64的全球唯一ID，标识唯一的终端设备。

**6.2.2 应用密钥(AppKey)** AppKey 是由应用程序拥有者分配给终端，很可能是由应用程序指定的根密钥来衍生的，并且受提供者控制。当终端通过空中激活方式加入网络，AppKey用来产生会话密钥NwkSKey和AppSKey，会话密钥分别用来加密和校验网络层和应用层数据。

**6.2.3 加网流程** 从终端角度看，加网流程是由和服务器的两个MAC命令交互组成的，分别是 join request 和 join accept。

**6.2.4 Join-request 消息** 加网流程总是由终端发送 join-request 来发起。

|  |  |  |  |
| --- | --- | --- | --- |
| **Size (bytes)** | 8 | 8 | 2 |
| **Join Request** | AppEUI | DevEUI | DevNonce |

·join-request 消息包含了AppEUI 和 DevEUI ，后面还跟了2个字节的声明 DevNonce。

·DevNonce 是一个随机值。网络服务器为每个终端记录过去的 DevNonce 数值，如果相同设备发出相同的 DevNonce 的join request就会忽略。

join-request 消息的MIC数值(见第4章 MAC帧格式)按照如下公式计算：

cmac = aes128\_cmac(AppKey, MHDR | AppEUI | DevEUI | DevNonce)   
 MIC = cmac[0..3]

join-request 消息不用加密。

**6.2.5 Join-accept 消息**  
 待补充。

## 6.3 独立激活 ABP

在某些情况下，终端可以独立激活。独立激活是让终端绕过 join request - join accept的加网流程，直接加入到指定网络中。

独立激活终端，意味着 DevAddr 和两个会话密钥 NwkSKey 和 AppSKey 直接存储在终端中，而不是DevEUI，AppEUI，AppKey。终端在一开始就配置好了入网必要的信息。

每个终端必须要有唯一的 NwkSKey 和 AppSKey。这样，一个设备的密钥被破解也不会造成其他设备的安全性危险。创建那些密钥的过程中，密钥不允许通过公开可用信息获得(例如节点地址)。

## 6.4 梳理解析

LoRaWAN第6章，主要对节点加网做了描述，它有两种方式。如果要用一句话来总结的话，那就是这一句了，请看：

如果是空中激活，则需要准备 DevEUI，AppEUI，AppKey 这三个参数，即设备自身MAC地址和要使用的应用(应用ID和密钥)。   
 如果是ABP激活，则直接配置 DevAddr，NwkSKey，AppSKey 这三个LoRaWAN最终通讯的参数，不再需要join流程。在这种情况下，这个设备是可以直接发应用数据的。

这里插个题外话，商用的LoRaWAN网络一般都是走OTAA流程，这样安全性才得以保证。  
*注：如果是空中激活，则需要准备 DevEUI，AppEUI，AppKey来join。如果是ABP激活，则直接配置 DevAddr，NwkSKey，AppSKey。*

## 6.5 代码位置

### 6.5.1 激活处理

协议的第6章，相关的核心代码是这么几行，位于 \src\mac\main.c。   
整个代码结构非常清晰，用一个宏(OVER\_THE\_AIR\_ACTIVATION)分开两段，分别对应两种激活方式。

case DEVICE\_STATE\_JOIN:

{

#if( OVER\_THE\_AIR\_ACTIVATION != 0 )

MlmeReq\_t mlmeReq;

// Initialize LoRaMac device unique ID

BoardGetUniqueId( DevEui );

mlmeReq.Type = MLME\_JOIN;

mlmeReq.Req.Join.DevEui = DevEui;

mlmeReq.Req.Join.AppEui = AppEui;

mlmeReq.Req.Join.AppKey = AppKey;

if( NextTx == true )

{

LoRaMacMlmeRequest( &mlmeReq );

}

DeviceState = DEVICE\_STATE\_SLEEP;

#else

// Choose a random device address if not already defined in Comissioning.h

if( DevAddr == 0 )

{

// Random seed initialization

srand1( BoardGetRandomSeed( ) );

// Choose a random device address

DevAddr = randr( 0, 0x01FFFFFF );

}

mibReq.Type = MIB\_NET\_ID;

mibReq.Param.NetID = LORAWAN\_NETWORK\_ID;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_DEV\_ADDR;

mibReq.Param.DevAddr = DevAddr;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_NWK\_SKEY;

mibReq.Param.NwkSKey = NwkSKey;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_APP\_SKEY;

mibReq.Param.AppSKey = AppSKey;

LoRaMacMibSetRequestConfirm( &mibReq );

mibReq.Type = MIB\_NETWORK\_JOINED;

mibReq.Param.IsNetworkJoined = true;

LoRaMacMibSetRequestConfirm( &mibReq );

DeviceState = DEVICE\_STATE\_SEND;

#endif

break;

}

### 6.5.2 参数配置

关于参数部分，相关的默认值全部位于\src\apps\LoRaMac\classA\硬件平台\Comissioning.h

/\*!

\* Mote device IEEE EUI (big endian)

\*

\* \remark In this application the value is automatically generated by calling

\* BoardGetUniqueId function

\*/

#define LORAWAN\_DEVICE\_EUI { IEEE\_OUI, 0x00, 0x00, 0x00, 0x00, 0x00 }

/\*!

\* Application IEEE EUI (big endian)

\*/

#define LORAWAN\_APPLICATION\_EUI { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

</code></pre>

<pre><code>

/\*!

\* AES encryption/decryption cipher application key

\*/

#define LORAWAN\_APPLICATION\_KEY { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }

/\*!

\* Current network ID

\*/

#define LORAWAN\_NETWORK\_ID ( uint32\_t )0

/\*!

\* Device address on the network (big endian)

\*

\* \remark In this application the value is automatically generated using

\* a pseudo random generator seeded with a value derived from

\* BoardUniqueId value if LORAWAN\_DEVICE\_ADDRESS is set to 0

\*/

#define LORAWAN\_DEVICE\_ADDRESS ( uint32\_t )0x00000000

/\*!

\* AES encryption/decryption cipher network session key

\*/

#define LORAWAN\_NWKSKEY { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }

/\*!

\* AES encryption/decryption cipher application session key

\*/

#define LORAWAN\_APPSKEY { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0x...}

# 第7章 LoRaWAN地区参数

## 7.1 介绍

这份文档描述了全球不同地区的LoRaWAN具体参数。这份文档是对LoRaWAN协议文档(从版本V1.0.2开始)的配套补充文档。为了避免新区域的加入而导致文档的变动，因此将地区参数章节从协议规范中剥离出来。

## 7.2 LoRaWAN地区参数

### 7.2.1 欧洲 863-870MHz 免授权频段

待补充，计划3月份补足。

### 7.2.2 美国 902-928MHz 免授权频段

待补充，计划3月份补足。

### 7.2.3 中国 779-787MHz 免授权频段

待补充，计划3月份补足。

### 7.2.4 欧洲 433MHz 免授权频段

待补充

### 7.2.5 澳洲 915-928MHz 免授权频段

待补充

### 7.2.6 中国 470-510MHz 频段

#### 7.2.6.1 中国 470-510MHz 前导码格式

要用如下的同步字：

|  |  |  |
| --- | --- | --- |
| **调制方式** | **同步字** | **前导码长度** |
| LoRa | 0x34 | 8 symbols |

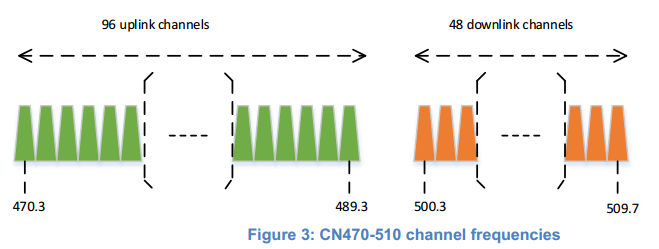
#### 7.2.6.2 中国 470-510MHz 信道频率

在中国，无线电管理局SRRC规定了这个频段用于民用表计应用。

470频段需要按照如下信道规划进行部署：

**上行** – 从0到95共96个信道，带宽为125KHz，速率从DR0到DR5，使用编码率4/5，从470.3MHz按200KHz递增到489.3KHz。

6到38和45到77的这几十个信道，主要用于中国电力。在中国电力使用了这些信道的区域，则LoRaWAN不能使用这些信道。

**下行** – 从0到47共48个信道，带宽为125KHz，速率从DR0到DR5，使用编码率4/5，从500.3MHz按200KHz递增到509.7KHz。   


LoRaWAN在满足如下条件下可以使用中国 470-510MHz 频段：

射频发射功率(EIRP(Effective Isotropic Radiated Power) 有效全向辐射功率)要小于50mW(或者17dBm)。

发射持续时间不能超过5000ms。

中国 470-510MHz 频段的终端需要使用如下默认设置：

·默认的射频发射功率为：14dBm。

·中国470-510MHz的终端设备应当可以在这个频段内进行操作，必须有足够的信道[数据结构](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)体来存储96个上行信道。

如果使用OTAA空中激活方式，终端应该广播JoinReq消息，使用96个信道中的随机信道，速率从DR5-DR0，带宽为125KHz。   
 ABP激活方式的设备应当在复位后，96个信道均可用。

#### 7.2.6.3 中国 470-510MHz 速率和发射功率

中国470-510MHz的物理层没有限制停留时间。终端设备不一定需要执行TxParamSetupReq的MAC命令。

如下编码用于中国 470-510MHz 频段的数据速率和发射功率。

|  |  |  |
| --- | --- | --- |
| **数据速率** | **配置** | **物理层比特率** |
| 0 | LoRa: SF12 / 125 kHz | 250 |
| 1 | LoRa: SF11 / 125 kHz | 440 |
| 2 | LoRa: SF10 / 125 kHz | 980 |
| 3 | LoRa: SF9 / 125 kHz | 1760 |
| 4 | LoRa: SF8 / 125 kHz | 3125 |
| 5 | LoRa: SF7 / 125 kHz | 5470 |
| 6:15 | RFU |  |

|  |  |
| --- | --- |
| **发射功率** | **配置** |
| 0 | 17 dBm |
| 1 | 16 dBm |
| 2 | 14 dBm |
| 3 | 12 dBm |
| 4 | 10 dBm |
| 5 | 7 dBm |
| 6 | 5 dBm |
| 7 | 2 dBm |
| 8…15 | RFU |

表34: 中国470 数据速率和发射功率表

#### 7.2.6.4 中国 470-510MHz JoinResp命令的CFList字段

中国 470-510MHz不支持LoRaWAN的JoinAccept命令中携带可选CFlist。 如果CFlist字段非空，那终端得忽略它。

#### 7.2.6.5 中国 470-510MHz LinkAdrReq命令

对于中国 470-510MHz 的版本，LinkADRReq命令中的ChMaskCntl有如下定义：

|  |  |
| --- | --- |
| **ChMaskCntl** | **ChMask用于** |
| 0 | 信道 0 到 15 |
| 1 | 信道 16 到 31 |
| 2 | 信道 32 到 47 |
| 3 | 信道 48 到 63 |
| 4 | 信道 64 到 79 |
| 5 | 信道 80 到 95 |
| 6 | 所有信道打开，不管ChMask字段，所有定义的信道都可用 |
| 7 | RFU |

表35: 中国470 ChMaskCntl数值表

如果ChMask的数值为RFU，终端则拒绝该命令，并且在回复中不设置Channel mask ACK。

#### 7.2.6.6 中国 470-510MHz 最大载荷长度

MAC层载荷的最大长度在下表中进行了规定。这是考虑了重传包，从物理层最大允许传输时间来推算出来。应用层载荷的最大长度(MAC可选控制字段FOpt为空情况下)也给了出来。如果FOpts字段非空的话，那么N的字还要更小。

|  |  |  |
| --- | --- | --- |
| **数据速率** | **M** | **N** |
| 0 | 59 | 51 |
| 1 | 59 | 51 |
| 2 | 59 | 51 |
| 3 | 123 | 115 |
| 4 | 230 | 222 |
| 5 | 230 | 222 |
| 6:15 | 未定义 | |

表36: 中国470 最大载荷长度

#### 7.2.6.7 中国 470-510MHz 接收窗口

· RX1接收信道被上行信道用来发起数据交互。RX1接收信道按照如下方式定义。   
 RX1信道数 = 上行信道数对48取余，例如上行信道为49时，则rx1信道数为1。

· RX1窗口的数据速率取决于发送数据速率(见下面的表37:中国 470-510MHz 数据速率偏移)。

· RX2(第二接收窗口)的设置使用固定数据速率和频率。默认参数为505.3 MHz / DR0。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RX1数据速率偏移** | **0** | **1** | **2** | **3** | **4** | **5** |
| **上行数据速率** | **RX1时隙的下行数据速率** | | | | | |
| DR0 | DR0 | DR0 | DR0 | DR0 | DR0 | DR0 |
| DR1 | DR1 | DR0 | DR0 | DR0 | DR0 | DR0 |
| DR2 | DR2 | DR1 | DR0 | DR0 | DR0 | DR0 |
| DR3 | DR3 | DR2 | DR1 | DR0 | DR0 | DR0 |
| DR4 | DR4 | DR3 | DR2 | DR1 | DR0 | DR0 |
| DR5 | DR5 | DR4 | DR3 | DR2 | DR1 | DR0 |

表37:中国 470-510MHz 数据速率偏移

RX1数据速率偏移的允许值只能在[0:3]的范围内，[4:7]保留给后面使用。

#### 7.2.6.8 中国 470-510MHz 默认设置

中国 470-510MHz 频段对以下参数提供了一些推荐值。

RECEIVE\_DELAY1 1 s

RECEIVE\_DELAY2 2 s (必须是RECEIVE\_DELAY1 + 1s)

JOIN\_ACCEPT\_DELAY1 5 s

JOIN\_ACCEPT\_DELAY2 6 s

MAX\_FCNT\_GAP 16384

ADR\_ACK\_LIMIT 64

ADR\_ACK\_DELAY 32

ACK\_TIMEOUT 2 +/- 1 s (随机延时1到3秒)

如果终端实际使用的参数和上面默认值不同(例如终端使用更长的延时RECEIVE\_DELAY1和2)，那些参数必须在终端授权时使用带外信道和网络服务器 通信告知。网络服务器正在不会接受非默认值的参数。

### 7.2.7 亚洲 923MHz 免授权频段

待补充

### 7.2.8 韩国 920-923MHz 免授权频段

待补充

## 3 版本

版本V1.0

## 2 说正事 - LoRa联盟发布了地区参数文件的版本B

其实没什么大事，这周早些时候，LoRa联盟发布了地区参数文件的版本B。

地区参数文件，描述了全球不同地区的LoRaWAN具体参数。这份文档是对LoRaWAN协议文档(从版本V1.0.2开始)的配套补充文档。为了避免新区域的加入而导致文档的变动，因此将地区参数章节从协议规范中剥离出来。

The LoRa Alliance just released the latest update to its [regional parameters for LoRaWAN 1.0.2 revB]. Some of the many improvements include:

- India: added support for the country’s 865 MHz band, which enabled the recent deployment

announced by Tata Communications that includes managing data from 200,000 sensors and

gateways using Hewlett Packard Enterprise’s Universal IoT platform.

- Australia: the LoRa Alliance demonstrated its responsiveness to the constantly changing

technical landscape by rapidly adding support for Australia’s recent regulatory change to its

ISM band. These changes greatly extend the maximum range and payloads of LoRaWAN

networks, using even the lowest data rates.

- Korea: expanded Korean band capabilities.

- Global: added more consistent support for regions where the maximum allowed transmit

power is expressed as equivalent isotropic radiated power (EIRP). This EIRP support makes it

much easier for network operators to manage heterogeneous devices while ensuring

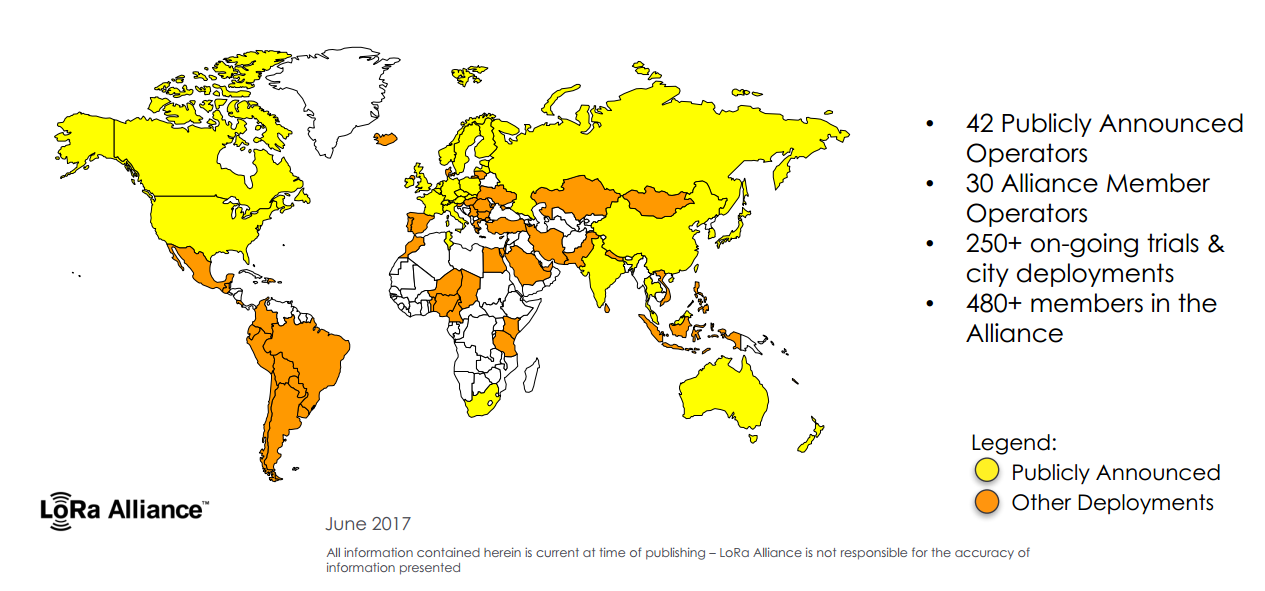
regulatory compliance.

简单梳理下：

印度新增865MHz频段，这为塔塔通讯近期宣布的 20 万传感器和基站节点建设计划进行了规范铺路。更新提及塔塔通讯将使用 HPE 的 IoT平台，HPE是惠普2015年分出来的技术公司，主攻云平台这块，这个知识点大家伙做个简单备忘。

另外一个大的更新应该是澳大利亚地区，澳大利亚政府最近更新了ISM频段的监管要求，这使得LoRaWAN可以极大地提高了载荷容量，特别是低速率情况下的数据量，你知道吗?以前澳大利亚在最低速率只能发11字节。

在公告中，官方还发布了最新的网络覆盖图，可以明显看到大洋洲这块转为了公开部署网络的国家。



# 前言

在[LoRaWAN协议中文版\_配套文件 地区参数(物理层)](http://blog.csdn.net/iotisan/article/details/55056092" \t "_blank)中已经为中国规划了470频段，因此国内开发者对此需求很强烈。

在最新(2017-02-27)的V4.3.1版本协议栈上已经新增了中国470频段。这篇文章从源码角度解析下其实现方式。

目前国内的LoRaWAN基站产品都和标准有一些不同，比如CLAA等，所以搞清楚整个代码实现还是很有必要的。只要熟悉了整个流程，对接任何一个基站都不是难事。

我正在陆续对协议的各个章节进行翻译，具体其他章节的译文，以及译文之外的代码解析，可点此查看帖子[LoRa学习笔记\_汇总](http://blog.csdn.net/iotisan/article/details/53930458" \t "_blank)。

# 源码解析

## 1.前导码格式的源码实现

同步字的处理在SX1276的驱动中：

void SX1276SetPublicNetwork( bool enable )

{

SX1276SetModem( MODEM\_LORA );

if( enable == true )

{

// Change LoRa modem SyncWord

SX1276Write( REG\_LR\_SYNCWORD, LORA\_MAC\_PUBLIC\_SYNCWORD );

}

else

{

// Change LoRa modem SyncWord

SX1276Write( REG\_LR\_SYNCWORD, LORA\_MAC\_PRIVATE\_SYNCWORD );

}

}

而前导码长度则是在每次SetTxConfig和SetRxConfig时配置进去。

## 2.信道频率的源码实现

先说上行信道的处理。

第一步，初始化时把所有信道6\*16=96个上行信道都使能了。

LoRaMacParamsDefaults.ChannelsMask[0] = 0xFFFF;

LoRaMacParamsDefaults.ChannelsMask[1] = 0xFFFF;

LoRaMacParamsDefaults.ChannelsMask[2] = 0xFFFF;

LoRaMacParamsDefaults.ChannelsMask[3] = 0xFFFF;

LoRaMacParamsDefaults.ChannelsMask[4] = 0xFFFF;

LoRaMacParamsDefaults.ChannelsMask[5] = 0xFFFF;

第二步，紧接着把96个信道的频点赋值一遍。

for( uint8\_t i = 0; i < LORA\_MAX\_NB\_CHANNELS; i++ )

{

Channels[i].Frequency = 470.3e6 + i \* 200e3;

Channels[i].DrRange.Value = ( DR\_5 << 4 ) | DR\_0;

Channels[i].Band = 0;

}

第三步，发送时在SetNextChannel中选择合适的频点，默认是96个信道中随机选择。

Channel = enabledChannels[randr( 0, nbEnabledChannels - 1 )];

这上面是上行信道处理三部曲，下行信道处理则轻松多了。主要是配合接收窗口处理，由这个宏定义了下行的起始频点。具体可以看下面第7点。

#define LORAMAC\_FIRST\_RX1\_CHANNEL ( (uint32\_t) 500.3e6 )

## 3.数据速率和节点发射功率编码

速率编码如下：

const uint8\_t Datarates[] = { 12, 11, 10, 9, 8, 7 };

发射功率编码如下：

const int8\_t TxPowers[] = { 17, 16, 14, 12, 10, 7, 5, 2 };

速率范围如下：

/\*!

\* Minimal datarate that can be used by the node

\*/

#define LORAMAC\_TX\_MIN\_DATARATE DR\_0

/\*!

\* Maximal datarate that can be used by the node

\*/

#define LORAMAC\_TX\_MAX\_DATARATE DR\_5

/\*!

\* Minimal datarate that can be used by the node

\*/

#define LORAMAC\_RX\_MIN\_DATARATE DR\_0

/\*!

\* Maximal datarate that can be used by the node

\*/

#define LORAMAC\_RX\_MAX\_DATARATE DR\_5

/\*!

\* Default datarate used by the node

\*/

#define LORAMAC\_DEFAULT\_DATARATE DR\_0

发射功率范围如下：

/\*!

\* Minimal Tx output power that can be used by the node

\*/

#define LORAMAC\_MIN\_TX\_POWER TX\_POWER\_2\_DBM

/\*!

\* Maximal Tx output power that can be used by the node

\*/

#define LORAMAC\_MAX\_TX\_POWER TX\_POWER\_17\_DBM

/\*!

\* Default Tx output power used by the node

\*/

#define LORAMAC\_DEFAULT\_TX\_POWER TX\_POWER\_14\_DBM

## 4.CFList

中国没有。具体见OnRadioRxDone中的FRAME\_TYPE\_JOIN\_ACCEPT分支。

## 5.LinkAdrReq命令

对于 ChMaskCntl 的处理都在 ProcessMacCommands() 的 SRV\_MAC\_LINK\_ADR\_REQ 分支中。

小彩蛋一个：你发现没，注释里写着Channel mask KO。不知是djaeckle (loramac-node的作者之一)调皮，还是语言习惯如此。

if( chMaskCntl == 6 )

{

// Enable all 125 kHz channels

for( uint8\_t i = 0, k = 0; i < LORA\_MAX\_NB\_CHANNELS; i += 16, k++ )

{

for( uint8\_t j = 0; j < 16; j++ )

{

if( Channels[i + j].Frequency != 0 )

{

channelsMask[k] |= 1 << j;

}

}

}

}

else if( chMaskCntl == 7 )

{

status &= 0xFE; // Channel mask KO

}

else

{

for( uint8\_t i = 0; i < 16; i++ )

{

if( ( ( chMask & ( 1 << i ) ) != 0 ) &&

( Channels[chMaskCntl \* 16 + i].Frequency == 0 ) )

{// Trying to enable an undefined channel

status &= 0xFE; // Channel mask KO

}

}

channelsMask[chMaskCntl] = chMask;

}

## 6.最大载荷长度

/\*!

\* Maximum payload with respect to the datarate index. Cannot operate with repeater.

\*/

const uint8\_t MaxPayloadOfDatarate[] = { 51, 51, 51, 115, 222, 222 };

/\*!

\* Maximum payload with respect to the datarate index. Can operate with repeater.

\*/

const uint8\_t MaxPayloadOfDatarateRepeater[] = { 51, 51, 51, 115, 222, 222 };

这在RxWindowSetup()进行处理，调用了最终的驱动函数。

if( RepeaterSupport == true )

{

Radio.SetMaxPayloadLength( modem, MaxPayloadOfDatarateRepeater[datarate] + LORA\_MAC\_FRMPAYLOAD\_OVERHEAD );

}

else

{

Radio.SetMaxPayloadLength( modem, MaxPayloadOfDatarate[datarate] + LORA\_MAC\_FRMPAYLOAD\_OVERHEAD );

}

## 7.接收窗口处理。

RX1的处理在OnRxWindow1TimerEvent()中，满足协议要求。

RxWindowSetup( LORAMAC\_FIRST\_RX1\_CHANNEL + ( Channel % 48 ) \* LORAMAC\_STEPWIDTH\_RX1\_CHANNEL, datarate, bandwidth, symbTimeout, false );

RX2的默认参数见如下宏：

#define RX\_WND\_2\_CHANNEL { 505300000, DR\_0 }

RX2的处理在OnRxWindow2TimerEvent()中：

if( RxWindowSetup( LoRaMacParams.Rx2Channel.Frequency, LoRaMacParams.Rx2Channel.Datarate, bandwidth,

symbTimeout, rxContinuousMode ) == true )

{

RxSlot = 1;

}

速率偏移处理如下：

datarate = LoRaMacParams.ChannelsDatarate - LoRaMacParams.Rx1DrOffset;

if( datarate < 0 )

{

datarate = DR\_0;

}

## 8.默认设置

目前基本各地区的参数都一样，因此协议栈也是直接共用如下参数：

/\*!

\* Class A&B receive delay 1 in ms

\*/

#define RECEIVE\_DELAY1 1000

/\*!

\* Class A&B receive delay 2 in ms

\*/

#define RECEIVE\_DELAY2 2000

/\*!

\* Join accept receive delay 1 in ms

\*/

#define JOIN\_ACCEPT\_DELAY1 5000

/\*!

\* Join accept receive delay 2 in ms

\*/

#define JOIN\_ACCEPT\_DELAY2 6000

/\*!

\* Class A&B maximum receive window delay in ms

\*/

#define MAX\_RX\_WINDOW 3000

/\*!

\* Maximum allowed gap for the FCNT field

\*/

#define MAX\_FCNT\_GAP 16384

# 第8章 Class B 介绍

这章描述了LoRaWAN Class B层，这是为电池节点优化设计的，不管它是移动还是固定在某个位置。

Class B 的终端必须执行如下操作，为了获得服务端发起的下行消息，终端必须按要求开启一个固定间隔的接收窗口。

LoRaWAN Class B 可以选择在终端上增加一个异步接收窗口。

LoRaWAN Class A 的限制之一就是终端发送数据使用的Aloha[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)；这使得客户应用程序或者服务端不能在确定时间内定位终端。Class B 的目的就是在Class A 终端随机上行后的接收窗口之外，让终端也能在可预见的时间内开启接收。Class B 是让网关周期发送信标来同步网络中的所有终端，为此终端需要在周期时隙的确定时间点打开一个短的接收窗口(叫做“ping slot”)。

*注意：是否要从Class A 切换到 Class B，这个要在终端的应用层进行处理。如果打算从网络端将Class A 切换到 Class B，客户程序只能利用终端 Class A的上行包来反馈一个下行包给节点，需要应用层上处理来识别这个请求 - 这个处理不在LoRaWAN层面。*

# 第17章 持续接收的终端(Class C)

具备Class C 能力的终端，通常应用于供电充足的场景，因此不必精简接收时间。

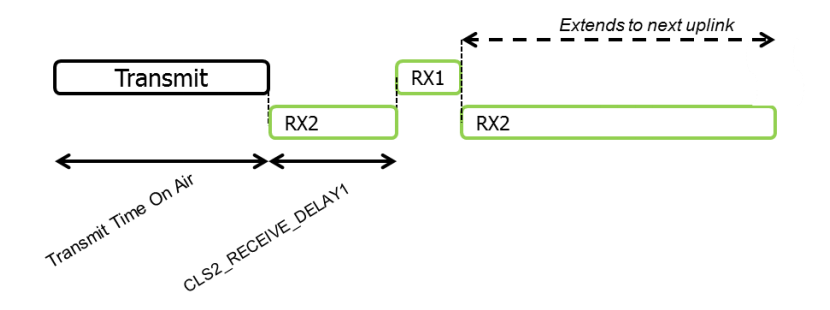
Class C 的终端不能执行 Class B 。

Class C 终端会尽可能地使用 RX2 窗口来监听。按照 Class A 的规定，终端是在 RX1 无数据收发才进行 RX2 接收。为了满足这个规定，终端会在上行发送结束和 RX1 接收窗口开启之间，打开一个短暂的 RX2 窗口，一旦 RX1 接收窗口关闭，终端会立即切换到 RX2 接收状态； RX2 接收窗口会程序打开，除非终端需要发送其他消息。

*注意：没有规定节点必须要告诉服务端它是 Class C 节点。这完全取决于服务端的应用程序，它们可以在 join 流程通过协议交互来获知是否是 Class C 节点。*

## 17.1 Class C 的第二接收窗口持续时间

Class C 设备执行和 Class A 一样的两个接收窗口，但它们没有关闭 RX2 ，除非他们需要再次发送数据。因此它们几乎可以在任意时间用 RX2 来接收下行消息，包括MAC命令和ACK传输的下行消息。另外在发送结束和 RX1 开启之间还打开了一个短暂的RX2窗口。

图13.Class C 终端的接收时隙时序图

## 17.2 Class C 对多播下行的处理

和 Class B 类似，Class C 设备也可以接收多播下行帧。多播地址和相关的 NWKSKEY 及 APPSKEY 都需要从应用层获取。Class C 多播下行帧也有相同的限制：

不允许携带MAC命令，既不能放在FOpts域中，也不能放在 port 0 的 payload 中，因为多播下行无法像单播帧那样具备相同的鲁棒性。

ACK 和 ADRACKReq 位必须要为0。MType 域需要为 Unconfirmed Data Down 类型的数值。

FPending 位表明有更多的多播数据要发送。考虑到 Classs C 设备在大部分时间处于接收状态，FPending位不触发终端的任何特殊行为。

这个目录包含了编译一个多通道基站库所需的源码。编译之后就会生成固定链接的libloragw.a。

lora\_gateway\libloragw\tst目录下还有不同子模块的[测试](http://lib.csdn.net/base/softwaretest" \t "_blank" \o "软件测试知识库)程序。

# HAL

## 1.1 HAL介绍

这部分也就是LoRa集中器的HAL层(LoRa concentrator Hardware Abstraction Layer)，它是个C库，让大家使用少量的C函数就可以对LoRa集中器芯片进行配置硬件，以及收发数据包。

LoRa集中器是数字化的多信道多数据包标准的射频芯片，使用LoRa或者FSK模式进行收发数据。

## 1.2 HAL的组成

这个库是由6(8)个模块组成：

·loragw\_hal

主模块，包含高等级函数来配置和使用集中器

·loragw\_reg

这个模块用来操作集中器的寄存器

·loragw\_spi

通过SPI接口来操作集中器的寄存器

·loragw\_aux

包含一个主机需要的wait\_ms函数，用于指定ms的延时

·loragw\_gps

通过基准时基来同步集中器内部计数，例如例程中的GPS授时。

·loragw\_radio

配置 SX125x 和 SX127x。

·loragw\_fpga (only for SX1301AP2 ref design)

SX1301AP2参考设计才需要，用于操作FPGA的寄存器，以及配置FPGA功能。

·loragw\_lbt (only for SX1301AP2 ref design)

SX1301AP2参考设计才需要，用于配置和使用LBT功能。

## 1.3 软件编译

### 1.3.1 软件细节

这个库按照ANSI C99进行编写。loragw\_aux模块中的ms精确延时含有POSIX格式函数，[嵌入式](http://lib.csdn.net/base/embeddeddevelopment" \t "_blank" \o "嵌入式开发知识库)平台可以用硬件定时器进行重写。

### 1.3.2 编译选项

library.cfg 中 DEBUG\_xxx 如果置为1，则会用 fprintf 输出对应的调试信息。

### 1.3.3 编译流程

对于交叉编译，需要设置 Makefile 中的 ARCH 和 CROSS\_COMPILE 变量，或者在 shell 环境中，使用正确的工具链名字和路径。例如：

export PATH=/home/foo/rpi-toolchain/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin:$PATH export ARCH=arm export CROSS\_COMPILE=arm-linux-gnueabihf-

libloragw目录下的Makefile会解析 library.cfg 文件，产生一个config.h的C头文件，包含 #define 选项。那些选项会使能或禁用loragw\_xxx.h 文件和 \*.c 原文件中的代码。

library.cfg 也用来直接选择动态链接库。

### 1.3.4 导出

如果想在其他系统使用编译后的库，你需要导出这些文件：

·libloragw/library.cfg -> 根配置文件

·libloragw/libloragw.a -> 静态库

·libloragw/readme.md -> license要求

·libloragw/inc/config.h -> 从 library.cfg 衍生出的C配置标志

·libloragw/inc/loragw\_\*.h -> 你需要用到的头文件 (例如. \_hal and \_gps)

在这个库链接到你的应用之后，只有 license 文件要求在程序文件中拷贝和保留。

## 1.4 硬件条件

### 1.4.1 硬件版本

loragw\_reg 和 loragw\_hal 是针对Semtech硬件编写的特殊版本：

·Semtech SX1301 芯片

·Semtech SX1257 or SX1255 收发器

如果硬件版本和库版本不匹配的话，这个库将无法使用。你可以用 test\_loragw\_reg 来测试软硬件是否匹配。

### 1.4.2 SPI通信

loragw\_spi 的SPI函数适合平台相关的，如果你用别的SPI接口可能需要重写这个函数：

·SPI master matched to the Linux SPI device driver (provided)

·SPI over USB using FTDI components (not provided)

·native SPI using a microcontroller peripheral (not provided)

你可以用 test\_loragw\_spi 来测试SPI通信。

### 1.4.3 GPS接收

为了使用库中的GPS模块，主机必须要通过串口连接GPS接收器，串口连接必须以“tty”设备出现在 /dev/ 目录，启用这个程序的用户必须用读写这个设备的权限。使用 chmod a+rw 来允许所有用户能操作指定的tty设备，或者使用sudo来运行你的程序(例如. sudo ./test\_loragw\_gps)。

当前版本，库只从串口读取数据，在GPS接收器上电后会收到他们发出NMEA帧 以及 u-blox 模块私有的 UBX 消息。

GPS接收器必须在发出PPS脉冲后发出UBX消息，让内部集中器的时间戳可以用GPS时基校准。如果GPS接收器发出了GGA NMEA语句，gateway则可以进行3D定位。

## 1.5 使用

### 1.5.1 设置软件环境

对一个典型应用，你需要这么做：

·源码中包含 loragw\_hal.h

·编译时链接 libloragw.a 静态库文件

·由于 loragw\_aux 的依赖关系，需要链接 librt 库

·如果应用需要直接访问集中器配置寄存器的话(例如做些高级配置)，你还需要这样做：

·源码中包含 loragw\_reg.h

### 1.5.2 使用软件API

要在你的应用中使用 HAL，需要遵守如下规则：

·在射频启动之前需要配置好 radios path 和 IF+modem path

·只有在调用了 start 函数之后，配置才会传送给硬件

·只有在 radio 使能，同时IF+modem 使能，以及集中器启动后，才能接收数据包。

·只有在 radio 使能，以及集中器启动后，才能发送数据包。

·改变配置之前，必须停止集中器。

一个对HAL的典型应用流程图如下：

<configure the radios and IF+modems>

<start the LoRa concentrator>

loop {

<fetch packets that were received by the concentrator>

<process, store and/or forward received packets>

<send packets through the concentrator>

}

<stop the concentrator>

*/!\ 注意：lgw\_send 在LoRa集中器仍然发包时，或者即使在准备开始发包时，是非阻塞立即返回。当有数据包在发送时，将无法收到任何数据。*

你的应用需要考虑发包的时长，或者在尝试发包前检查下状态(使用 lgw\_status)。

当前一包未完成时立即发一包，会导致前一包无法发送，或者发送部分(会导致接收端出现CRC错误)。

### 1.5.3 调试模式

为了调试程序，可以激活调试信息后( 在 library.cfg 中设置 DEBUG\_HAL=1 )，编译 loragw\_hal 函数。这样就会输出很多细节信息，包括stderr的错误细节信息。

# 2.帮助程序

工程中的这些程序提供了一些示例，应该如何使用HAL库。帮助系统构建者单独测试不同部分。

## 2.1. util\_pkt\_logger

This software is used to set up a LoRa concentrator using a JSON configuration

file and then record all the packets received in a log file, indefinitely, until

the user stops the application.

这个软件用来让LoRa集中器使用JSON配置文件，以及记录所有的包于一个log文件，除非用户停止这个应用。

## 2.2. util\_spi\_stress

This software is used to check the reliability of the link between the host

platform (on which the program is run) and the LoRa concentrator register file

that is the interface through which all interaction with the LoRa concentrator

happens.

这个软件用来检测主CPU与LoRa协调器寄存器文件的连接的稳定性。

## 2.3. util\_tx\_test

This software is used to send test packets with a LoRa concentrator. The packets

contain little information, on no protocol (ie. MAC address) information but

can be used to assess the functionality of a gateway downlink using other

gateways as receivers.

这个软件用来做发包测试。包里没有协议信息，但可以用来检测基站下行功能，使用另一台基站来做接收。

## 2.4. util\_tx\_continuous

This software is used to set LoRa concentrator in Tx continuous mode,

for spectral measurement.

这个软件用来设置LoRa集中器为持续TX模式，用于频谱测试。

## 2.5. util\_spectral\_scan

This software is used to scan the spectral band in background, where the

LoRa

这个软件用来扫描基站工作环境的频段。

## 2.6. util\_lbt\_test

This software is used to test "Listen-Before-Talk" channels timestamps.

这个软件用来测试“Listen-Before-Talk”的信道时间戳。

# 3. 帮助脚本

## 3.1. reset\_lgw.sh

This script must be launched on IoT Start Kit platform to reset concentrator

chip through GPIO, before starting any application using the concentrator.

这个脚本仅在 IoT Start Kit 平台上运行，用于在启动任何应用前，通过GPIO复位集中器芯片。