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。

twowinter 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）：

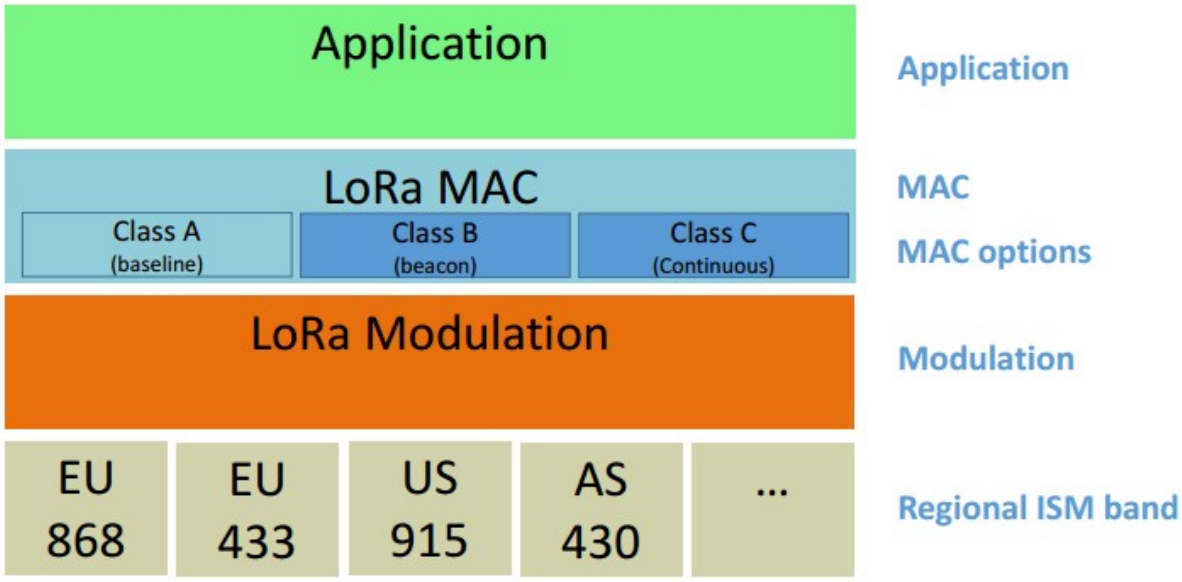


图 1.LoRaWAN Classes

- **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:

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:

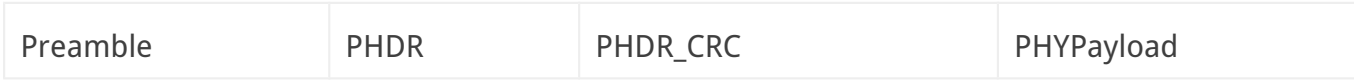


图 3.下行 PHY 帧格式

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。

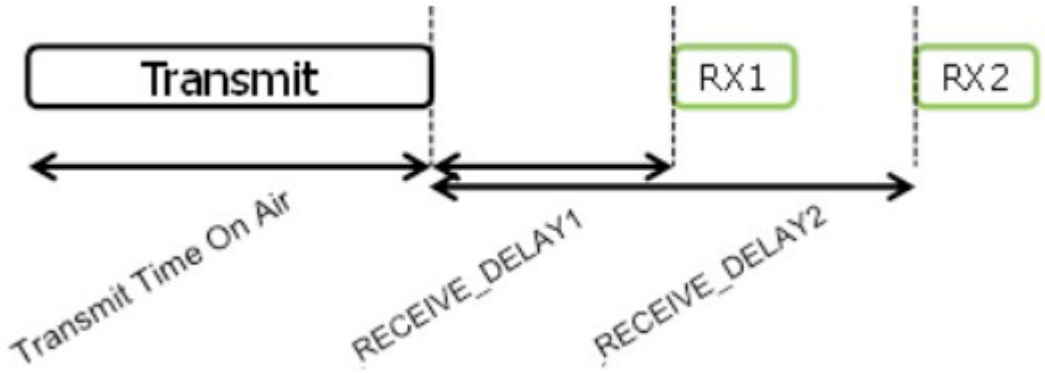


图 4.终端接收时隙的时序图

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

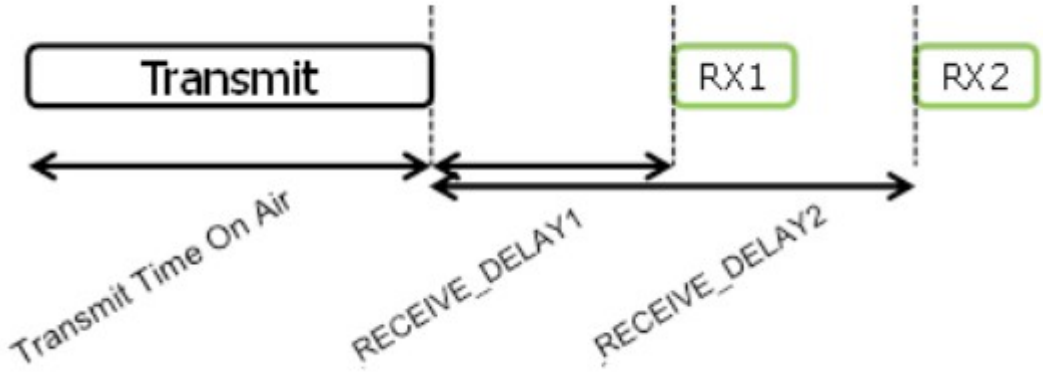
终端在第一或第二接收窗口收到下行消息后，或者在第二接收窗口阶段，不能再发起另一个上行消息。

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。

2 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。

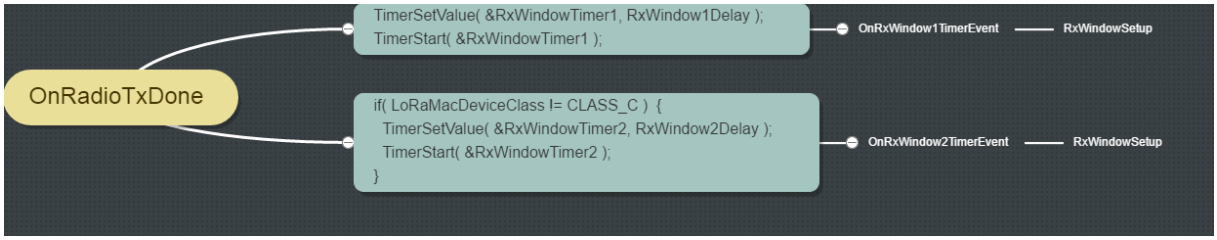


目前 RX1 一般是在上行后 1 秒开始，RX2 是在上行后 2 秒开始。

3 源码分析

3.1 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。



3.2 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 );
    }
  }
  ...
```

```
}
```

3.3 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) 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)

PHY 载荷：

MHDR	MACPayload	MIC

或者

MHDR	Join-Request	MIC

或者

MHDR	Join-Response	MIC

图 6.PHY 载荷结构

MAC 载荷：

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)
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**。

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，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	MIC	CRC
MAC层	Preamble	PHDR	PHDR_CRC	MHDR	MACPayload						MIC	CRC
PHY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：

1. 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。

2. 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。

3 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。

3.1 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。

3.2 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。

3.2.1 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;
  }
}
```

3.2.2 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。

3.3 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
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。

2 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。

3 代码位置

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;
```

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 );
  }
}
```

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。
本文作者 twowinter，转载请注明作者：http://blog.csdn.net/iotisan/

LinkADRReq 的源码解析

按照代码思路走一遍。

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。

2.解析 ChMask 字段

```
chMask = ( uint16_t )payload[macIndex++];
chMask |= ( uint16_t )payload[macIndex++] << 8;
```

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。

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.判断速率有效性

```
if( ValidateDatarate( datarate, channelsMask ) == false )
{
  status &= 0xFD; // Datarate KO
}
```

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
}
```

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;
}
```

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)。

2 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。

(twowinter，你数数，这是一句话?)
(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。)

3 代码位置

3.1 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;
}
```

3.2 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
本尊有机会接触了几个 LoRaWAN 基站厂家，发现大家为了调试方便，一般也会支持这些默认值。

```
/*!
 * 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, 0
```

LoRaWAN 地区参数

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。

2 LoRaWAN 地区参数

2.1 欧洲 863-870MHz 免授权频段

待补充，计划 3 月份补足。

2.2 美国 902-928MHz 免授权频段

待补充，计划 3 月份补足。

2.3 中国 779-787MHz 免授权频段

待补充，计划 3 月份补足。

2.4 欧洲 433MHz 免授权频段

待补充

2.5 澳洲 915-928MHz 免授权频段

待补充

2.6 中国 470-510MHz 频段

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

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。

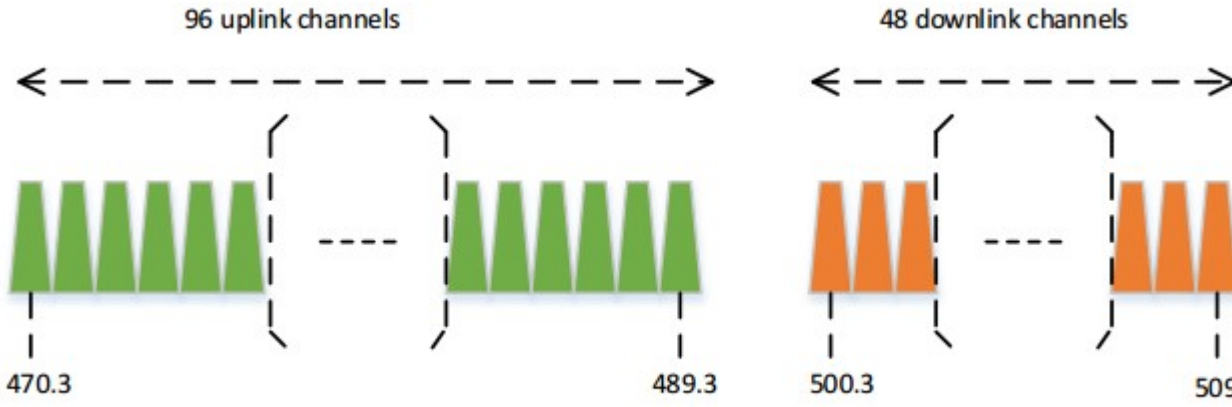


Figure 3: CN470-510 channel frequencies

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**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。

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

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。

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。

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

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。

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。

2.7 亚洲 923MHz 免授权频段

待补充

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

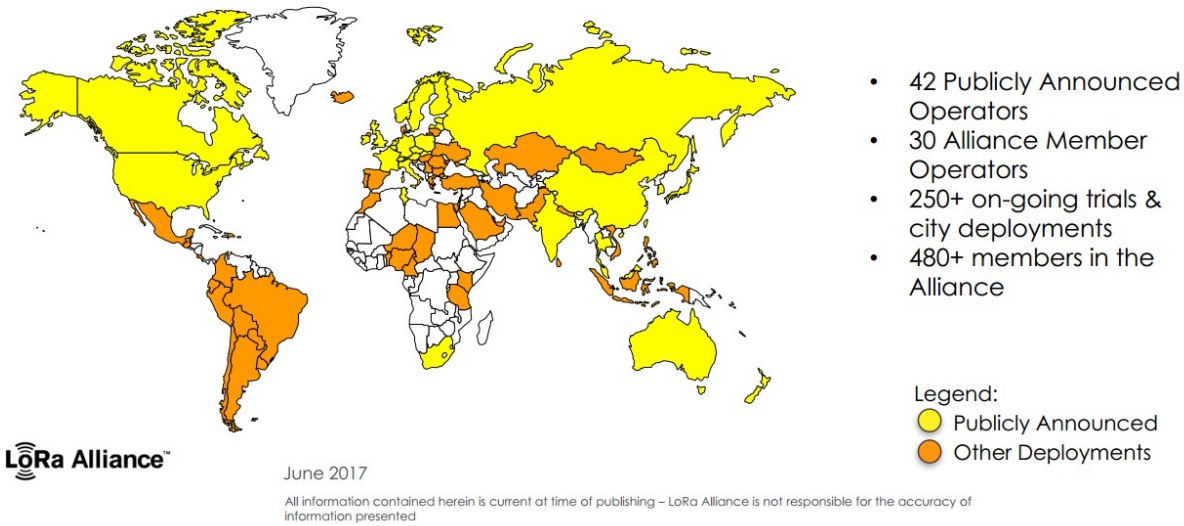
regulatory compliance.

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)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。
本文作者 twowinter，转载请注明作者：http://blog.csdn.net/iotisan/

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**；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

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。

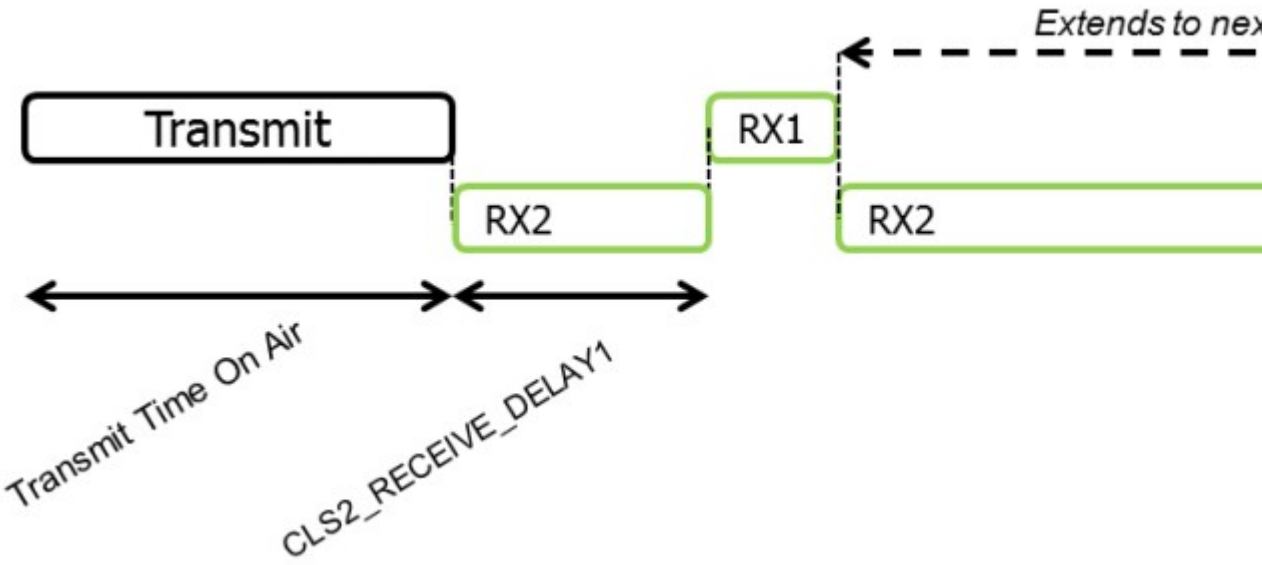


图 13.Class C 终端的接收时隙时序图

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**程序。

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。