LoRa网关双通道版本下行发送问题分析

葛鑫 2019-1-2

一些错误现象列表

- 服务器指定下行不会发送
- 上行频率解析错误
- 大量包的CRC教研无法通过
- `SPI0`通道能够收到434.575Mhz的包
- 从`SPI0`收到的包的频率都是`433.575`或`434.575`(SPI1的接收解析看起来是正确的)
- `SPI_1`能收到大量本该从`SPI_0`收的包

配置文件修改

感觉根源问题来自于网关的射频，也就是HAL层频率参数配置，频率参数配置又来自于网关配置文件。所以现在开始从网关配置文件至上而下寻找问题。

每一个LoRa频道在配置文件中对应了一系列参数，例如

```
"chan_multiSF_1": {
            /* Lora MAC channel, 125kHz, all SF, 434.375 MHz */
            "enable": true,
            "radio": 1,
            "if": -200000
        },
```

里面的对应参数没有官方解释，所以只能根据字面意义推断

`enable`：这个频道是否使用

`radio`：配置在哪片1255上（每一个射频有2片1255）

`if`：频率偏移（对于中心频率）

因为`radio_0`的中心频率是`433.575`，`radio_1`的中心频率是`434.575`，因此这个频道的对应频率就是`434.575Mhz - 200000 = 434.375Mhz`

修改的配置文件情况如下

- `SPI_0`通道打开了`433.175`,`433.375`，`433.575`，`433.775`四个频道，其中前两个使

用`radio_0`，后两个使用`radio_1`
- `SPI_1`通道打开了`433.975`，`434.175`，`434.375`，`434.575`四个频道，其中前两个使用`radio_0`，后两个使用`radio_1`

网关上层配置顺序

网关单通道版本`main`函数大致执行过程：

- 加载并配置相关配置文件
- 设置socket
- 调用`lgw_start()`启动网关
- 创建上行、下行、队列、时间同步线程
- 进入`while`循环，打印固定时间间隔打印网关参数。

双通道版本的`main`函数大致执行过程：

- 加载并配置针对于`SPI_0`的配置文件
- 设置socket
- 调用`lgw_start(0)`启动`SPI_0通道`
- 加载并配置针对于`SPI_1`的配置文件
- 调用`lgw_start(1)`启动`SPI_1`通道
- 创建**上行两个线程**，以及下行，队列，时间同步线程。
- 进入`while`循环，打印固定时间间隔打印网关参数。

配置部分

射频部分如何进行配置?

1. 读取配置文件
2. 网关上层代码解析配置文件，将射频参数写入数组
3. 网关HAL层代码得到数组，通过对应`SPI`读写寄存器，将参数配置到射频模块

所以现在来看一下，这些步骤到底是怎么做的

解析配置文件部分

它由`lora_pkt_fwd.c`中的`parse_SX1301_configuration(path)`函数完成，配置文件本身是一个JSON文件。这里只分析关于射频部分的参数它是怎么解析的。

- RF_chain的解析

    - 若对应radio的`enable`为`false`，什么也不做
    - 否则将`freq_hz`，`rssi_offset`，`type`等参数解析到一个`rfconf`的结构体中
    - 若此radio的发送`tx_enable`为`true`，则需要配置`tx_freq_min`和`tx_freq_max`，否则程序无法启动
    - 用`lgw_rxrf_setconf(i, rfconf)`将此结构体传给HAL层
- 频率配置（LoRa multi-SF channels）解析
    - 所有的参数配置在一个`ifconf`的结构体中
    - 若某个频道的`enable`参数是`false`，则不做解析，否则解析`rf_chain`，`freq_hz`两个参数
    - 将此结构体用`lgw_rxif_setconf(i, ifconf)`传给网关HAL层

不管是对于RF_chain的配置，还是频道的配置，在循环的开始都有`memset(&rfconf, 0, sizeof rfconf)`和`memset(&ifconf, 0, sizeof ifconf)`因此，**不会出现后配置的参数覆盖前者的bug**，问题应该基本确定没有出在这里。

HAL层配置rfconf和ifconf部分

接下来看一下上述两个函数`lgw_rxrf_setconf(i, rfconf)`和`lgw_rxif_setconf(i, ifconf)`是怎么做的

- HAL层的RF配置
    - 下层拿到`rfconf`结构体后，做一些错误处理部分，接下来

```
    /* set internal config according to parameters */
    rf_enable[rf_chain] = conf.enable;
    rf_rx_freq[rf_chain] = conf.freq_hz;
    rf_rssi_offset[rf_chain] = conf.rssi_offset;
    rf_radio_type[rf_chain] = conf.type;
    rf_tx_enable[rf_chain] = conf.tx_enable;
    rf_tx_notch_freq[rf_chain] = conf.tx_notch_freq;
```

- HAL层的IF配置
    - 拿到`ifconf`结构体后，做一些错误处理
    - 将`conf.bandwidth`赋值给`rf_rx_bandwidth`变量

```
switch(ifmod_config[if_chain]) {
    case IF_LORA_STD:
        ...
        if_enable[if_chain] = conf.enable;
        if_rf_chain[if_chain] = conf.rf_chain;
        if_freq[if_chain] = conf.freq_hz;
        lora_rx_bw = conf.bandwidth;
```

```
        lora_rx_sf = (uint8_t)(DR_LORA_MULTI & conf.datarate);

    case IF_LORA_MULTI:
        ...
        if_enable[if_chain] = conf.enable;
        if_rf_chain[if_chain] = conf.rf_chain;
        if_freq[if_chain] = conf.freq_hz;
        lora_multi_sfmask[if_chain] = (uint8_t)(DR_LORA_MULTI &
conf.datarate);

    case IF_FSK_STD:
        ...
}
```

所以说，最重要的几个参数`enable`，`rf_chain`，`freq_hz`都被配置到了对应数组中`if_enable[if_chain]`，`if_rf_chain[if_chain]`，`if_freq[if_chain]`。但是**这几个数组是全局数组**。

所以错误在这里

```
数组  SPI_0 | SPI_1
0     x      ✔
1     x      ✔
2     x      ✔
3     ✔      x          这是本来想要达成的效果
4     ✔      x
5     ✔      x
6     ✔      x
7     x      ✔


数组  SPI_0 | SPI_1
0     x       ✔
1     x       ✔
2     x       ✔
3     ✔   ->  ✔          由于是全局数组，3-6频道没有进行更改，没有覆盖前者
4     ✔   ->  ✔
5     ✔   ->  ✔
6     ✔   ->  ✔
7     x       ✔
```

在2个`setconf()`函数完成后，根据`main`的执行过程可知道，下一步是调用`lgw_start()`启动射频。所以来看一下`lgw_start()`的启动过程。

lgw_start()的启动过程

- 调用`lgw_connect()`连接，设置spi的读写模式以及一些状态参数。

- 做了很多看不懂的寄存器读写操作
- 有用到`if_enable[]`，`if_rf_chain[]`，`if_freq[]`数组里的参数去读写寄存器

结论

射频配置参数从配置文件-上层解析-HAL层，层层传递，到HAL层时被配置在了全局数组中，用来读写射频寄存器。所以上层调用时会出现配置bug。因为这些参数也是解析上行包的频率的基准，所以会计算错上行频率。与现实情况比较符合。

下一步工作

- 数组开成全局是有原因的，最大的好处是，有很多HAL层函数需要用到这些数组中的参数完成自己的功能。因此改成局部数组来修改此bug的方式比较麻烦，需要函数间多传递一个数组参数，可能还会造成潜在错误。
- 第二是再开一组对应的数组，带来的问题是，代码量翻倍的工作，要不复制粘贴一遍 hal层的所有函数，根据对应`port`配置不同数组，要不就需要在所有需要用到数组的地方加`if else`做一个`port`值判断。

由于整个HAL层代码量大，函数多，改起来需要一些时间。