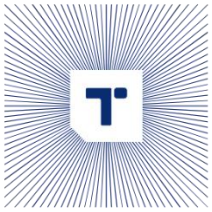


Telink

泰凌 SIG Mesh SDK 开发手册

AN-17120401-C4

Ver.1.3.0

2020/05/29

摘要

本文为泰凌 SIG Mesh SDK 的开发手册

Published by

Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,

Zhangjiang Hi-Tech Park, Shanghai, China

Legal Disclaimer

Information

For further information on the technology, product and business term, please contact Telink Semiconductor Company (www.telink-semi.com).

For sales or technical support, please send email to the address of:

telinkcnsales@telink-semi.com

telinkcnsupport@telink-semi.com

修改记录

Version 1.3. 0 (2020-05)

第四次发布。 和上一版相比， 以下章节有更新：

- ✧ 删除 draft feature

Version 1.2. 0 (2020-04)

第三次发布。 和上一版相比， 以下章节有更新：

- ✧ 1 SDK 概述
- ✧ 4 调试工具操作说明
- ✧ 11 Mesh LPN 工程介绍
- ✧ 8 全局配置文件说明

新增以下章节：

- ✧ 2 MCU 基础模块
- ✧ 5 厂测模式
- ✧ 6 SDK 常用的几个模块
- ✧ 7 Vendor model 的使用
- ✧ 9 8258 MESH 工程介绍
- ✧ 13 平台接入
- ✧ 14 恢复出厂配置
- ✧ **错误!未找到引用源。 错误!未找到引用源。**
- ✧ 15 Fast bind 模式(即 PROVISION_FLOW_SIMPLE_EN 模式)
- ✧ 16 私有 Fast provision 功能
- ✧ 17 私有 online status 功能演示
- ✧ 18 OTA 简要测试说明
- ✧ 19 网络分享
- ✧ 20 通过 INI 控制节点实例讲解

Version 1.1.0(2018-7)

第二次发布， 和上一版相比，以下章节有更新:

1. SDK 概述
2. 全局配置文件说明
3. 8269 MESH 工程介绍
4. Provisioner(Gateway)工程介绍
5. SWITCH 工程介绍

Version 1.0.0 (2017-12)

第一次发布。

目录

图片列表

表格列表

1. SDK 概述

SDK 给用户提供基于 SIG_mesh 协议应用开发的 demo code，用户可以在这些 demo code 基础上开发自己的应用程序。

目前工程适用的 IC 部分为 825x/8278/8269。泰凌微提供了上位机工具(sig_mesh_tool.exe)，通过 usb 与 mast dongle 连接实现 provisioner 的功能，master dongle 目前仅支持 8269 芯片型号

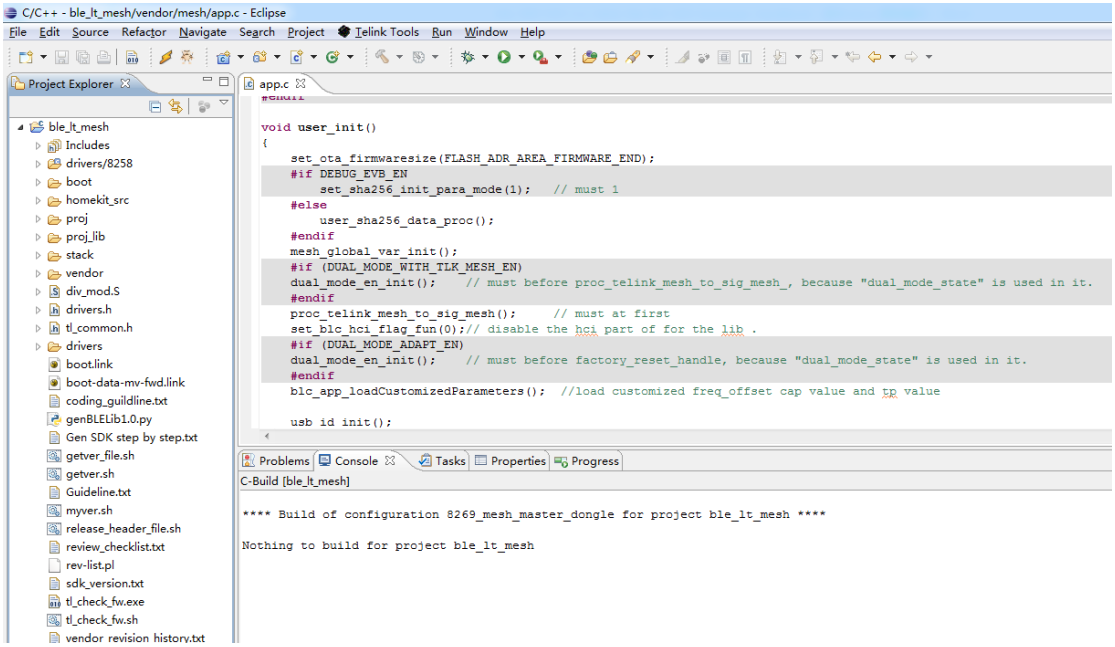
如果需要快速了解和演示基本的功能，请参考：

http://wiki.telink-semi.cn/wiki/solution/BLE-SIG-Mesh-Quick-Start/

1.1 SDK 的文件架构

SDK 文件架构分为 app 应用层和 BLE&SIG_mesh 协议层。当导入工程文件后，(工程导入方法请参考<AN_IDEUG-E1_Telink IDE User Guide.pdf>,新建工程参考文档<AN_16063000-E1_Guide For Adding New Project On Existing SDK.pdf>)，显示的文件架构如下图所示。有几个主要的顶层文件夹:boot，common，drivers，homekit_src ，proj_lib，stack，vendor 。

图 1-1 文件架构图



- **boot:**提供芯片的 bootloader，即 MCU 上电启动或 deepsleep 唤醒后的汇编处理过程，为后面 C 语言程序的运行搭建好环境。
- **drivers**:提供与 MCU 紧密相关的硬件设置和外设驱动程序，如 clock、flash、i2c、usb、gpio、uart 等。
- **proj**:提供 MCU 相关的外设驱动程序，如 flash，i2c，usb，gpio, uart 等。
- **proj_lib**: 提供 MCU 运行所必需的库文件，包括 BLE 协议栈、RF 驱动、PM 驱动等，这部分是以库文件形式提供的，用户无法看到源文件，如 libIt_8258_mesh.a 为蓝牙协议栈的

库文件，libsig_mesh.a 为 SIG_mesh 普通节点的库文件，libsig_mesh_LPN.a 为 SIG_mesh 中的低功耗节点的库文件，libsig_mesh_prov.a 为 SIG_mesh 中的 provision 节点的库文件。

- ✧ **stack**:存放 BLE 协议栈相关的头文件。源文件被编译到库文件里面，对于用户是不可见的。
- ✧ **vendor**:用于存放用户应用层代码。目前 vendor 目录下有:

——8267_master_kma_dongle:上位机测试使用，配合 GATT 模式上位机工具可以作为一个 provisioner 的角色，用于演示和 debug。

——common:主要包含了 mesh/mesh_lpn/mesh_provision/mesh_switch 等共用的模块，例如 SIG mesh model 的处理，led 部分，出厂初始化 ，测试命令等模块。

——mesh/mesh_gw_node_homekit/mesh_lpn/mesh_provision/mesh_switch/ spirit_lpn 这几个文件夹的结构一样，每个文件夹对应一个应用类型，都包含了 app.c、app.h、app_att.c、app_config.h、main.c。app.c/app.h 主要是初始化和底层回调功能；app_att.c 是蓝牙 att 表的描述以及接口函数的说明；app_config.h 是定义工程中对应的宏和声明；main.c 是主函数和中断函数的入口。

1.1.1 main.c

包括 main 函数入口，系统初始化的相关函数，以及无限循环 while(1)的写法，建议不要对此文件进行任何修改，直接使用固有写法。

```
int main (void) {
FLASH_ADDRESS_CONFIG;
#if PINGPONG_OTA_DISABLE
     ota_fw_check_over_write();   // 非 pingpong OTA 的 firmware copy
#endif
blc_pm_select_internal_32k_crystal(); //选择内部 32k rc 作为 32k counter 时钟源
  cpu_wakeup_init();//MCU 最基本的硬件初始化
int deepRetWakeUp = pm_is_MCU_deepRetentionWakeup();   //判断是否从 deep retention 唤醒
  rf_drv_init(RF_MODE_BLE_1M); //RF 初始化
  gpio_init(!deepRetWakeUp); //gpio 初始化，user 在 app_config.h 中配置相关参数
        clock_init(SYS_CLK_16M_Crystal);
       if( deepRetWakeUp ){
             user_init_deepRetn ();//deep retention 醒来的快速初始化
       }
       else{
       user_init_normal ();//ble 初始化，整个系统初始化，user 进行设定
 }
     irq_enable();            //开全局中断
 while (1) {
#if (MODULE_WATCHDOG_ENABLE)
                wd_clear(); //clear watch dog
#endif
       main_loop ();  //包括 ble 收发处理、低功耗管理、mesh 和 user 的任务
 }
}
```

1.1.2 app_config.h

用户配置文件，用于对整个系统的相关参数进行配置，包括 BLE 相关参数、GPIO 的配置、PM 低功耗管理的相关配置等。

后面介绍各个模块时会对 app_config.h 中的各个参数的含义进行详细说明。

1.1.3 BLE stack entry

Telink BLE SDK 中 BLE stack 部分 code 的入口函数有两个:

1) main.c 文件 irq_handler 函数中 BLE 相关中断的处理入口 `irq_blt_sdk_handler`。

```
_attribute_ram_code_ void irq_handler(void)
{
        ......
        irq_blt_sdk_handler ();
        ......
}
```

2) application file main_loop 中 BLE 逻辑和数据处理的函数入口 `blt_sdk_main_loop`。

```
void main_loop (void)
{
 mesh_loop_proc_prior();//优先处理,主要是跳过 mailoop 8269 的 10ms interval
////////////////////// BLE entry /////////////////////////////
        blt_sdk_main_loop();

////////////////////// UI entry /////////////////////////////
factory_reset_cnt_check();//五次上电执行 factory reset 的实现
mesh_loop_process();//mesh 相关的 loop 函数
        ......

////////////////////// PM configuration /////////////////////
        ......
}
```

1.2 Demo Project

TeLink SIG MESH SDK 给用户提供了多个 BLE demo。

用户通过软硬件 demo 的运行，可以观察到直观的效果。用户也可以在 demo code 上进行修改，完成自己的应用开发。

图 1-2 MESH SDK 提供的 demo code

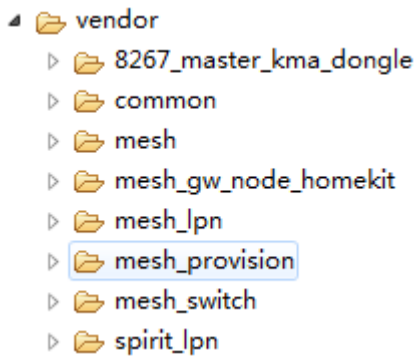
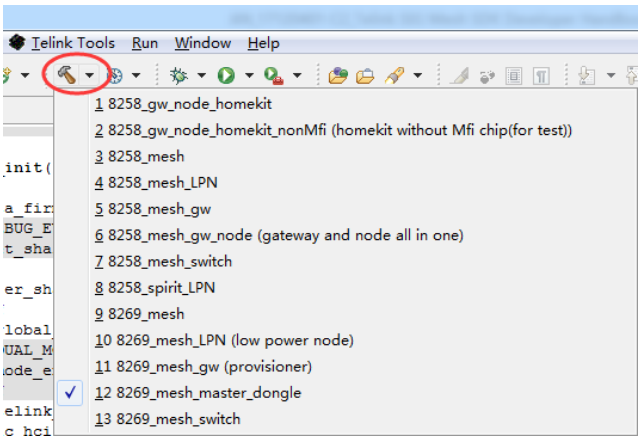


图 1-3 MESH SDK 的编译选项



mesh 的 demo 及区别如下表所示。

表格 1-1 Mesh Demo 区别

Demo	Vendor folder	Application	Mesh Feature
8258_mesh/ 8269_mesh	.\mesh	CT/HSL light 等	Relay, friend, proxy
8258_mesh_LPN\ 8269_mesh_LPN	.\mesh_lpn	LPN	LPN, proxy
8258_mesh_gw\ 8269_mesh_gw	.\mesh_provision	Gateway provisioner	adv provisioner, Relay, friend
8258_mesh _gw_node	.\ mesh_provision	Gateway+light node	adv provisioner, Relay, friend, proxy
8258_mesh_switch\ 8269_mesh_switch	.\mesh_switch	遥控器应用	proxy
8258_spirit_LPN	.\ spirit_lpn	天猫精灵自定义 LPN 模式	proxy
8258_gw_node_ho mekit	.\mesh_gw_node _homekit	Gateway+light node+homekit	adv provisioner, Relay, friend, proxy
8269_mesh_master _dongle	.\8267_master_k ma_dongle	Tool	GATT provisioner

✧ “8269_mesh_master_dongle”编译选项:有 GATT provisioner 功能，无 relay，friend 功能。上位机测试使用，配合 GATT 模式上位机工具可以作为一个 provisioner 的角色，用于演示和 debug。

- ✧ “8258_mesh”，“8269_mesh”编译选项:是普通 SIG MESH 节点的编译工程，可以被 provisioner 配置网络，有 relay，friend，proxy 功能，无 provision 功能。
- ✧ “8258_mesh_LPN”，“8269_mesh_LPN”编译选项，是 LPN MESH 节点的编译工程，通过 friend ship 接收 message，无 relay，friend，proxy，provision 功能。
- ✧ “8258_mesh_gw”，“8269_mesh_gw”编译选项:是 gateway provisioner 节点的编译工程，有 adv provisioner 功能，可以配置其他节点，同时也有 relay，friend 功能。
- ✧ “8258_mesh_switch”，“8269_mesh_switch”编译选项:是普通遥控器(switch) MESH 节点的编译工程。该遥控器组网后，为了实现更低的功耗需求，基本上只发命令，不接收命令。无 relay，friend 功能，
- ✧ “8258_gw_node_homekit”，“8258_gw_node_homekit_nonMfi”编译选项:具有 gateway adv provisioner + mesh node + homekit 这三种模式的功能。
- ✧ “8258_spirit_LPN” 编译选项:是天猫精灵自定义 LPN 模式。

1.3 LIGHT_TYPE_SEL 介绍

该宏用于选择预先配置好的一些常用的产品类型。

```
#define LIGHT_TYPE_NONE                 0
#define LIGHT_TYPE_CT                   1
#define LIGHT_TYPE_HSL                  2
#define LIGHT_TYPE_XYL                  3
#define LIGHT_TYPE_POWER                4
#define LIGHT_TYPE_CT_HSL               5
#define LIGHT_TYPE_DIM                  6   // only single PWM
#define LIGHT_TYPE_PANEL                7    // only ONOFF model
#define LIGHT_TYPE_LPN_ONOFF_LEVEL      8    // only ONOFF , LEVEL model
```

LIGHT_TYPE_CT

CT 是色温灯的简写，对应的产品类型是色温灯，包含色温相关的 model，比如 Light CTL Server，Light CTL Setup Server，Light CTL Temperature Server 以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_HSL

对应的产品类型是彩色灯(HSL 灯)，包含 Light HSL Server，Light HSL Hue Server，Light HSL Saturation Server，Light HSL Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_XYL

对应的产品类型是 XYL 灯，包含 Light xyL Server，Light xyL Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server，Light Lightness Server 等。

LIGHT_TYPE_POWER

对应的产品类型是 power adapter 等，包含 Generic Power Level Server，Generic Power Level Setup Server，以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server 等

LIGHT_TYPE_CT_HSL

对应的产品类型是 色温灯+彩色灯(HSL 灯)，包含色温以及 HSL 对应的 model，以及 Generic OnOff Server，Generic Level Server，Light Lightness Server 等，其中色温灯珠和 HSL 灯珠共用一个 lightness 和 onoff 值。另外，同一时间只有一种灯珠在点亮。

LIGHT_TYPE_DIM

对应的产品类型是调光灯，包含 Light Lightness Server，Light Lightness Setup Server 以及对应的 extend model，比如:Generic OnOff Server，Generic Level Server 等。

LIGHT_TYPE_PANEL

对应的产品类型是开关面板，该面板处于 server 角色，也就是被 app 等设备控制并执行 onoff 切换。包含 Generic OnOff Server model。默认开关个数是 3 个（由 LIGHT_CNT 定义）。

LIGHT_TYPE_LPN_ONOFF_LEVEL

对应的产品类型是 LPN 设备，默认包含 Generic OnOff Server model 等，mesh OTA model 关闭。主要用于 demo LPN 的功能。

Note:开发 LPN 设备时，要注意 825x retention RAM 的使用不能超过 32K。
Note:Node 最终包含的所有 model，都会显示在 composition data 中(全局变量:model_sig_cfg_s_cps)。

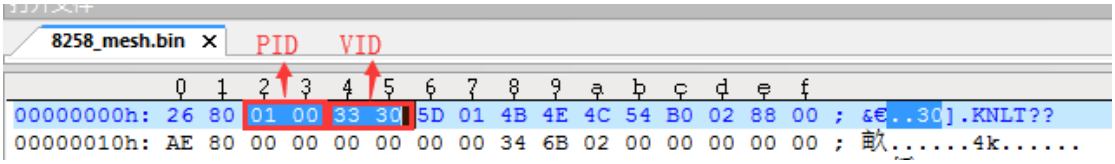
1.4 产品版本号(VID)产品类型(PID)设置

配置文件:\vendor\common\version.h，该文件在汇编代码中也会使用到。

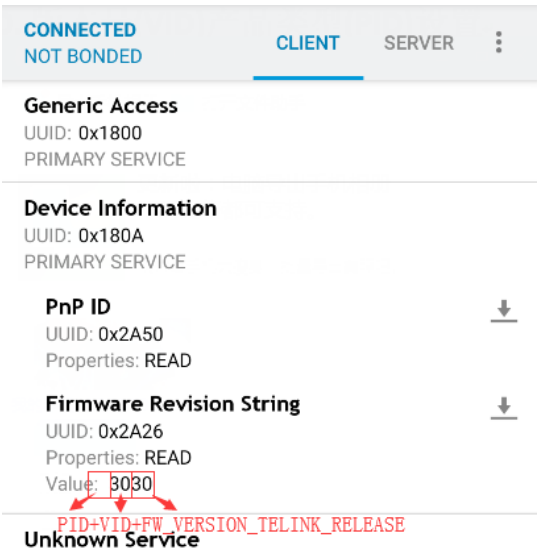
示例如下:

```
#define MESH_PID_SEL                        (LIGHT_TYPE_SEL)
#define MESH_VID                            (VERSION_GET(0x33, 0x30))
#define FW_VERSION_TELINK_RELEASE           (VERSION_GET(0x33, 0x30))
```

1) composition data 中的 PID 和 VID 分别取自这里的 MESH_PID_SEL，MESH_VID。
2) firmware 文件的第 3-6 字节分别表示这里的 PID 和 VID



3) 在通用 APP 的 ATT 页面的显示如下:



MESH_PID_SEL(PID)：因为通用 APP 上显示是以 ASCII 码解析，所以字符"0x00 0x01"这两个字符不可见。客户按实际需求修改为自己的 PID。

MESH_VID(VID)："0x33, 0x30"按 ASCII 显示为"30"。 客户按实际需求改为自己的 PID。

FW_VERSION_TELINK_RELEASE："0x33, 0x30"按 ASCII 显示为"30"。这个是 telink release SDK 时的版本号，客户不应该修改

4) 开发者按实际需求，修改 MESH_PID_SEL 和 MESH_VID 的值即可。

1.5 Mesh 应用收发包处理

1.5.1 发包函数

节点之间发包

调用 mesh_tx_cmd2normal_primary()，该函数具体用法见后面介绍。此方式发送的数据遵循 SIG mesh 协议。access_cmd_onoff()等命令是对 mesh_tx_cmd2normal_primary()进行封装而来。

开发者可以启用 sim_tx_cmd_node2node()来进行发送命令的演示(该函数默认是屏蔽的)，效果是:上电后，每隔 3 秒自动交替发送 ON/OFF 命令。详见后续对该函数的说明。

直连节点发给 master

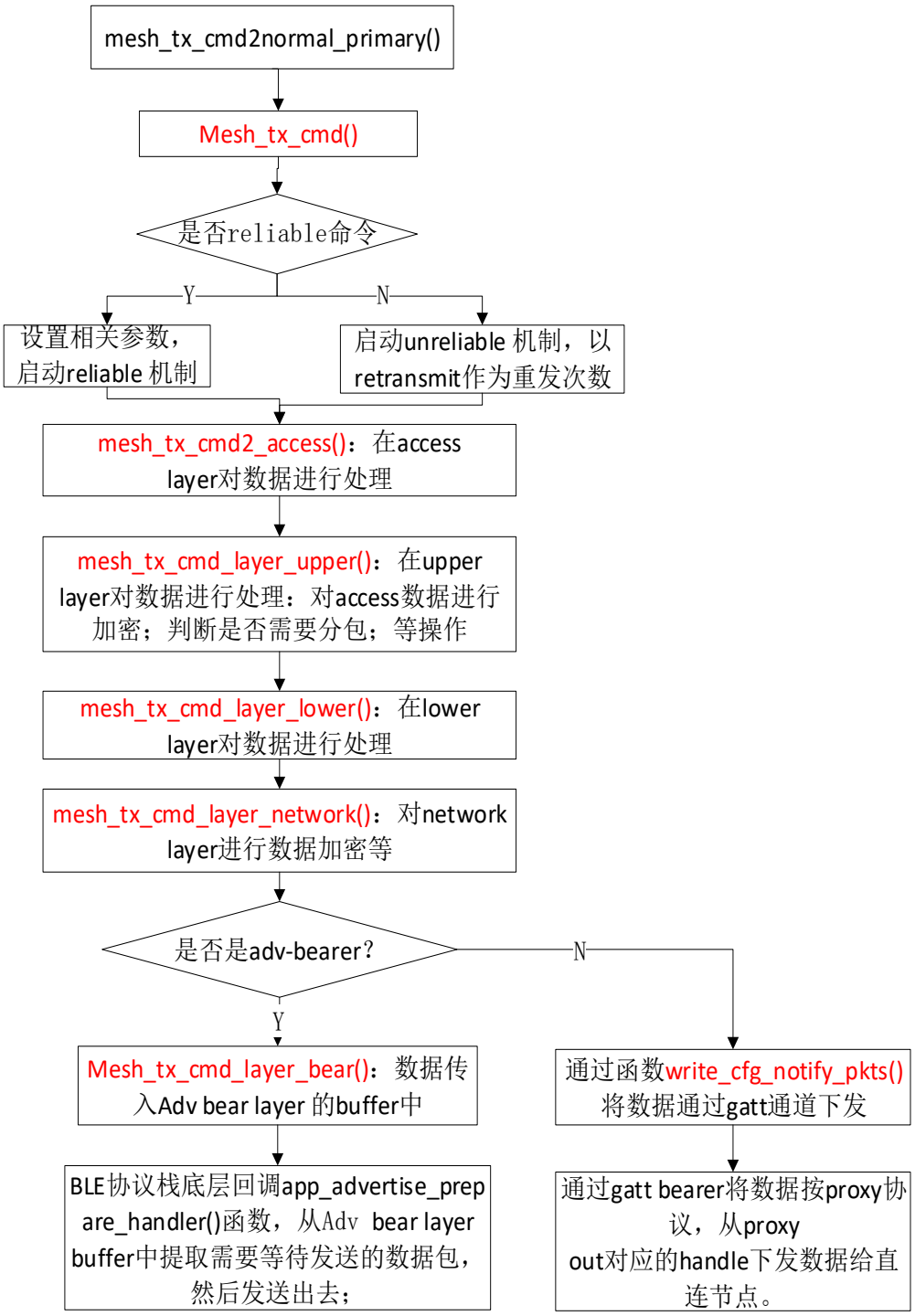
调用 bls_att_pushNotifyData()，具体用法请参考<AN_17092701_Telink 826x BLE SDK Developer Handbook> 3.4.3.10 节，此方式用于发送客户任意自定义格式的数据。但是没有 mesh 功能，一般不用。

注意:需要新增 UUID，然后才能用该方式，否则可能会和当前 UUID 的协议冲突。新增 UUID 的方法，可以在 my_Attributes_provision[]/my_Attributes_proxy[]/my_Attributes[]定义，自己订制 BLE service

1.5.2 发包流程图简介

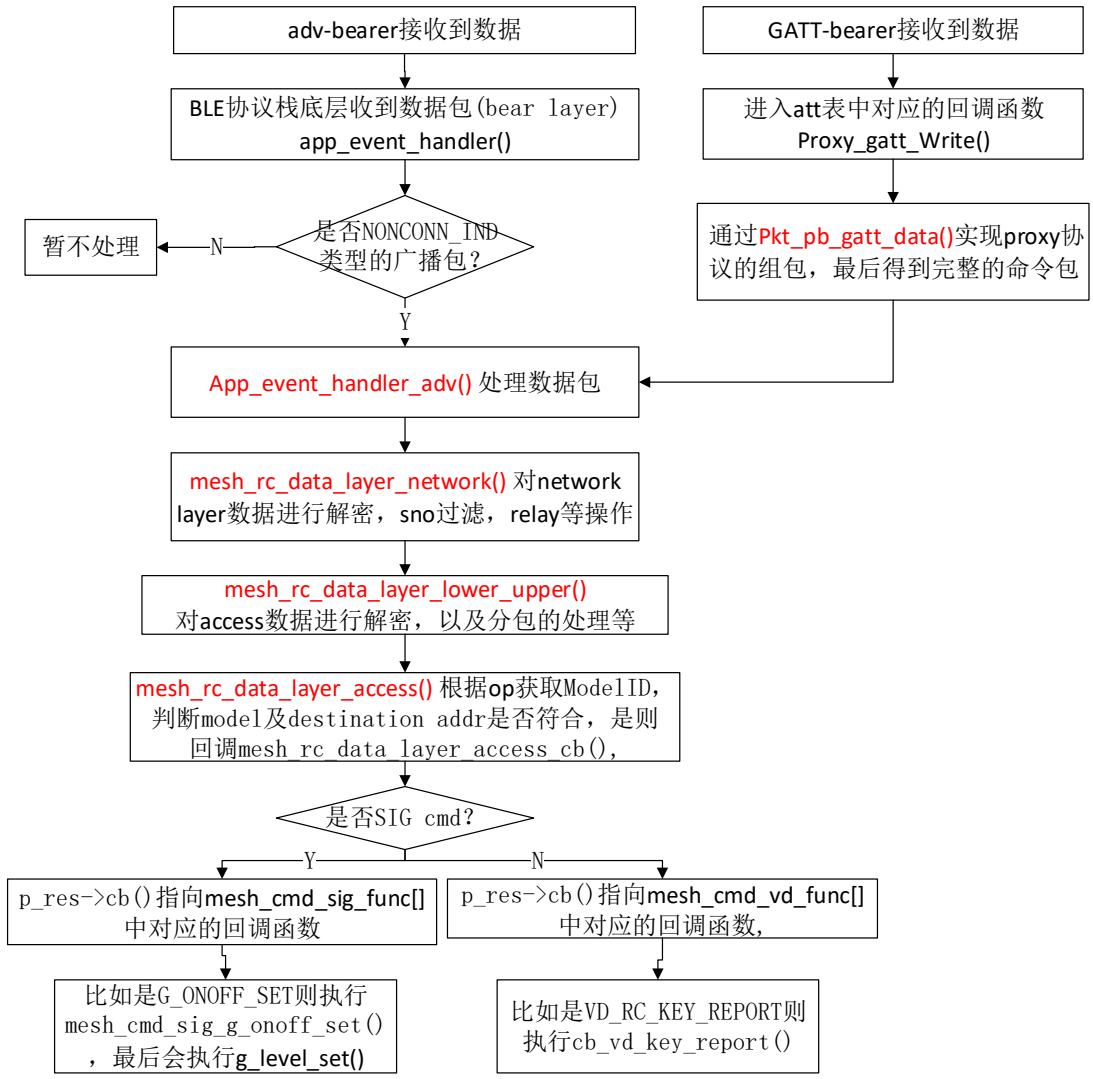
注:红色字体为 library 函数。

图 1-4 发包流程图


mesh_tx_cmd2normal_primary()
Mesh_tx_cmd()
是否reliable命令
Y
N
设置相关参数，启动reliable 机制
启动unreliable 机制，以retransmit作为重发次数
mesh_tx_cmd2_access()：在access layer对数据进行处理
mesh_tx_cmd_layer_upper()：在upper layer对数据进行处理：对access数据进行加密；判断是否需要分包；等操作
mesh_tx_cmd_layer_lower()：在lower layer对数据进行处理
mesh_tx_cmd_layer_network()：对network layer进行数据加密等
是否是adv-bearer?
N
Y
Mesh_tx_cmd_layer_bear()：数据传入Adv bear layer 的buffer中
通过函数write_cfg_notify_pkts()将数据通过gatt通道下发
BLE协议栈底层回调app_advertise_prepare_handler()函数，从Adv bear layer buffer中提取需要等待发送的数据包，然后发送出去；
通过gatt bearer将数据按proxy协议，从proxy out对应的handle下发数据给直连节点。

1.5.3 收包流程图简介

图 1-5 收包流程图



1.5.4 收包回调函数的介绍

generic model

generic model 的接口在文件 vendor/common/generic_model.c，回调函数见结构体 mesh_cmd_sig_func[]。比如 generic on message，收到这个命令后，按上面介绍的“收包流程图”，走到 mesh_cmd_sig_g_onoff_set()这个函数，用户在这个函数里面实现开关灯或设置渐变参数,渐变效果在 light_transition_proc 处理，处理间隔为 LIGHT_ADJUST_INTERVAL(20ms)。

vendor model

vendor model 的接口在文件 vendor/common/vendor_model.c。参考 mesh_cmd_vd_func[]，用户可以自行添加需要的 vendor command，收到这样的命令，按 SIG mesh spec 流程，会走到对应的 callback 函数，比如 cb_vd_key_report()。

1.5.5 SIG_mesh 信道

SIG_mesh 的通信信道分为两种：

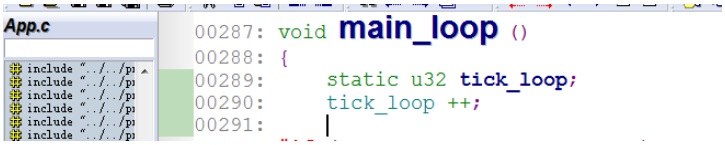
一种是 adv-bearer，是通过 adv 机制实现相互通信的，通信双方不需要建立 BLE 的连接，通讯 channel 是标准的 37/38/39；

另外一种是通过 gatt-bearer，通过建立 BLE 连接实现通信，通讯 channel 是 1-36。

1.6 telink debug 方法介绍

1.6.1 Tdebug Tool 调试

此方式可以稳定可靠的进行调试，并且不影响 mcu 运行,能实时查看和修改全局变量，也能查看函数的运行状态等,函数是否执行可以在函数里面定义一个全局变量,然后再执行计数操作，然后通过 Tdebug 查看这个全局变量即可.比如:

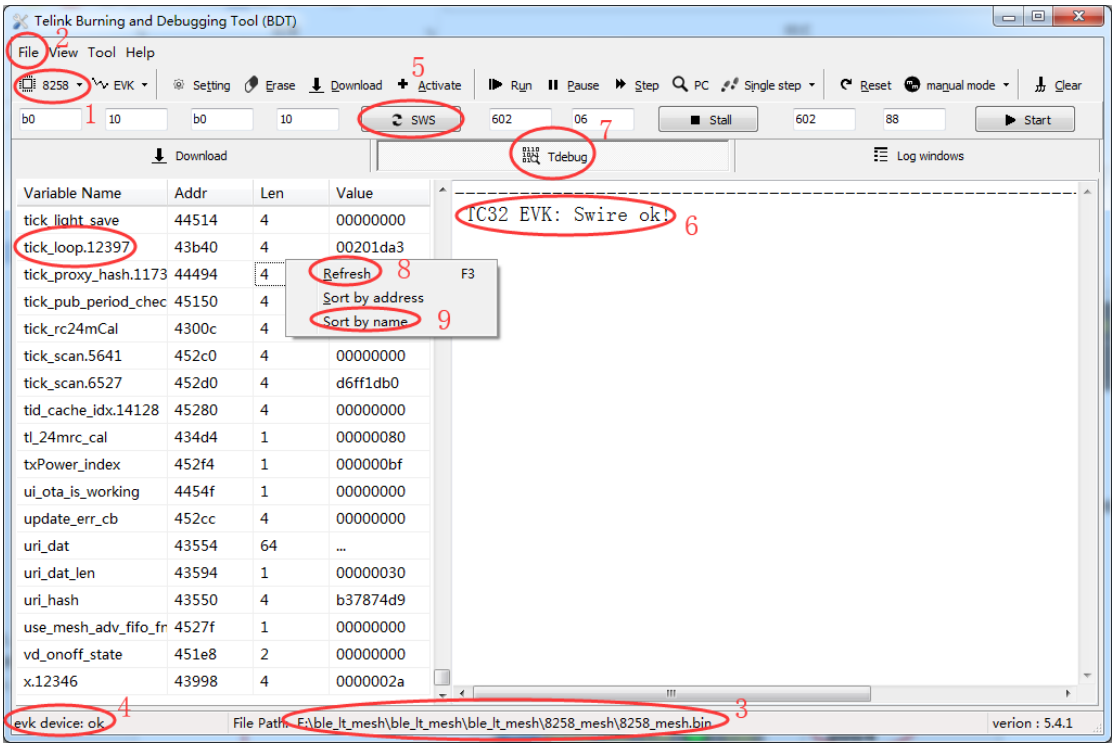


Tdebug 简介如下，详见 BDT 工具的“Help”→”User guide”的文档说明。

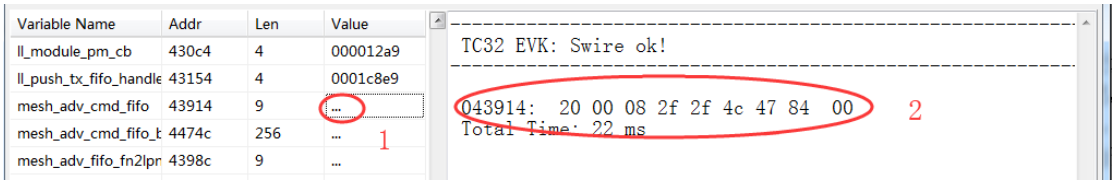
注:下图第 8 个 step 通过右键弹出该菜单。

第 9 个 step 按 name 排序后找到 tick_loop，(因为 tick_loop 是 static 变量，在代码中可能会重名，所以编译器增加了后缀.12397 做区分)，然后按右键，点击 Refresh，就会读取所有的全局变量并刷新，tick_loop 的值也会被刷新。

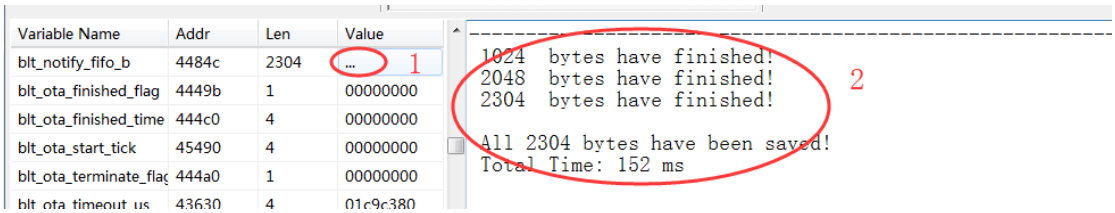
如果需要修改全局变量，在 value 一栏修改值，然后点回车即可。回读的话，按右键，点击 Refresh 即可。



大于 4 byte，小于 1K byte 的结构体变量或者数组，通过以下方式读取，点击 1 处的"..."，读到的值会显示在 2 的位置:



大于 1kbyte 的，会自动生成文件存放在 BDT 工具的 Telink Burning and Debugging Tool\config\user\Read.bin



1.6.2 Log 打印调试 Log Print Debugging

在 SDK2.9 版本后提供了通过 GPIO 模拟 uart 输出的方式进行 log 打印，默认是 1Mbps。此方法需要留意的是，为了保证 log 输出不出错，输出 log 期间默认是执行 irq_disable()，所以 log 数据过多有可能会影响 MCU 以及 RF 收包处理的速度等，也有可能会造成 mesh 的不稳定,反而不利于查找问题的.因此在使用时候尽可能的减少 log 的信息输出.调试完功能后把能关闭的 log 尽量都关闭.

Log 可以设置 print level(TL_LOG_LEVEL) 以及 print module(TL_LOG_SEL_VAL)。

print level:为了不把过多的 log 默认编译到 firmware 里面，TL_LOG_LEVEL 默认设置为 TL_LOG_LEVEL_ERROR，print level 需要小于等于 TL_LOG_LEVEL_ERROR 才会打印。TL_LOG_LEVEL_LIB 用于 library 里面的 code，或者用于打印非 library code 的重要 log。TL_LOG_LEVEL_USER 是供用户使用的类型，在 library 里面不会使用到。所以建议客户在打印的时候，使用 LOG_USER_MSG_INFO() 来打印。如果需要使用 LOG_MSG_INFO()等接口，需要修改 TL_LOG_LEVEL。

```
#define TL_LOG_LEVEL_DISABLE        0
#define TL_LOG_LEVEL_USER           1U      // never use in
#define TL_LOG_LEVEL_LIB            2U      // it will not be
#define TL_LOG_LEVEL_ERROR          3U
#define TL_LOG_LEVEL_WARNING        4U
#define TL_LOG_LEVEL_INFO           5U
#define TL_LOG_LEVEL_DEBUG          6U
#define TL_LOG_LEVEL_MAX            TL_LOG_LEVEL_DEBUG

......

#define TL_LOG_LEVEL                TL_LOG_LEVEL_ERROR
```

Print module(即 TL_LOG_SEL_VAL):需要包含对应的 module 比如 TL_LOG_USER，该 module 才会打印。

```
typedef enum{
    TL_LOG_MESH          = 0,
    TL_LOG_PROVISION     = 1,
    TL_LOG_LOWPOWER      = 2,
    TL_LOG_FRIEND        = 3,
    TL_LOG_PROXY         = 4,
    TL_LOG_GATT_PROVISION    = 5,
    TL_LOG_WIN32         =6,
    TL_LOG_GATEWAY       = 7,
    TL_LOG_KEY_BIND      = 8,
    TL_LOG_NODE_SDK      = 9,
    TL_LOG_NODE_BASIC    = 10,
    TL_LOG_REMOTE_PROV   = 11,
    TL_LOG_CMD_RSP       ,
    TL_LOG_COMMON        ,
    TL_LOG_CMD_NAME      ,
    TL_LOG_NODE_SDK_NW_UT    ,
    TL_LOG_IV_UPDATE     ,
    TL_LOG_USER          ,    // never use in library.
    TL_LOG_MAX,
} ? end printf_module_enum ? printf_module_enum;
```

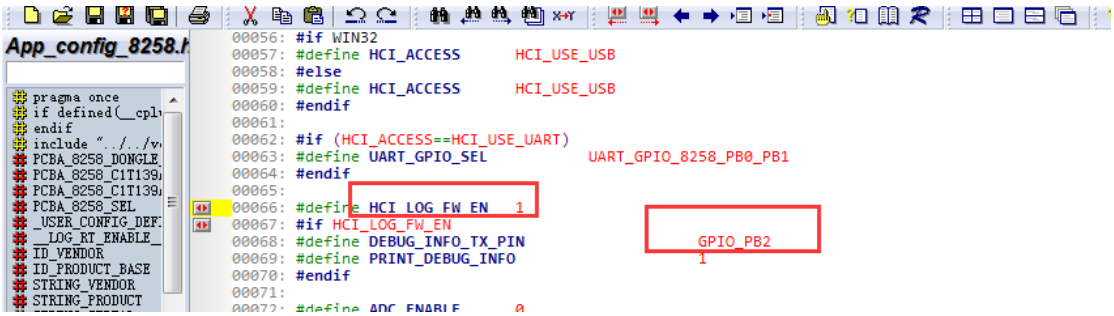
只有 print level 和 Print module 都符合条件的情况下，对应的 log 才会打印。

Log 打印设置步骤：

Step 1　设置　HCI_LOG_FW_EN　的值设为 1

PRINT_DEBUG_INFO 的功能是：使用 GPIO 口模拟 UART 来输出 log，该模式仅支持 TX 不支持 RX，专门用于输出 log.

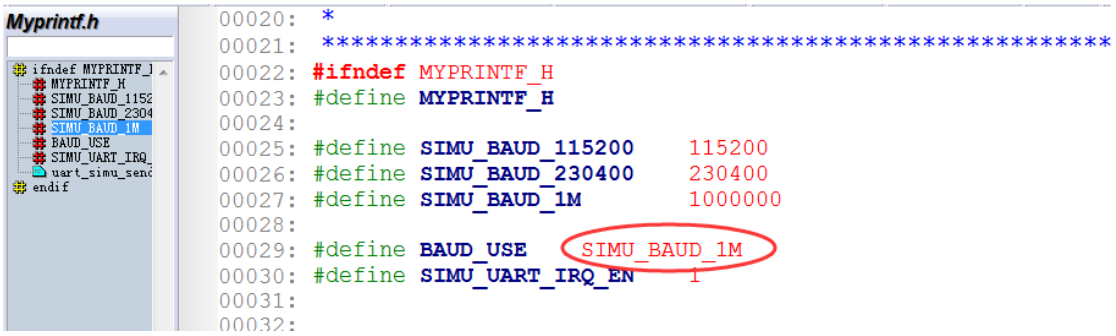
Step 2　设置 print pin



Step 3　设置 baud rate. 默认是 1Mbps.

因为在 IO 模拟 UART 输出的时候，是关闭中断了的(SIMU_UART_IRQ_EN 等于 1)，所以，在 IO 模拟 UART 的模式下，log 速度越快越好。所以不建议降低 baud rate

注意：可能有一些 USB 转串口的板子在 1M 速率下会有严重的误报，可以适当降低 uart 速率(但是速度降低会影响 log 的打印速度)，或者更换 USB 转串口工具，比如型号为 CH340G。



Step 4　选择 log module

在当前的分级 log 定义中，除了本章节开头提到的 print level 外，还有 log module 的设置。要输出 log，需要满足 print level 以及 log module 设定才会输出。

Demo SDK 为了使 log 更简洁，TL_LOG_SEL_VAL 我们默认只打开 TL_LOG_USER，其他 log 都是关闭的。

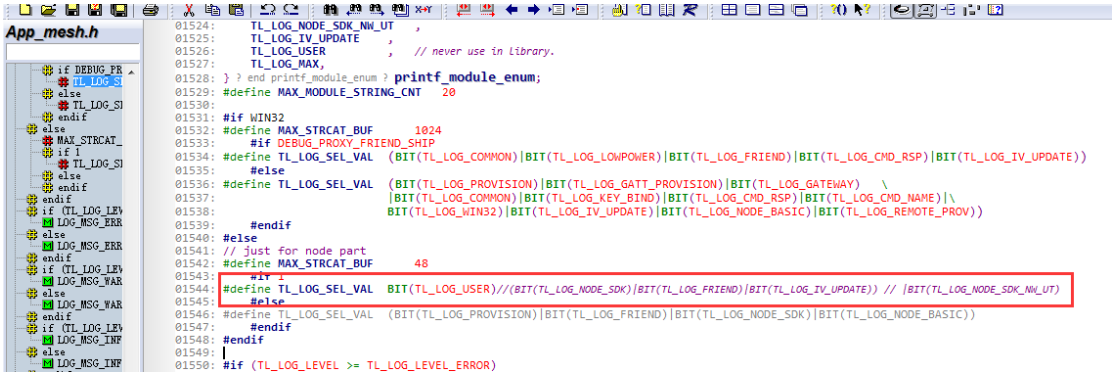
TL_LOG_USER 默认是没有任何地方调用的。用户需要增加打印的时候，使用如下方式即可：

LOG_USER_MSG_INFO(pbuf, len, format,...)，其中，

pbuf：需要把某个 buffer 转换成字符打印出来时使用。如无，请写 0.

len：pbuf 的长度。

如需要使用其他 API，比如 LOG_MSG_INFO，请先确认 TL_LOG_LEVEL 是否满足(默认是不满足的)。



Step 5　如果需要打开我们预设的一些调试 log，设置 TL_LOG_SEL_VAL 的值即可，比如：

```
#define TL_LOG_SEL_VAL (BIT(TL_LOG_USER)|BIT(TL_LOG_PROVISION)|BIT(TL_LOG_FRIEND)|BIT(TL_LOG_NODE_SDK)|BIT(TL_LOG_NODE_BASIC))
```

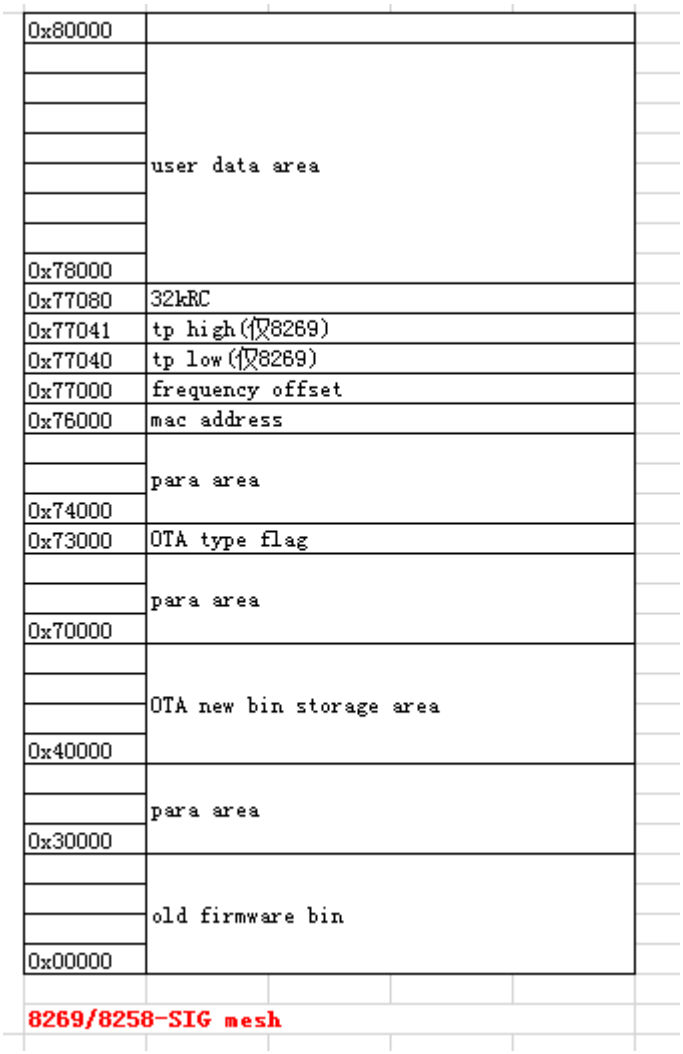
2. MCU 基础模块

本文主要介绍和 mesh 相关的部分，详细部分请参考《AN_19011501-C2_Telink Kite BLE SDK Developer Handbook》的"MCU 基础模块"章节。

2.1 Flash and RAM map

2.1.1 Flash map 介绍(以 512K flash 为例)

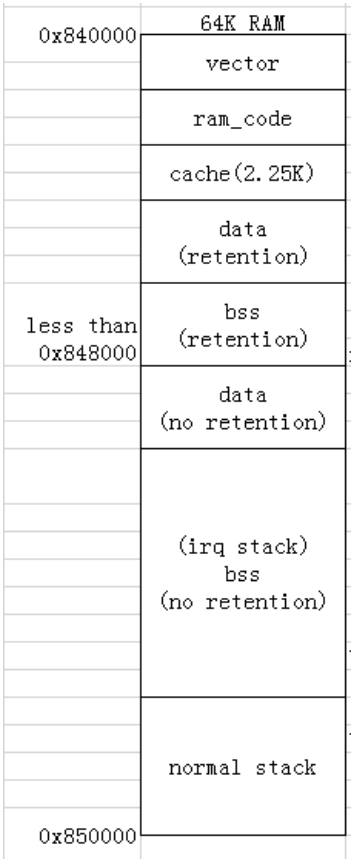
图 2-1 Flash Map



2.1.2 RAM map(以 8258 64K 为例)

对应的配置文档，详见 ./boot.link

图 2-2 RAM Map



- data(retention):包含有_attribute_data_retention_前缀的变量，以及所有初始化值不等于 0 的全局变量,默认为 retention data 属性；
- bss(retention):包含有_attribute_bss_retention_前缀的变量，以及所有初始化值 等于 0 的全局变量,默认为 retention bss 属性；
- data(no retention):有 _attribute_no_retention_data_ 前缀的全局变量
- bss(no retention):有 _attribute_no_retention_bss_ 前缀的全局变量。并且包含 irq_stack,其 size 由 IRQ_STK_SIZE 定义,默认 0x300，demo SDK 使用量小于 0x200，
- normal stack:bss 之后，64KRAM 之前都属于 normal stack。初始化值为 0xffffffff，目前为了加快 RAM 初始化速度，只初始化 3K 的 RAM。

注意:_attribute_bss_retention_和_attribute_no_retention_bss_这两个前缀，对于初始化不为 0 的变量不能使用。否则初始化值无效，默认为 0。

2.1.3 堆栈(stack) 溢出的判断

首先，要确保，bss(no retention)的末尾地址要小于 RAM 的末尾地址，也就是要留有空间给 normal stack。

End of bss(no retention) 可以通过 tdebug Tool 对变量按地址进行排序，最后一个变量之后的地址，就是末尾地址。

另外也可以通过 编译生成的 *.lst 文档计算:

然后，把所有的功能都测试一遍，然后查看 normal stack 和 irq_stack 是否有溢出。

Normal stack:初始化值为 0xffffffff。demo SDK 需要 stack 2K 左右，因为目前使用的 stack 都小于 3K，所以为了加快 RAM 初始化速度，只初始化 3K 的 RAM。通过读取 RAM 查看，如果发现 61K 位置即 0x84F400--0x84F403 这 4 个 byte 不是 0xFFFFFFFF，则表示堆栈使用已经超过 3K，如果超过，请联系我们做进一步的分析确认。

irq_stack:初始化值为 0x00000000，通过 tdebug 查看 irq_stk[]变量，观察使用情况；如果发现 irq_stk[0--3] 这 4 个 byte 非 0，则表示已经溢出。

2.2 时钟模块

MCU 的 clock 由 CLOCK_SYS_CLOCK_HZ 定义，825x 默认是 16MHz，8269 默认是 32MHz

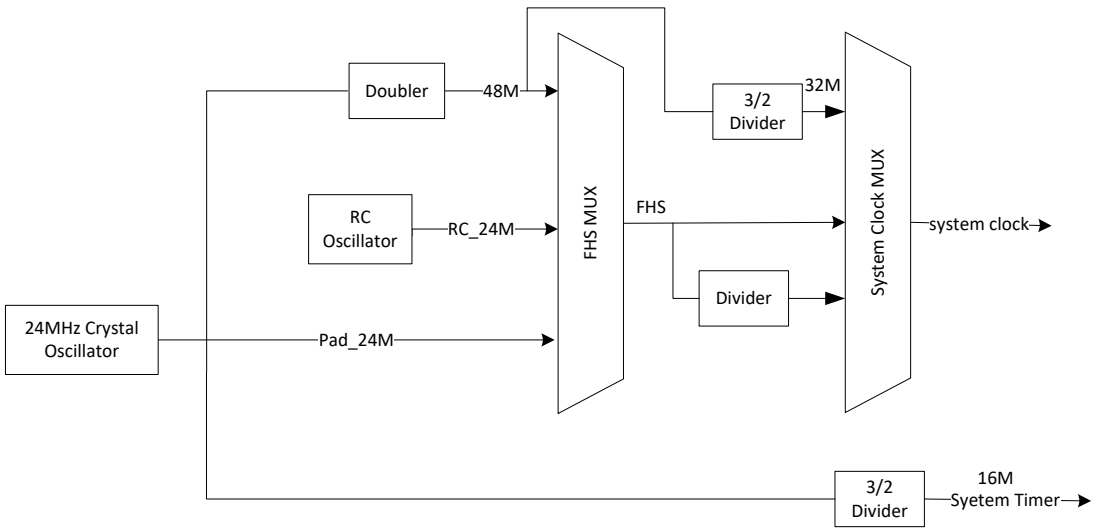
2.2.1 System clock & System Timer

系统时钟（system clock）是 MCU 执行程序的时钟。

系统定时器（System Timer）是一个只读的定时器，为 BLE 的时序控制提供时间基准，同时也可以提供给用户使用。

在 Telink 上一代 IC（826x 系列）上，System Timer 的时钟来源于 system clock，而 8x5x IC 上，System Timer 和 system clock 是独立分开的。如下图所示，System Timer 是由外部 24M Crystal Oscillator 经 3/2 分频得到的 16M。

图 2-3 system clock & System Timer



图上可以看到，system clock 可以由外部 24M 晶体振荡器经"doubler"电路倍频到 48M 后再分频得到 16M、24M、32M、48M 等，这一类 clock 我们称为 crystal clock（如 16M crystal system clock、24M crystal system clock）；也可以由 IC 内部 24M RC Oscillitor 处理后得到 24M RC clock、32M RC clock、48M RC clock 等。这一类我们称为 RC clock（BLE SDK 不支持 RC clock）。

在 BLE SDK 中，我们推荐使用 crystal clock。

初始化时调用下面的 API 配置 system clock，在枚举变量 SYS_CLK_TYPEDEF 定义中选择时钟对应的 clock 即可。

```
void clock_init(SYS_CLK_TYPEDEF SYS_CLK)
```

由于 8x5x System Timer 与 system clock 不一样，用户需要了解 MCU 上各硬件模块的 clock 是来源于 system clock 还是 System Timer。我们以 system clock 为 24M crystal 的情况来进行说明，此时 system clock 为 24M，而 System Timer 是 16M。

在文件 app_config.h 中 system clock 以及对应的 S、mS、uS 的定义如下:

```
#define CLOCK_SYS_CLOCK_HZ    24000000
enum{
    CLOCK_SYS_CLOCK_1S = CLOCK_SYS_CLOCK_HZ,
    CLOCK_SYS_CLOCK_1MS = (CLOCK_SYS_CLOCK_1S / 1000),
    CLOCK_SYS_CLOCK_1US = (CLOCK_SYS_CLOCK_1S / 1000000),
};
```

所有时钟源为 system clock 的硬件模块，在设置模块的 clock 时，只能使用上面 CLOCK_SYS_CLOCK_HZ、CLOCK_SYS_CLOCK_1S 等；换言之，如果用户看到模块中 clock 的设置使用的是以上几个定义，说明该模块的时钟源为 system clock。

如 PWM 驱动中 PWM 周期和占空的设置如下，说明 PWM 的时钟源是 system clock。

```
pwm_set_cycle_and_duty(PWM0_ID, (u16) (1000 * CLOCK_SYS_CLOCK_1US),  (u16) (500 * CLOCK_SYS_CLOCK_1US) );
```

System Timer 是固定的 16M，所以对于这个 timer，SDK code 中使用如下的数值来表示 S、mS 和 uS。

```
//system timer clock source is constant 16M, never change
enum{
    CLOCK_16M_SYS_TIMER_CLK_1S =  16000000,
    CLOCK_16M_SYS_TIMER_CLK_1MS = 16000,
    CLOCK_16M_SYS_TIMER_CLK_1US = 16,
};
```

SDK 中以下几个 API 都是跟 System Timer 相关的一些操作，所以涉及到这些 API 操作时，都使用上面类似"CLOCK_16M_SYS_TIMER_CLK_xxx"的方式来表示时间。

```
void sleep_us (unsigned long microsec);
unsigned int clock_time(void);
int clock_time_exceed(unsigned int ref, unsigned int span_us);
#define ClockTime                   clock_time
#define WaitUs                      sleep_us
#define WaitMs(t)                   sleep_us((t)*1000)
```

由于 System Timer 是 BLE 计时的基准，SDK 中所有 BLE 时间相关的参数和变量，在涉及到时间的表达时，都是用"CLOCK_16M_SYS_TIMER_CLK_xxx"的方式。

2.2.2 System Timer 的使用

Main 函数中 cpu_wakeup_init 初始化完成后后，System Timer 就开始工作，用户可以读取 System Timer 计数器的值（简称 System Timer tick）。

System Timer tick 每一个时钟周期加一，其长度为 32bit，即每 1/16 us 加 1，最小值 0x00000000，最大值 0xffffffff。System Timer 刚启动的时候，tick 值为 0，到最大值 0xffffffff 需要的时间为:(1/16) us * (2^32) 约等于 268 S，每过 268 S System Timer tick 转一圈。

MCU 在运行程序过程中 system tick 不会停止。

System Timer tick 的读取可以通过 clock_time()函数获得:

```
    u32 current_tick = clock_time();
```

该 BLE SDK 整个 BLE 时序都是基于 System Timer tick 设计的，程序中也大量使用这个 System Timer tick 来完成各种计时和超时判断，强烈推荐 user 使用这个 System Timer tick 来实现一些简单的定时和超时判断。

比如要实现一个简单的软件定时。软件定时器的实现基于查询机制，由于是通过查询来实现，不能保证非常好的实时性和准确性，一般用于对误差要求不是特别苛刻的应用。实现方法:

1) 启动计时:设置一个 u32 的变量，读取并记录当前 System Timer tick。

```
u32 start_tick = clock_time();    // clock_time()返回 System Timer tick 值。
```

2) 在程序的某处不断查询当前 System Timer tick 和 start_tick 的差值是否超过需要定时的时间值。若超过，认为定时器触发，执行相应的操作，并根据实际的需求清除计时器或启动新一轮的定时。

假设需要定时的时间为 100 ms，查询计时是否到达的写法为:

```
if( (u32) ( clock_time() - start_tick) >   100 * CLOCK_16M_SYS_TIMER_CLK_1MS)
```

由于将差值转化为 u32 型，解决了 system clock tick 从 0xffffffff 到 0 这个极限情况。

实际上 SDK 中为了解决系统时钟不同导致和 u32 转换的问题，提供了统一的调用函数，不管系统时钟多少，都可以下面函数进行查询判断:

```
if( clock_time_exceed(start_tick, 100 * 1000))  //第二个参数单位为 us
```

需要注意的是:由于 16M 时钟转一圈为 268 S，这个查询函数只适用于 268 S 以内的定时。如果超过 268 S，需要在软件上加计数器累计实现（这里不介绍）。

应用举例:A 条件触发（只会触发一次）的 2 S 后，程序进行 B()操作。

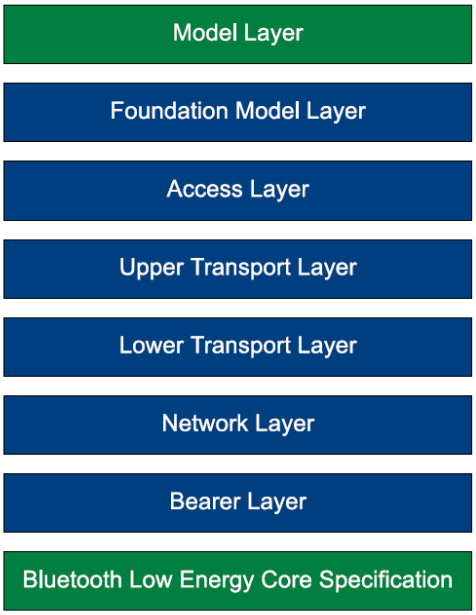
```
u32   a_trig_tick;
int   a_trig_flg = 0;
while(1)
{
 if(A){
       a_trig_tick = clock_time();
       a_trig_flg = 1;
 }
 if(a_trig_flg &&clock_time_exceed(a_trig_tick,2 *1000 * 1000)){
       a_trig_flg = 0;
       B();
 }
}
```

3. MESH spec 中的常用概念介绍

以下介绍的顺序，按《MshPRFv1.0.1.pdf》的标题顺序进行，本文主要做大概的介绍，详细的介绍，请参考 SIG MESH SPEC 的相关章节。

3.1 Layered architecture

图 3-1 Layered Architecture



3.1.1 Model layer

model 定义了一个节点支持的功能，每一个 model 都定义了自己的 op code，以及 status。比如 generic onoff model，定义了 Generic ON/OFF/GET/STATUS。

Provisioner 在组网的时候，会通过 get composition data 命令去获取节点支持的所有 model id，然后 provisioner 就能知道节点具体支持什么功能了。只有这个节点支持了对应的 model 之后，才应该给它发送该 model 定义的 op code。

Model 又分为 server model 和 client model。server model:简单的说，他是一个被控制的角色，有自己的状态，可以被别的节点改变和获取，比如 onoff server model，可以接收 onoff set/ get 命令，可以回复 onoff status 命令，但是不能发送 onoff set/get 命令，也不会处理 onoff status 命令。

client model:是一个控制 server 节点的角色，没有自己的状态。比如 onoff client model，可以发送 onoff set/ get 命令，可以处理收到的 onoff status 命令，但是不能发送 onoff status 命令，也不会处理 onoff set/get 命令。

3.1.2 Foundation Model layer

Foundation Model 的模式和 model 基本一样，是基础 model，包含 Configuration Server model，Configuration Client model，Health Server model，Health Client model。

对于被配网节点都必须包含 Configuration Server model， provisioner 节点必须包含 Configuration Client model。这两个 model 包含的常用 op code 是 subscription add/delete(即组号添加/删除)等，并且这两个 model 的 access layer 层的加密都使用 device key，所以一般来说只有 provisioner 节点才能发送 configuration model 的 set/get 命令。

3.1.3 access layer

把 op code 和 parameter 按规定的格式组合在一起。

3.1.4 transport layer

使用 app key 或者 device key(configuration model 使用)进行加解密。判断并确认是否需要执行 分包和组包协议。

目前为了兼容 BLE4.2 等不支持长广播包的设备，所以都统一设定 adv 的最大 payload 为 31byte。

3.1.5 Network layer

对于发送流程:主要包含，对数据包添加 sequence number，等，并使用 network key，iv index 对数据进行加密。发送完成后 sequence number 会执行"加 1"的操作。

对于接收流程:主要包含，使用 network key，iv index 对数据进行解密，解密后判断 sequence number 是否有效(即是否大于已经接收过的值)，如果无效则直接丢弃。

3.1.6 Bearer layer

把已经执行过加密的数据包通过 type 为 LL_TYPE_ADV_NONCONN_IND(0x02) 的广播包发送到 mesh 网络中。

3.2 Architectural concepts

3.2.1 States

一个节点的状态，比如 onoff States，lightness States

3.2.2 Bound states

两个相互关联的 states，比如 onoff 和 lightness。比如，当 onoff 的值由 1 变为 0 的时候，lightness 的值也会变成 0；当 onoff 的值由 0 变为 1 的时候，lightness 的值也会由 0 变为关灯之前的 lightness 的值.同理，当 lightness 的值在 0 和非 0 之间变化时，onoff 的值也会由 0 变为 1。

3.2.3 Messages

就是把加密完成后，发送到 mesh 网络中的 packet，我们也叫做 mesh packet、mesh command。

3.2.4 Node & Elements

Node 表示一个完整的节点或者蓝牙模块，Element 表示 Node 里面的某个操作元素。

Node address 有且只有一个，element address 可以是一个或者多个，当 element address 是多个的情况，这些 address 都会是连续的。第一个 element address 又叫做 primary address，Node address 的值和 primary address 相同。

有多个 element 的原因是，当一个 node 有多个相同的 states 的时候，就需要用到了。比如一个插座产品，插座上有 3 个插孔，要使用 Generic ONOFF 命令控制插孔的 onoff 状态，如果只有一个 address 的话，当节点收到命令后，是没办法确定要控制哪一个插孔的，所以为了能指定控制某一个插孔，就需要使用多个 element address。

另外，色温灯(CT Light)的 element 个数是 2 个，虽然色温灯只需要一个 onoff states，但是他需要两个 generic level model，一个是和 lightness 对应，另一个是和 Temp 对应，所以就需要 2 个 element。

同理，HSL 灯(RGB 灯)的 element 个数是 3，因为 HSL 灯需要 3 个 generic level model，分别和 lightness、Hue 以及 Sat 对应。

在组网的时候，Node 会在 provision flow 的交互信息里面上报 element 的个数，比如 2，provisioner 会分配一个地址给 Node，比如是 0x0002。Node 收到后，会按顺序分配 0x0002 给 element 1, 0x0003 给 element 2。Provisioner 在对下一个节点进行组网的时候，会从 0x0004 开始分配。

3.2.5 Models

参考 3.3.1 “Model layer”的介绍。

3.2.6 Publish & subscribe

- **Publish**:publish 就是 Element 主动发送 status 的过程，可以通过 Config Model Publication Set 命令配置 publish address，以及设置周期 publish 参数。当配置了 publish address 后，只要状态发生变化，Node 都会自动执行 publish status 的动作。是否需要周期发送，就要看周期 publish 的参数。
- **Subscribe**:Subscribe (即订阅)说的是:节点接收到别的节点 publish 出来的 status message(比如 generic onoff status)，或者 control message(比如 generic onoff)后，根据 model 的 Subscribe list[]来判断是否处理该 message。Subscribe list[]里面是 group address 或者 virtual address，不能是 unicast address，也不能是 0xffff。可以通过 Config Model Subscription Add，Config Model Subscription Virtual Address Add 等命令添加。

  判断是否接收并处理的依据是:

1) 当收到 message 的 destination address 为非 unicast address，再判断是否能在对应 model 的 Subscribe list 的 address 里面匹配到。
2) 当 destination address 为 unicast address，则直接判断和自身的 element address 是否匹配。
3) 当 destination address 为 0xffff，则表示符合判断条件。

3.2.7 Security

加解密时需要使用的要素包含:Network key，IV index，App key 或者 device key。

一个 message 需要进行两层加密，分别是使用 app key 或者 device key 对整个 access layer(包含 op code，parameters)进行加密，以及使用 network key + iv index 对 network layer 进行加密，network layer 就是发送到 mesh 网络中的 packet，包含 source address、destination address 和 sequence number 等。

对 access layer 加密的时候，如果 op code 对应的 model 是 config model，则使用 device key，否则使用 app key。

对于需要分包的 message，因为 access layer 的 payload 在加密的时候是整个 payload 进行加密的，所以需要接收到所有的 packet 后，才能完成解密和校验。

我们的 SDK 默认支持 2 个 network key (NET_KEY_MAX)和两个 app key(APP_KEY_MAX)。

多个 network key 可用于管理多个 network。

多个 app key 可以用于管理不同安全级别的产品。比如 mesh 网络里面既有灯，也有门锁，如果需要门锁的安全级别更高的话，只有某些人能控制，此时可以通过单独给门锁分配一个独立的 app key，并且保证这个 app key 只有某些手机 app(provisioner)知道，在进行网络分享的时候，也不会分享出去。这样就可以提高安全级别。

3.2.8 存储 Sequence Number 的处理

前面"3.1.5 Network layer"章节有提到，mesh message 的 Sequence Number(SNO)每发完一次命令都会进行累加，接收端在进行 SNO 判断的时候，要求大于已经收到过的值，如果小于等于则认为是无效的 message。这个就要求发送端每发送一个命令后，都要把 SNO 存储到 flash 中，这个存储频率太高了，特别是在需要分包发送的时候。所以我们做了一个处理:SNO 每增加 MESH_CMD_SNO_SAVE_DELTA(默认 0x80)，才存储一次，此时，为了能保证在断电并重新上电后的 SNO 大于已经使用过的值，在上电初始化时，把读取到的 SNO 加上 MESH_CMD_SNO_SAVE_DELTA。具体实现部分，请参考 mesh_flash_save_check()和 mesh_misc_retrieve()关于 MESH_CMD_SNO_SAVE_DELTA 的处理。

3.2.9 Friendship

Friend ship 是指 Friend Node(FN)和 Low Power Node(LPN)之间通过规定的 establish friend ship flow 建立的关系。在 friend ship 建立成功后，LPN 在 sleep 期间，当有节点发命令给 LPN 时，FN 会先把这个 message 先保存下来。当 LPN 唤醒后，会发送 POLL 的查询命令给 FN，此时 FN 就会把保存的 message 发送给 LPN。这样就能达到较低的功耗，缺点是需要网络中有 friend node 的存在，以及命令的接收和响应有一定的延迟。

更详细的部分请参考 spec 以及后续 LPN 的章节。

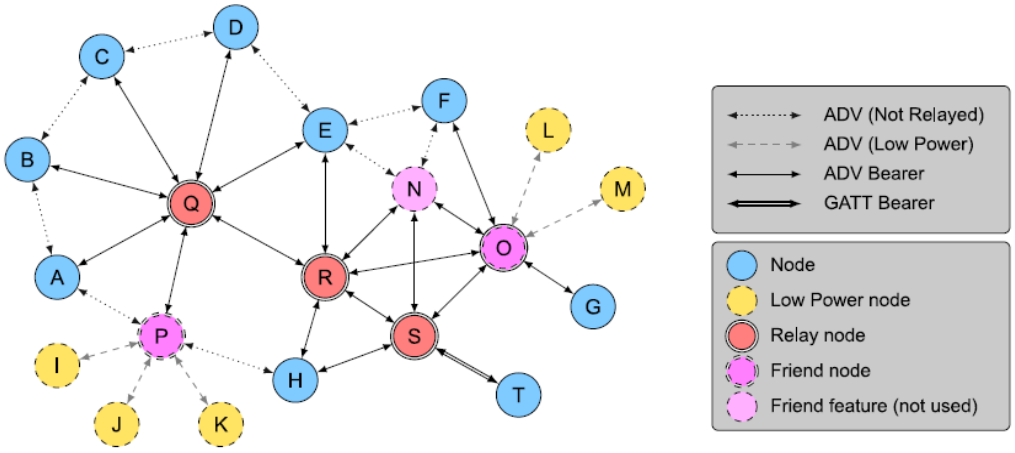
3.2.10 Features

Mesh features 主要有:

- ✧ Relay feature ---- 节点收到有效的 network message 后，会把 message 的 TTL 值减 1，然后 relay 发送出去，如果收到的 TTL 的值小于等于 1，则不进行 relay。这样可以使 mesh 网络传输距离更远。引入 TTL，主要是控制最后收到该 message 的节点的 delay 时间在可控范围内。我们 SDK 的 TTL 的值默认是 TTL_DEFAULT(0x0A)，可以修改这个宏，provisioner 也可以通过 Config Default TTL Set 配置，最大值是 127。
- ✧ Proxy feature ---- proxy 是用于手机 APP 接入 mesh 网络的协议。在 mesh 网络中，APP 也是一个独立的节点，有自己的 Node address。引入 proxy，是因为目前大部分手机不能完全自定义发送广播包，以及不能一直处于监听 mesh 网络的状态(中间要切换到 WiFi 等)，所以手机 APP 需要通过 BLE GATT 连接一个节点，直连节点收到 APP 发过来的数据后会转发出去，当直连节点收到 mesh 网络中回复给 app 的 message 后，会先通过 GATT 按 proxy 协议回复给手机 APP。
- ✧ APP 下发 message 给直连节点用的是 ATT_OP_WRITE_CMD(0x52)，直连节点回复 message 给 APP 用的是 notify，即 ATT_OP_HANDLE_VALUE_NOTI(0x1B)。
- ✧ Low Power feature ---- 参考 3.2.9 的"Friendship"的介绍。
- ✧ Friend feature ----参考 3.2.9 的"Friendship"的介绍。

3.2.11 Mesh Topology

图 3-2 Mesh 网络拓扑图示例



我们 SDK，默认常供电节点都支持 Relay，Friend. 所有节点都支持通过 GATT proxy 和 ADV 组网。

3.3 Mesh networking

3.3.1 Network layer

Address

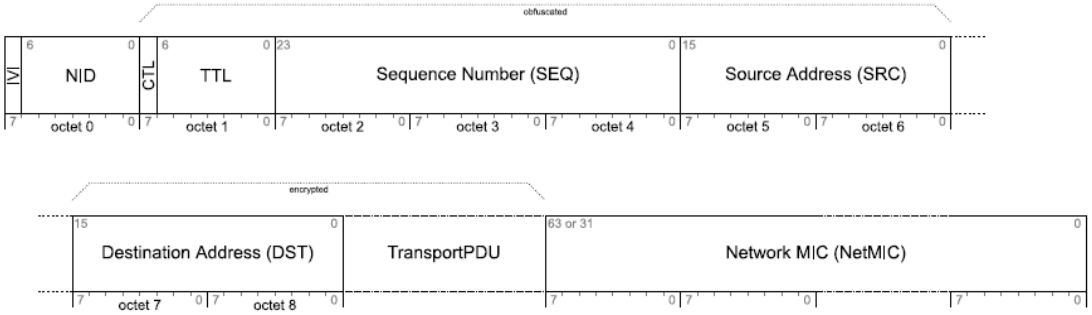
表格 3-1 16 bit 地址分配

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxx (excluding 0b0000000000000000)	Unicast Address
0b10xxxxxxxxxxxxx	Virtual Address
0b11xxxxxxxxxxxxx	Group Address

- ✧ Unassigned address:0 表示 Unassigned address
- ✧ Unicast address:用于表示 element address
- ✧ Group address:组号地址，用于组控以及 publish----subscribe 机制。
- ✧ Virtual address:需要结合 16BYTE 的 lable UUID 使用，Virtual address 是这个 UUID 经过 hash 算法后生成的值。当 group address(共 16384 个)不够使用的时候，可以进行扩展。目前的应用暂时用不到。

Network PDU

图 3-3 Network PDU 格式



表格 3-2 Network PDU Field Definitions

Field Name	Bits	Notes
IVI	1	Least significant bit of IV Index
NID	7	Value derived from the NetKey used to identify the Encryption Key and Privacy Key used to secure this PDU
CTL	1	Network Control
TTL	7	Time To Live
SEQ	24	Sequence Number
SRC	16	Source Address
DST	16	Destination Address
TransportPDU	8 to 128	Transport Protocol Data Unit
NetMIC	32 or 64	Message Integrity Check for Network

- IVI: iv index(即 SDK 的 iv_idx_st.tx[3]这个 byte 的最低 bit，目前 SDK 的 iv index 按 big endian 存储)。
- NID:和 network key 相关
- CTL:标记是否是 control message。

Network transmit count/interval (重传次数和重传间隔的定义)

network transmit count 是指发送一个命令，需要重传的次数，这些重传的 rf packet 是一模一样的，包括 sno 等。重传的目的是为了提升接收成功率。假如当网络中只有两个 mesh 节点，只发送一次的成功率是 80%，即丢包率是 20%，那么，理论上，发送 6 次的丢包率是 20%的 6 次方，即 0.0064%，即收包成功率是 99.993%，当然这个是理论分析。和 RF 环境等还有一定的关系。

我们的 SDK 协议栈默认重发 5 次，即 TRANSMIT_CNT_DEF(5)，总共发发送次数是 n+1 = 6 个。

network transmit interval 是指重传的两个包之间的发送间隔，我们 SDK 默认在 30-40ms 之间，由 TRANSMIT_INVL_STEPS_DEF(2) 决定，计算方式是 ((TRANSMIT_INVL_STEPS_DEF + 1)*10 + (0----10))ms。”

network transmit count、transmit interval 还可以通过 SIG 定义的标准 config 命令 CFG_NW_TRANSMIT_SET 来配置。

综上所述，我们 SDK 默认发送一个 network packet，比如 generic ONOFF no ack 命令(该命令不需要分包)，需要的时间大概是 40 * 6 = 240ms。

Reliable retry(发包重试次数)

Reliable retry 是指应用层的重试，用于有 status 回复的命令，比如 generic ONOFF。当发送一个 network packet 后(包含 network transmit)，会判断是否收到 status，如果没有收到，我们会进行 retry，retry 的时候，network packet 里面的 sequence number 会变化。我们默认最多重试两次。

3.3.2 Access layer

表格 3-3 Access Payload Field

Field Name	Size (octets)	Notes
Opcode	1, 2, or 3	Operation Code
Parameters	0 to 379	Application Parameters

表格 3-4 Opcode Format

Opcode Format	Notes
0xxxxxxx (excluding 01111111)	1-octet Opcodes
01111111	Reserved for Future Use
10xxxxxx xxxxxxxx	2-octet Opcodes
11xxxxxx zzzzzzzz	3-octet Opcodes

op code 有三种类型，1byte，2byte 和 3byte。1byte，2byte 是 SIG 定义的命令。3byte 是 vendor 自定义的命令，其中 2 个 byte 是 vendor ID（CID）,整个 mesh 网络中，一个 vendor id 最多支持 64 个 vendor opcode。

Access layer 包含 op code 和 parameter，最大支持 380 byte。

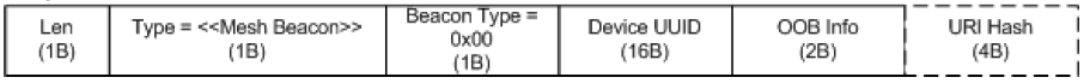
3.3.3 transport layer

目前为了兼容 BLE4.2 等不支持长广播包的设备，所以都统一设定 adv 的最大 payload 为 31byte。去掉一些数据包的通讯协议需要占用的部分，单包的有效 payload 是 11byte，所以当 Access layer 超过 11byte 后，就需要分包，所以对于 vendor op code 来说，当 parameters 大于 8 个 byte(8=11-3)，mesh 协议栈就会自动执行分包发送。用户不需要介入。

3.3.4 mesh beacon

下图是 unprovisioned device beacon 的 PDU。

Unprovisioned Device Beacon



Device UUID 可以唯一识别 node。因为有些手机，比如 IOS 不能获取 mac，以及在未来的 remote provision 中无法获得 mac,所以在 SIG mesh 中是通过 Device UUID 来唯一识别 node，而不是通过 mac。

unprovisioned beacon 通过 non-connectable ADV 的 packet 来发送，用于 PB-ADV provision 模式。

Oob info 及 URI Hash 请参考 spec 《3.9.2 Unprovisioned Device beacon》

在未组网时，会发送 unprovision beacon，通过 unprov_beacon_send()来发送。发送周期由"beacon_send.inter = MAX_BEACON_SEND_INTERVAL"来定义，默认是 2 秒。

在组网后，会发送 security beacon，通过 mesh_tx_sec_nw_beacon()来发送。另外还可以通过 CFG_BEACON_SET 命令打开或者关闭这个发送使能开关。请参考本文章节《4.4 控制对应的节点》"“SecNwBc”按钮"的操作。发送周期由 SEC_NW_BC_INV_DEF_100MS 来定义，默认是 10 秒。

3.3.5 iv update folw

即 iv index 的 update flow。network layer 和 access layer 的加解密过程都需要用到 iv index。前面提到，mesh 网络要求 network PDU 的 sequence number 要一直累加，而 sequence number 是 3byte 表示。当使用很长一段时间后，sequence number 接近最大值的时候，就需要考虑更新 iv index，否则 sequence number 就会归 0，导致接收端认为是一个无效的 message。从某种程度上，可以理解 iv index 为 sequence number 的扩展位。

iv index update flow 是节点自行发起，自行 update 的过程。目前 SDK，当节点检测自己的 sequence number 超过了 IV_UPDATE_START_SNO (0xC00000) 后，会主动发起 iv update folw。每次 Iv update 后，iv index 会加 1.

详细的 iv update flow 请查阅 spe 相关章节: SPEC V1.0《3.10.5 IV Update procedure》。

3.3.6 heartbeat

mesh 的心跳包，可以配置周期发送，所以可以用于做在线离线检测(周期 publish 机制也可以做在线离线检测)，以及 hops 的计算，即计算 heartbeat message 经过多少跳之后，才被接收到。

经过统计一定时间内收到的 heartbeat 的个数(count)，并计算得到每个 heartbeat 的 hops 的 值，得到 min hops 和 max hops，进而了解整个网络的布局，以及每一个节点的 message 传输的可靠程度。不过每次配置 heartbeat subscription 只能监听和统计一个节点的状态。

hops 计算方式是 ：

hops = InitTTL - RxTTL +1。

- ✧ InitTTL:是 heartbeat publish set 里面的 TTL 参数。
- ✧ RxTTL:是收到的 message 的 network PDU 里面的 TTL。

节点默认不发送 heartbeat，具体配置方式，详见"5.6 Heartbeat 的演示"章节

3.3.7 Health

Health model 相关的 message 是用来反映节点的 warning 或者 error 状态。比如电量的 warning 和 error 指示等，详见 spec 4.2.15.1 Current Fault，如下表:

表格 3-5 Health Mode Message

Value	Description
0x00	No Fault
0x01	Battery Low Warning
0x02	Battery Low Error

Value	Description
0x03	Supply Voltage Too Low Warning
0x04	Supply Voltage Too Low Error
0x05	Supply Voltage Too High Warning
0x06	Supply Voltage Too High Error
0x07	Power Supply Interrupted Warning
0x08	Power Supply Interrupted Error
0x09	No Load Warning
0x0A	No Load Error
0x0B	Overload Warning
0x0C	Overload Error
0x0D	Overheat Warning
0x0E	Overheat Error
0x0F	Condensation Warning
0x10	Condensation Error
0x11	Vibration Warning
0x12	Vibration Error
0x13	Configuration Warning
0x14	Configuration Error
0x15	Element Not Calibrated Warning
0x16	Element Not Calibrated Error
0x17	Memory Warning
0x18	Memory Error
0x19	Self-Test Warning
0x1A	Self-Test Error
0x1B	Input Too Low Warning
0x1C	Input Too Low Error
0x1D	Input Too High Warning
0x1E	Input Too High Error
0x1F	Input No Change Warning
0x20	Input No Change Error
0x21	Actuator Blocked Warning
0x22	Actuator Blocked Error
0x23	Housing Opened Warning
0x24	Housing Opened Error
0x25	Tamper Warning
0x26	Tamper Error
0x27	Device Moved Warning
0x28	Device Moved Error
0x29	Device Dropped Warning

Value	Description
0x2A	Device Dropped Error
0x2B	Overflow Warning
0x2C	Overflow Error
0x2D	Empty Warning
0x2E	Empty Error
0x2F	Internal Bus Warning
0x30	Internal Bus Error
0x31	Mechanism Jammed Warning
0x32	Mechanism Jammed Error
0x33–0x7F	Reserved for Future Use
0x80–0xFF	Vendor Specific Warning / Error

4. 调试工具操作说明

4.1 下载固件

关于 BDT 工具的详细操作说明，请参考 “Help”→“User guide”的说明文档，以下仅介绍使用到的操作。

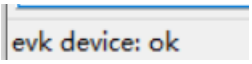
在操作之前，需要先下载“8258_mesh.bin”到每一个节点中（8258 Dongle），然后烧录 provisioner 节点，provisioner 目前有两个模式，即 GATT 连接的 master dongle 模式 和 gateway 模式(ADV 模式)，：

GATT 模式:把“8269_mesh_master_dongle.bin”下载到 8269 的 Master Dongle (8269 Dongle)。

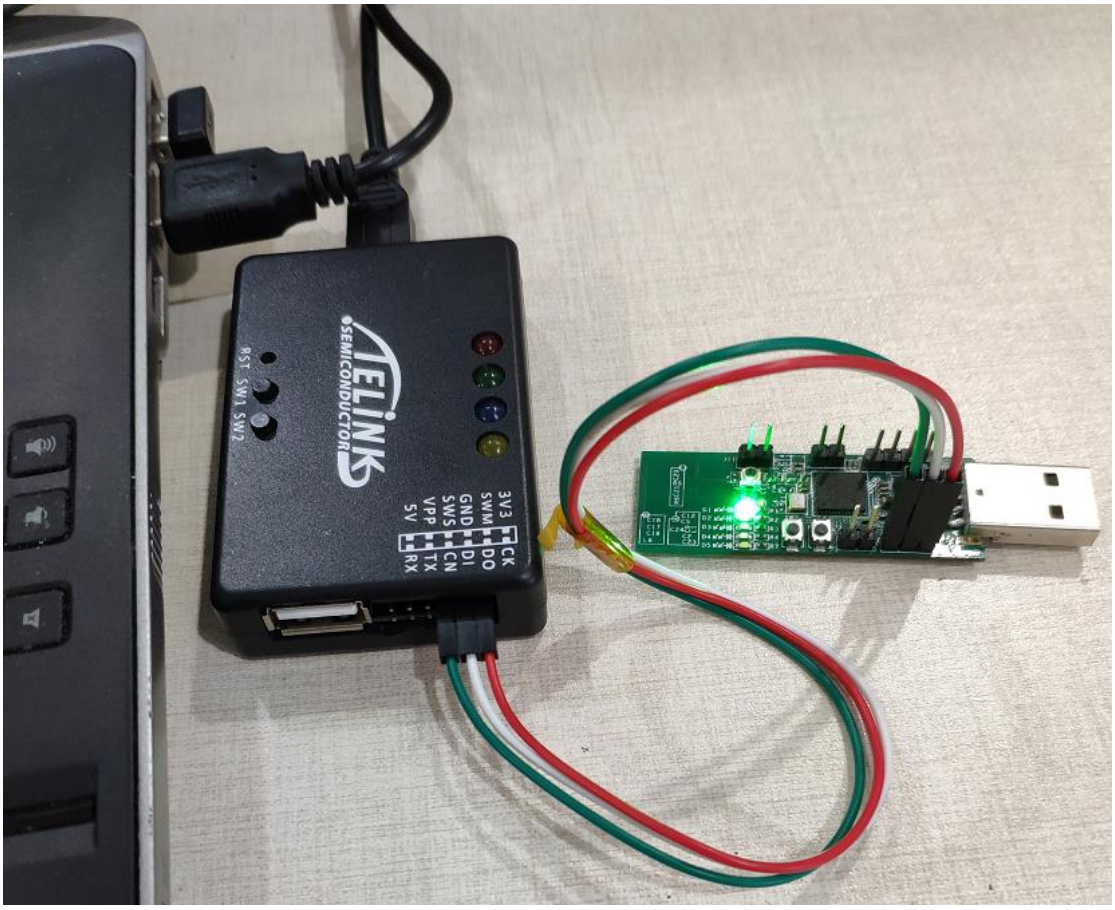
Gateway 模式:把” 8258_mesh_gw.bin”下载到 8258 的 gateway Dongle 中。

例如，用户可以按照以下步骤下载固件到 8258 light 节点。

1) 硬件连接：通过 USB 线连接 EVK 板的 miniUSB 接口和 PC 的 USB 口，如果 EVK 板子上面的指示灯闪烁一下，表示 EVK 板和 PC 是正常连接的。8258 Dongle 通过 USB 接口来连接 EVK 板的 USB 接口。

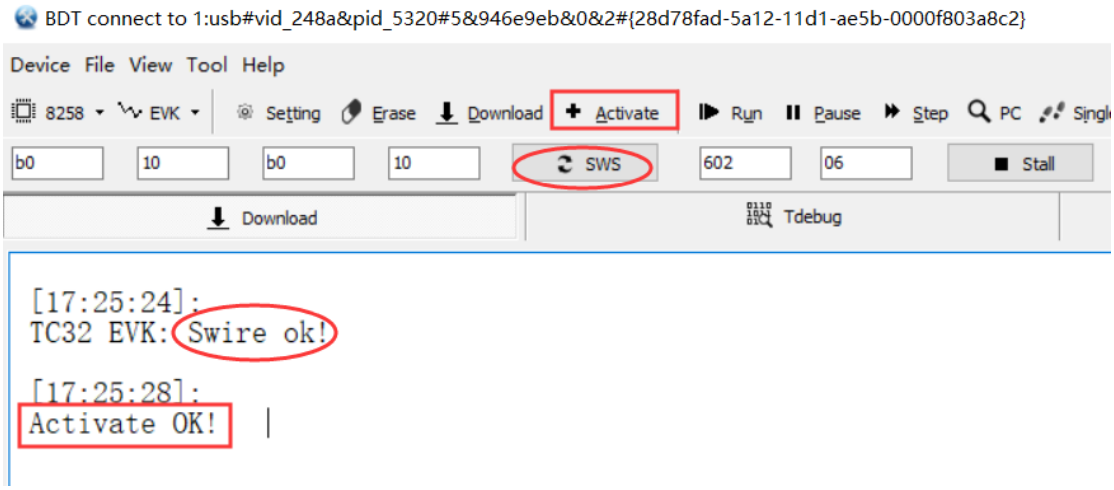
正常情况，工具左下角会显示: 

。



2) 通过 Telink BDT 工具将“8258_mesh.bin”下载到 8258 Dongle 的 flash。

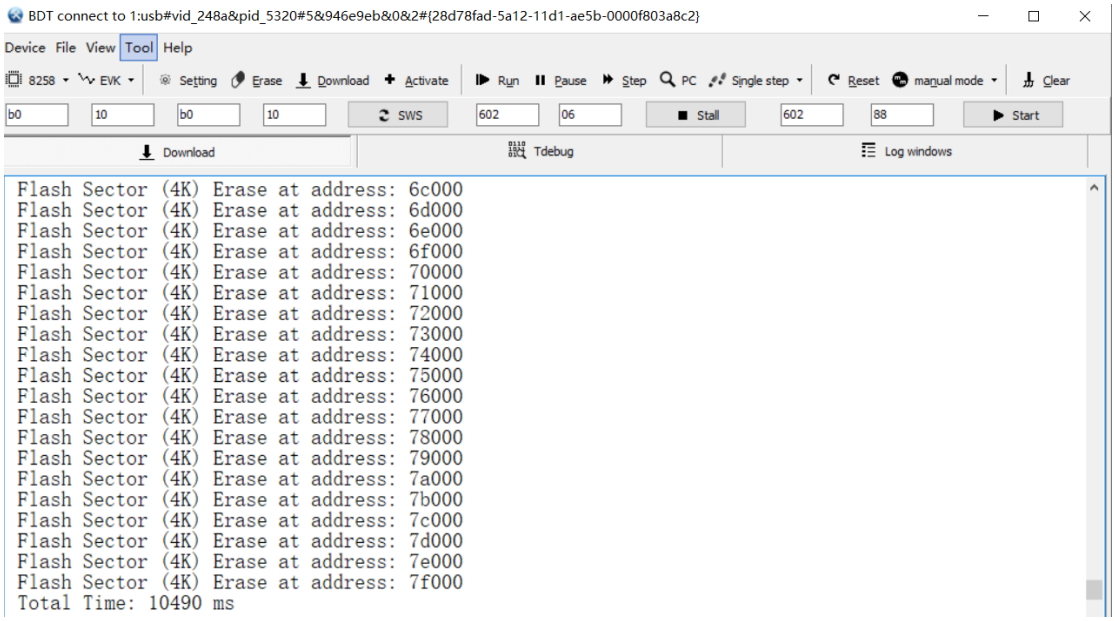
图 4-1 BDT 界面



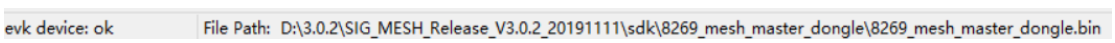
Step 1　打开 BDT，首先 8258 选择对应的型号，再点击 SWS 确认 EVK 和 8258 dongle 节点 是否可以正常通讯，如果正常，会看到"Swire ok"。如果不正常，则可能是芯片处于睡眠状态，需要点击 "Activate"按钮进行激活唤醒，特别是低功耗设备需要这个操作，激活成功会提示"Activate OK"。Activate 包含有 MCU 重启的操作。

Step 2　擦除 8258 Dongle 板的整个 flash 部分。点击 Erase 按钮。

注:erase 的起始地址和 size 可以在 Setting 进行配置。

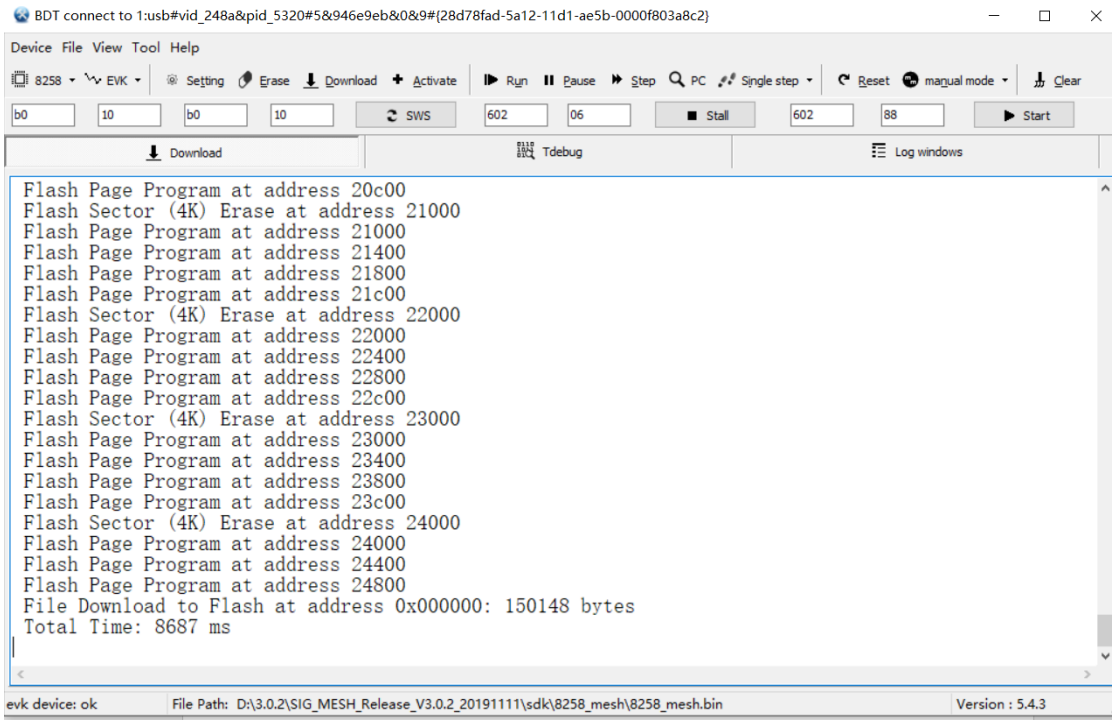


Step 3　点击"File"按钮在选择"open"选择对应的"8258_mesh.bin"文件，点击打开，然后 BDT 界面对应的文件：



Step 4　点击" Download "按钮，将选中的"8258_mesh.bin"烧写到 flash 从 0 开始的地址。

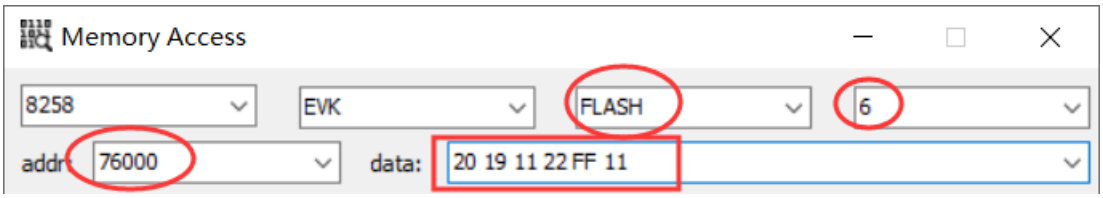
注:down load 起始地址和 size 可以在 Setting 这里配置，默认是 0。



固件烧写步骤

Step 1　点击工具栏“Tool”—“Memory Access”，弹出对话框：

Note:烧录 light 节点和 gateway 节点时需要此操作，GATT master dongle 不需要。



按以上参数输入 6 个 byte 的 MAC，输完后，在 data 控件栏敲下回车键执行写入操作。在 Addr 控件栏敲下 Tab 键，会执行 read 操作，进行回读确认。

如果没有输入 mac，8258 dongle 重新上电后，检测到 0x76000 没有 MAC，就会随机分配一个，并保存到 flash 0x76000 处。

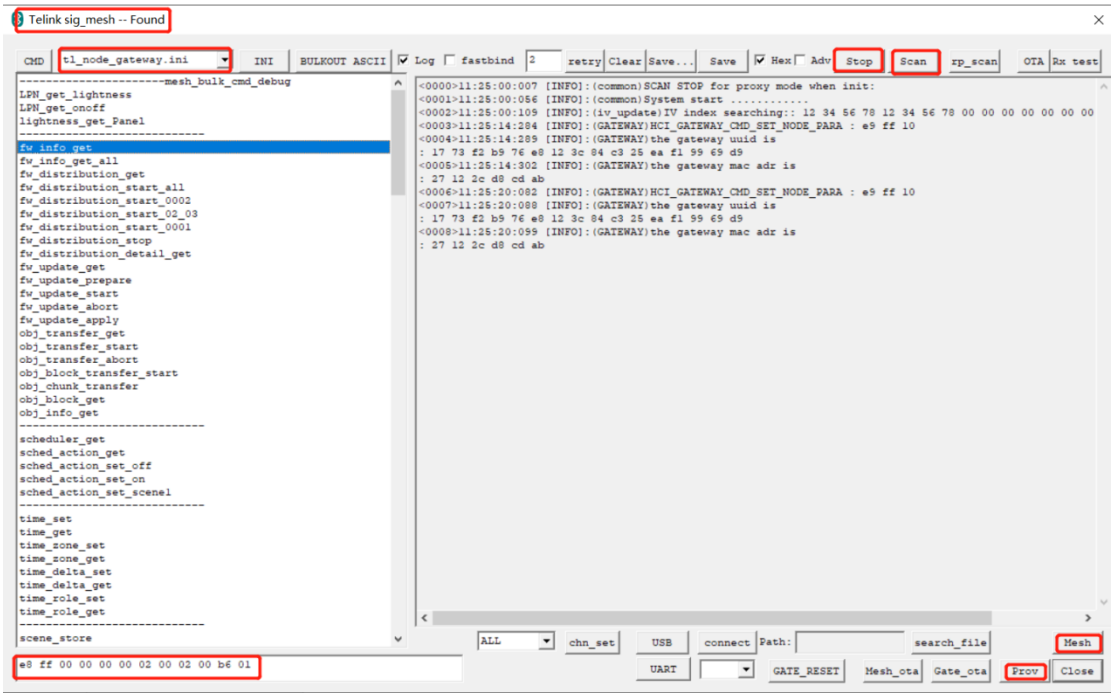
Step 2　重新上电后，8258 Dongle 板就能够作为一个灯节点。

以上是烧录的时候，BDT 使用到的操作步骤。用户也可以通过点击“Help”→“User guide”打开说明文档来获取其他更多的功能和操作方法。

4.2 Gateway USB 模式的 BLE 连接和加灯

1) 打开“sig_mesh_tool.exe”工具，将烧录好 8258_mesh_gw.bin 的 gateway dongle 插到 PC 的 USB 口中。
2) 如下图所示，工具左上角的“Found”表示 8258 gateway Dongle 和 PC 工具正常连接，并且可以正常通信。此时工具会根据接入的硬件自动选择“tl_node_gateway.ini”

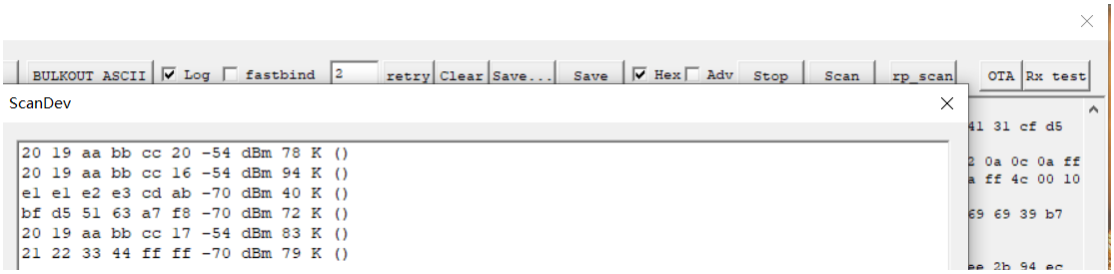
图 4-2 “SIG_MESH_TOOL”工具界面



3) 将 8258mesh 节点上电。

4) 点击右上角的“Scan”按钮，工具将打开一个“ScanDev”窗口，该窗口会显示对应的 MAC 地址列表，包含 rssi 和频偏。

图 4-3 “ScanDev”窗口



5) 双击“ScanDev”窗口中对应的条目，选择节点。

Gateway 模式:双击后，只是选择该节点，不需要任何连接动作和命令发出。

8258 gateway Dongle 模式:双击后会建立 BLE 连接，如果 8258 gateway Dongle 上的红灯亮起，表示 BLE 连接正常建立。“Stop”按钮用于终止当前的 BLE 连接，8258 gateway Dongle 上的白灯亮起，表示 BLE 连接断开。目前只支持单个节点的 BLE GATT 连接。

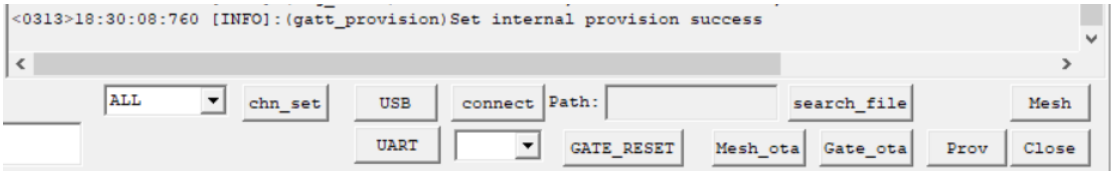
6) 点击右下角“Prov”按钮来打开“provision”窗口。

注意:在初始状态下，“Provision”按钮和“bind_all”按钮是被禁止的，并且用户不可同时操作这两个按钮。

“network_key”是第一次打开“provision”窗口的时候随机生成的，在点击“SetPro_internal”前可以修改。

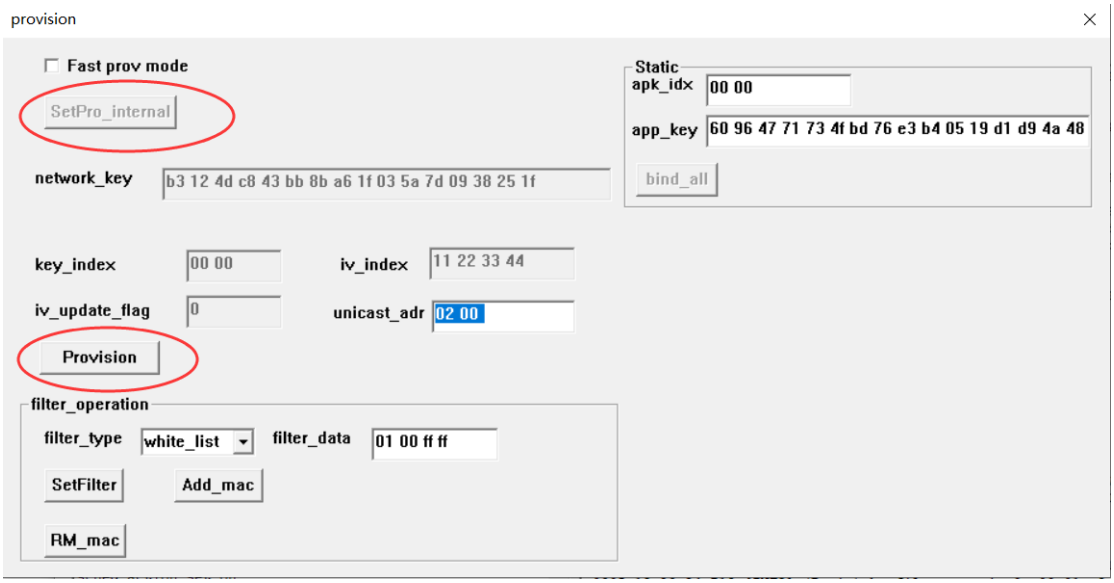
图 4-4 Provision 窗口

7) 点击“SetPro_internal”用来设置网络的初始参数，在 log 窗口中打印“Set internal provision success”来表示设置内部的网络参数成功。



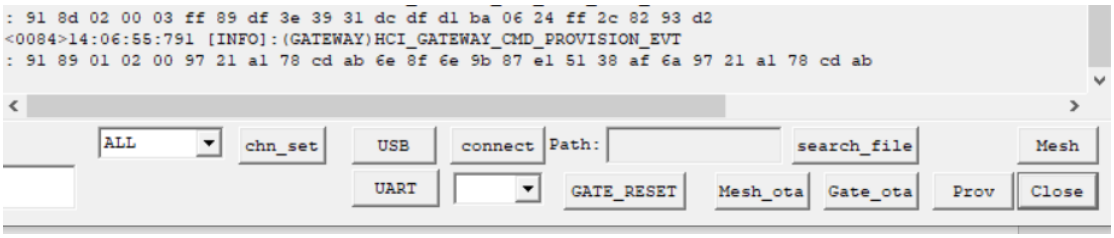
一旦点击 “SetPro_internal”，对应的 netkey 等参数就不能再更改，所以“SetPro_internal”会被灰化。参数被保存在 mesh_database.json 文件中，下次重新开启工具，会自动读取 mesh_database.json 里面的参数。此时如果还需要修改 network_key，需要解散整个网络，全部恢复到出厂设置。

此时“Provision”按钮使能。unicast_addr 是即将分配给待 provision 的节点的 primary address，客户可以手动修改，但是，如果没特别需求不建议修改。

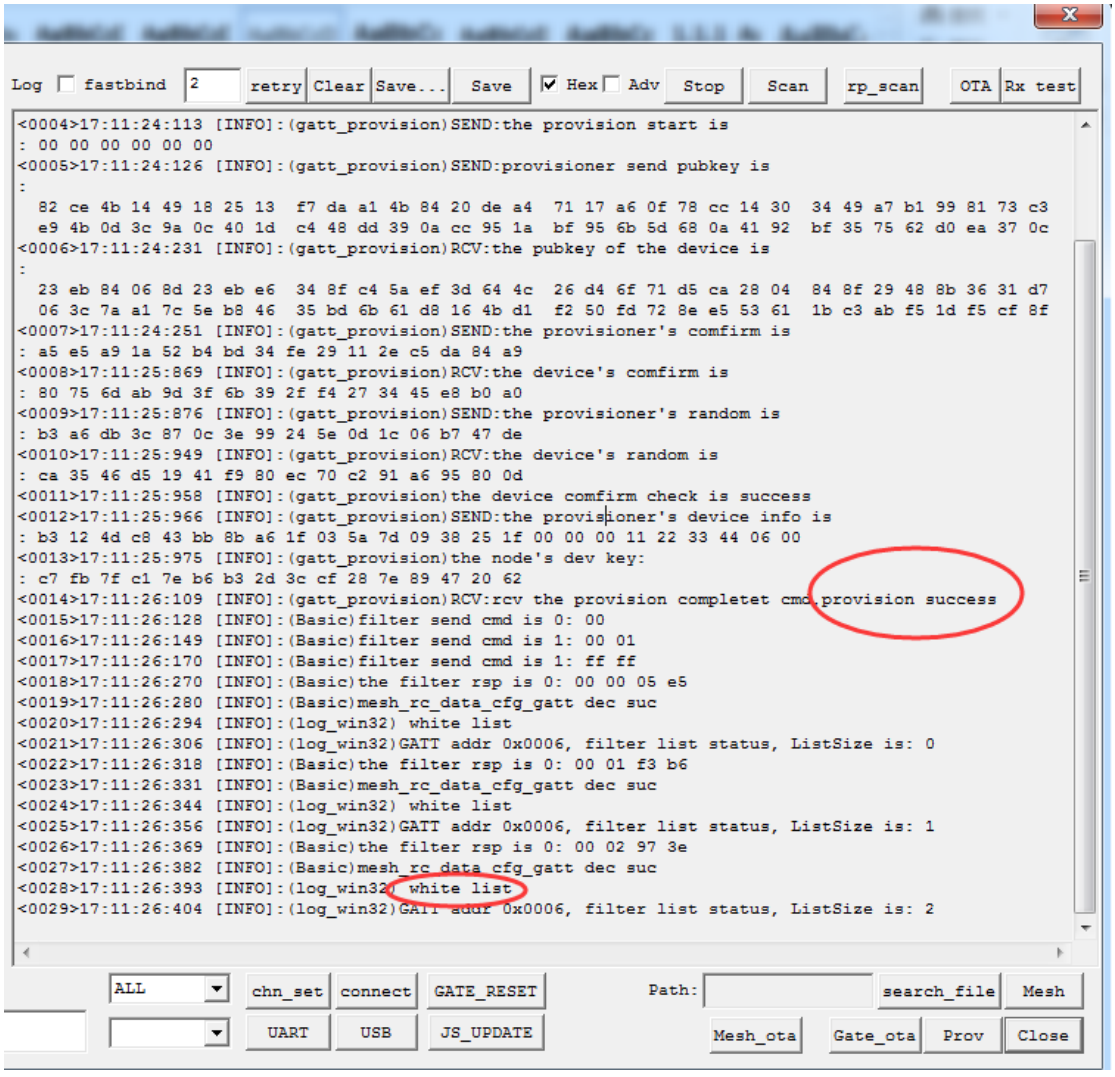


8) 点击"Provision"按钮来执行 SIG 的 provision flow，就能把对应的节点加入到网络中，连接的节点的红色 LED 会闪烁 4 下表示成功。log 信息显示如下图。

Gateway 模式的 log:

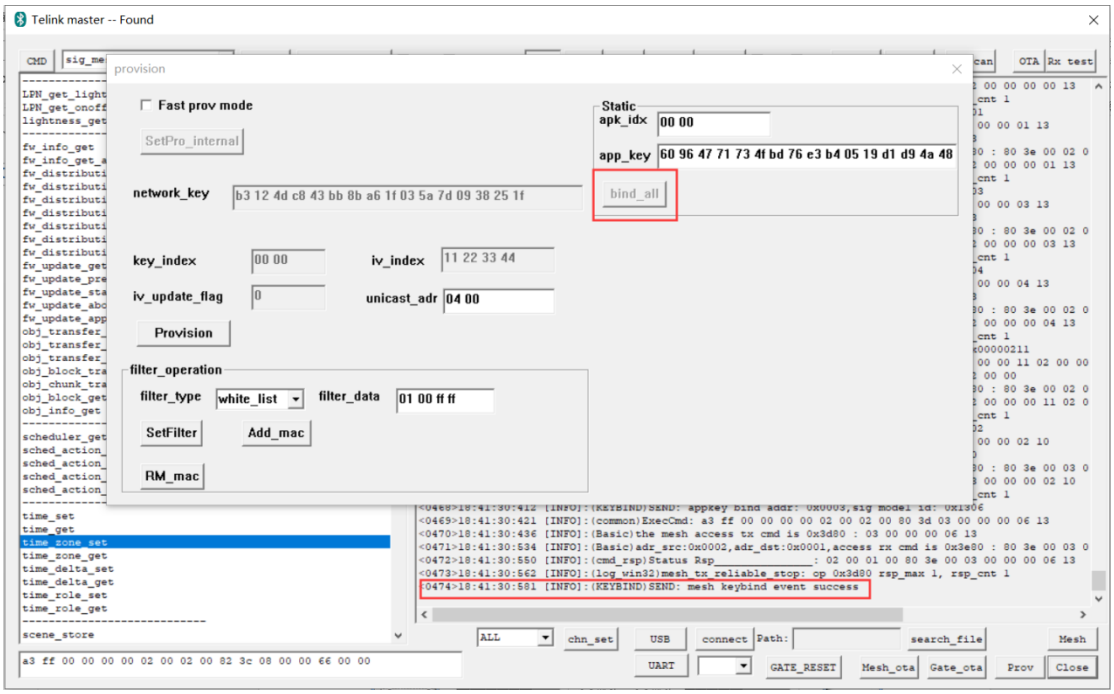


GATT Master dongle 模式的 log:

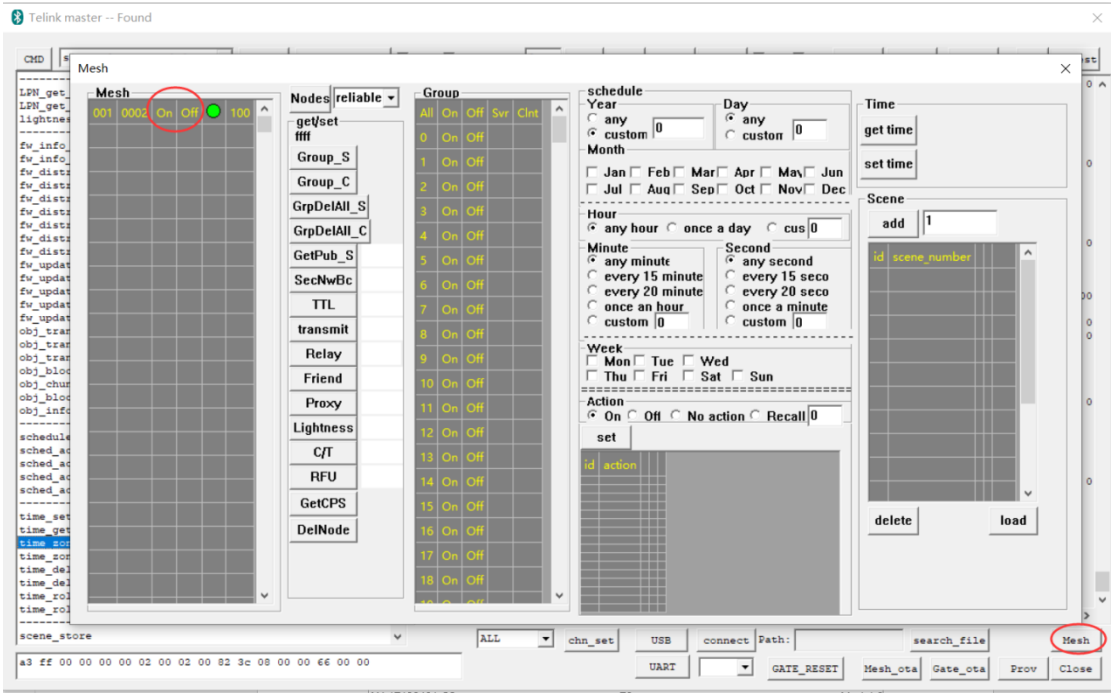


9) 设置好 app_key 后点击“bind_all”，会先发送 get composition data 命令，获取所有 model id，然后为所有 model 绑定 app_key。

Bind_all 成功后，“unicast_adr”会自动根据当前节点占用的 element 个数累加，表示对下一个节点进行 provision 所使用的 primary address。(比如 CT 灯的 element 个数是 2，则每加一个 CT 灯，“unicast_adr”控件的值就会加 2)



10) 绑定 App_key 后点击主界面 Mesh 按钮进去 mesh 界面可进行开关灯等操作。



11) 解散网络

GATT master dongle 模式和 Gateway 模式均可按如下方式操作：

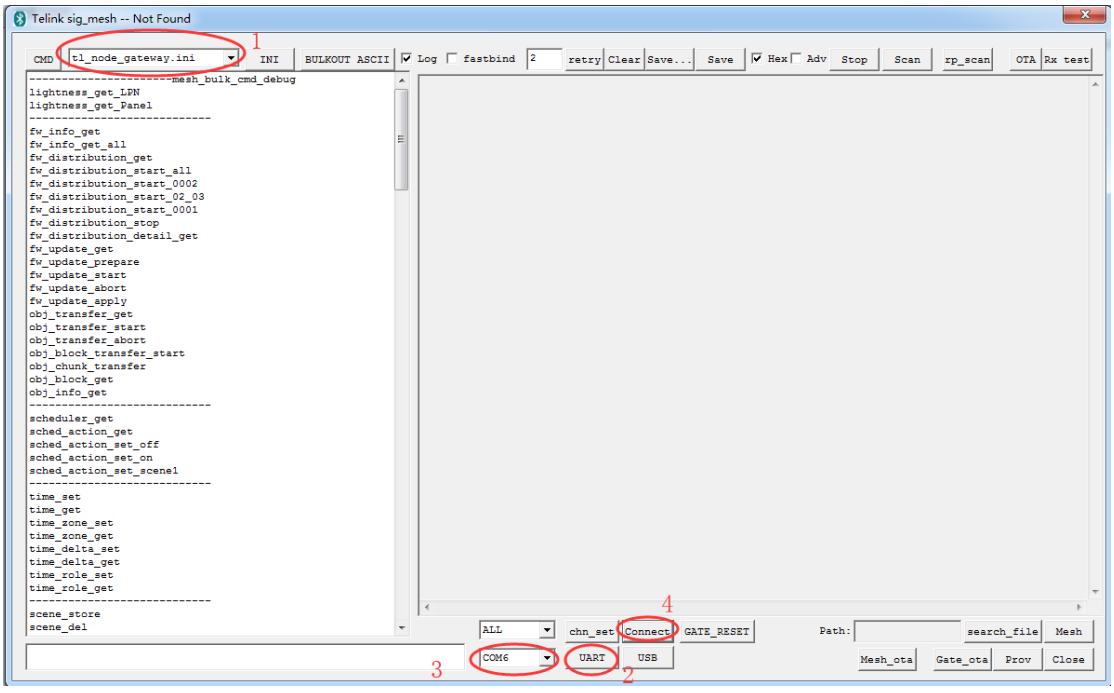
依次选择节点，然后按 “4.5.3 通过 UI 配置某一节点的详细参数”章节里面“DelNode”按钮的操作说明，点击“DelNode”踢掉该节点。

Note：GATT master dongle 模式需要先删除非 GATT 直连节点，最后再删除 GATT 直连节点，因为删除 GATT 直连节点会导致当前 GATT 连接断开。

4.3 Gateway UART 模式的 BLE 连接和加灯

主要是配置 UART 端口：

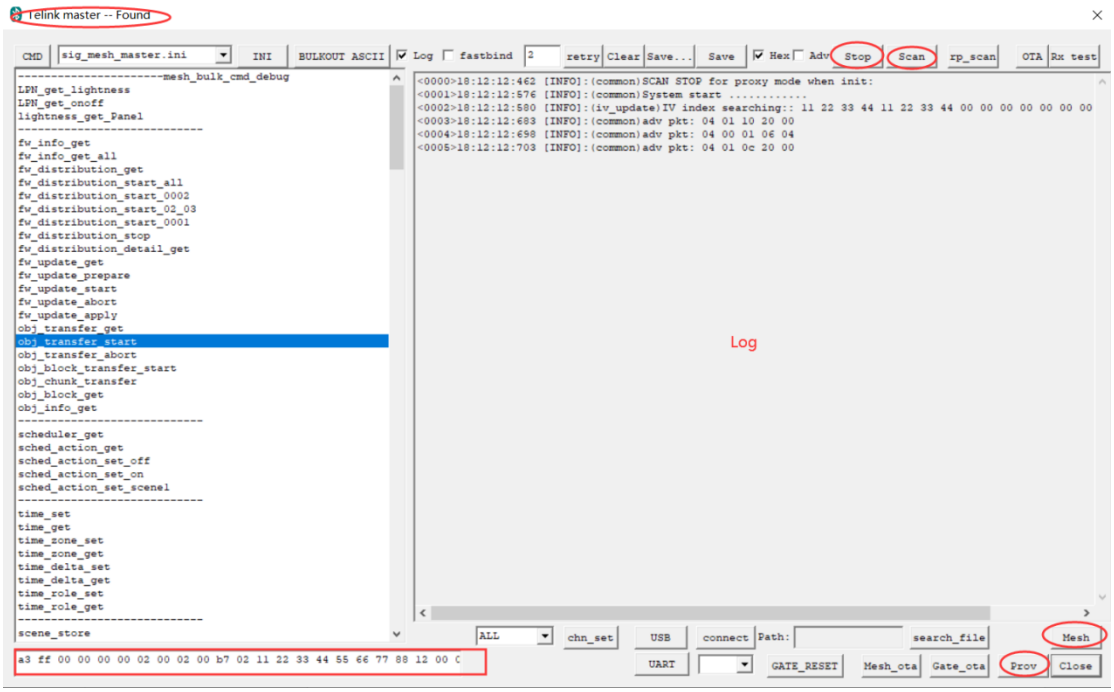
1) gateway firmware 的 HCI_ACCESS 选择 HCI_USE_UART，重新编译。
2) 插入串口工具到 PC，tx/rx 和 gateway 的 rx/tx 接通。
3) 打开“sig_mesh_tool.exe”。
4) 点击 UART 然后选择 显示出来的 COM 口。
5) 点击“Connect”按钮，如果连接成功，按钮会变成“Disconnect”。现在就可以通过 UART 来执行 gateway 的功能了。
6) 剩余所有操作和 USB 模式都一样。请参考“4.2 Gateway USB 模式的 BLE 连接和加灯”



4.4 GATT master dongle 模式的 BLE 连接和加灯

1) 打开“sig_mesh_tool.exe”工具，将烧录好程序的 8269 Master Dongle 插到 PC 的 USB 口中。
2) 如下图所示，工具左上角的“Found”表示 8269 Master Dongle 和 PC 工具正常连接，并且可以正常通信。此时工具会根据接入的硬件自动选择“sig_mesh_master.ini”

图 4-5 “SIG_MESH_TOOL”工具界面



3) 从第 3 步开始，请参考本章“gateway 模式的 BLE 连接和加灯”的介绍。

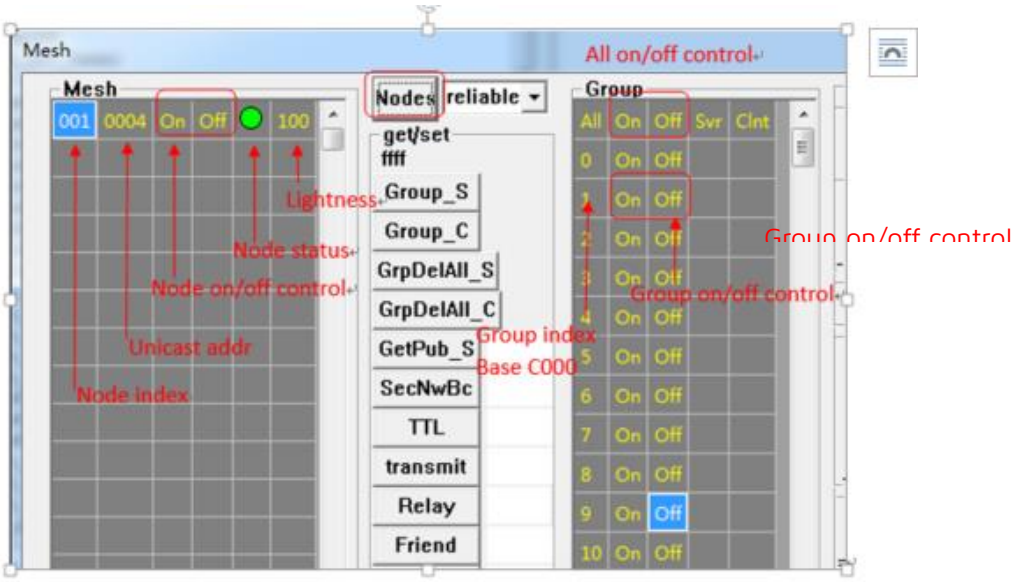
4.5 控制对应的节点

GATT master dongle 和 gateway 的操作方式和 UI 都相同。

4.5.1 UI 显示和单节点、所有节点的 onoff 控制

1) 点击主窗口右下角的“Mesh”按钮。会弹出一个“Mesh”窗口。

图 4-6 Mesh 窗口



2) 用户能够点击“Mesh”窗口中的“Nodes”来刷新所有的灯的状态。

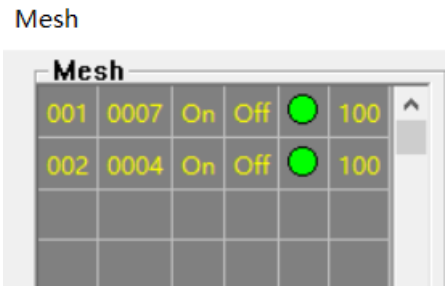
如果“Nodes”右边的下拉框选择 reliable 的时候，点击“Nodes”，是发送了 lightness get 命令。根据 lightness status 的值刷新 UI。

如果“Nodes”右边的下拉框选择 unreliable 的时候，点击“Nodes”，是发送了 lightness get noack 命令。但是后续发出的 onoff 等命令是 no ack，所以节点不回复 status，所以 UI 无法刷新，此时一般是结合 publish 机制来刷新 UI

如果“Nodes”右边的下拉框选择 online status 的时候，点击“Nodes”，是不会发送任何命令，只是初始化 UI 为空，然后根据返回的 online status 数据刷新 UI。Note:online status 模式是私有模式，需要节点的 firmware 打开 ONLINE_STATUS_EN 这个宏开关。

另外，lightness 显示的时候，是把 SIG 定义的 0-65536 的刻度转换成 0-100 的刻度之后再显示的。

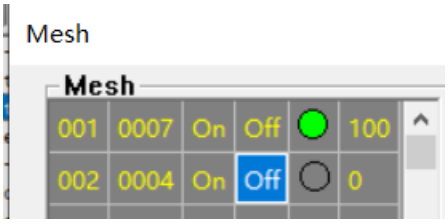
图 4-7 节点状态



3) 单个节点操作：点击“On”/“Off” 按钮（如图红框所示），对应的灯会控制开关的状态。对应的节点的状态会上报给工具来刷新对应的状态。

4) 表示状态“on”， 表示状态“off”， 表示“离线”。

图 4-8 单个节点的控制



5) 全开全关的控制：点击在“All”标记旁的“On”/“Off”，MESH 网络内的所有节点都会开关。

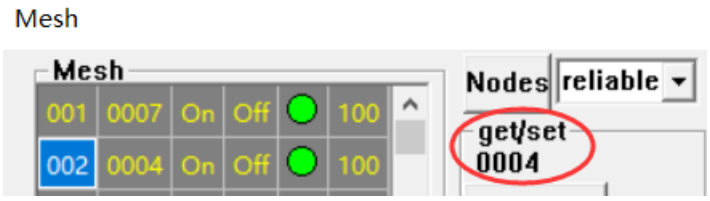
图 4-9 所有的节点的控制

4.5.2 分组控制(即 subscription 的功能演示)

点击对应灯节点的索引号，如 002，获取该灯的 node address 并显示在下图的位置。

此处默认是 0xffff，表示未选择任何一个节点。

图 4-10 获取节点地址



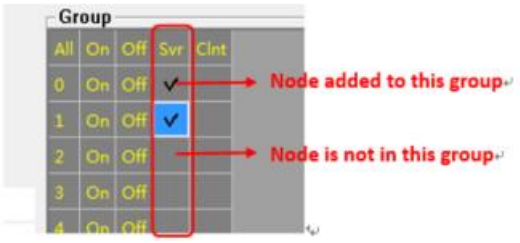
用户可以左键/右键点击 Group 控件中对应 group 的"Svr"框(在 Off 的右边)， 将选中的该灯节点添加到对应的组/从对应组中删除。"Svr"框中显示 √，表示节点已加入组中；"Svr"框中显示空白，表示节点已不在组中。

- ✧ "Svr"这一列对应操作的是 generic onoff server model (0x1000)
- ✧ "Clnt"这一列对应操作的是 generic onoff client model (0x1001)

因为一般情况下，节点只支持 server model，所以，我们一般只操作 "Svr"这一列。

group index 和组地址的转换关系是:组地址 = group index + 0xC000.

图 4-11 分配一个灯到多个组



用户可以点击 Group 控件中对应 group 的"On"/"Off"来控制整个组的开关。

图 4-12 组的控制

4.5.3 通过 UI 配置某一节点的详细参数

双击下图 1 的位置，选中该节点，选中后，会先自动发送 get group(subscription list)命令，获取该节点的 group 并显示在对应的 UI. 然后接着判断当前节点是否支持 scene、time 和 scheduler 功能。如果支持，会自动发送 get 命令获取 scene，time 和 scheduler 列表。

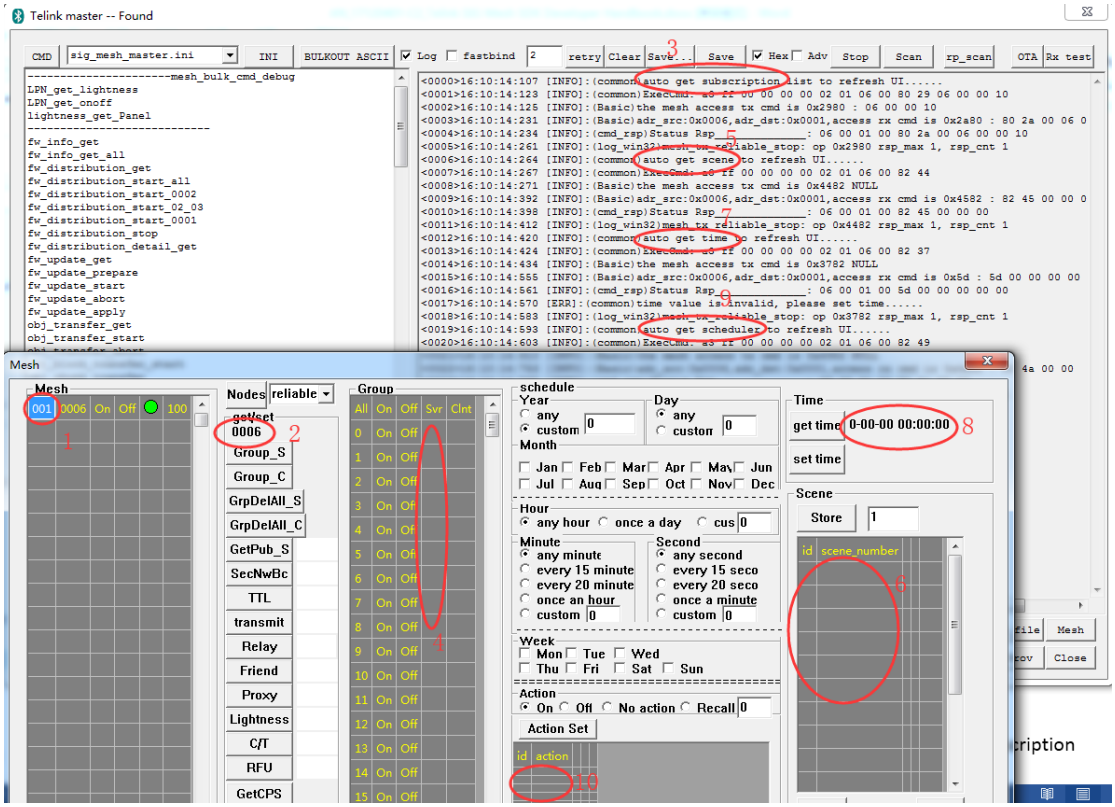
获取 scene 的时候，发送 SCENE_REG_GET 命令获取所有的有效场景序号，并显示在 UI 列表中。

获取 time 的时候，发送 TIME_GET 命令获取当前节点的 time，并显示在 UI 中。如果节点刚上电，time 是 0，表示等待被配置时间，这种情况下，时钟会一直保持 0，不会计时。配置时间的方式有两种。第一，通过 app/gateway 发送 time set 命令；第二，配置节点的 time modle 的 publish 属性，刚上电的节点获取别的节点 publish 出来的 time status。

获取 scheduler 的时候，会先发送 SCHD_GET 命令获取所有有效的 scheduler index，根据返回的 index 的值，分别再发送 SCHD_ACTION_GET 获取详细的参数，并显示在 UI 列表中。下

图所示是节点刚 provision 完成，还没有添加任何 scheduler，所以不需要发送 SCHD_ACTION_GET。

另外，因为节点是刚 provision 完成，所以 group，scene、time 和 scheduler 都是空的。



以下的按钮介绍都是基于当前选择的节点。

“Group_S”按钮

点击该按钮会发送 CFG_SIG_MODEL_SUB_GET 命令，获取当前选中节点的 onoff server model 的 subscription address list 即组号列表，并在“Svr”这一列中显示出来。

“Group_C”按钮

点击该按钮会发送 CFG_SIG_MODEL_SUB_GET 命令，获取当前选中节点的 onoff client model 的 subscription address list 即组号列表，并在“Clnt”这一列中显示出来。

大部分的节点不支持 onoff client model，所以这个按钮一般不用。

“GrpDelAll_S”按钮

点击该按钮会发送 CFG_MODEL_SUB_DEL_ALL 命令，删除当前选中节点的 onoff server model 的 subscription address list，并把“Svr”这一列中清空。

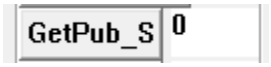
“GrpDelAll_C”按钮

点击该按钮会发送 CFG_MODEL_SUB_DEL_ALL 命令，删除当前选中节点的 onoff client model 的 subscription address list，并把“Clnt”这一列中清空。

“GetPub_S”按钮

点击该按钮会发送 CFG_MODEL_PUB_GET 命令，操作的 model 是 onoff server model，并把返回的值在后边的显示框中显示。功能是获取节点的 publish address。

在输入框中，修改 publish address，然后按键盘的“Enter”键，会发送 CFG_MODEL_PUB_SET，对应的 publish address 是输入框里面的值



此处配置 publish 参数的时候，除了 publish address 外，其他参数都是默认值。详见发送命令的 log:

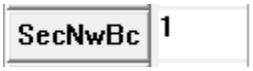
```
ExecCmd: a3 ff 00 00 00 00 02 01 06 00 03 06 00 00 00 00 00 ff 00 15 00 10
```

如果需要修改 publish 的其他参数，可以通过 INI 命令 cfg_pub_set_sig ，并修改参数为自己期望的参数，然后发送即可。

“SecNwBc”按钮

点击该按钮会发送 CFG_BEACON_GET 命令，并把返回的值显示在右边的显示框中。该命令配置是否发送 security network beacon。

在输入框中，修改值，然后按键盘的“Enter”键，会发送 CFG_BEACON_SET，对应的参数是输入框里面的值



“TTL”按钮

点击该按钮会发送 CFG_DEFAULT_TTL_GET 命令，并把返回的值显示在右边的显示框中。该命令获取节点的 default TTL 值。SDK 默认值通过 TTL_DEFAULT 定义。

在输入框中，修改 TTL 值，然后按键盘的“Enter”键，会发送 CFG_BEACON_SET，对应的参数是输入框里面的值

“transmit”按钮

点击该按钮会发送 CFG_NW_TRANSMIT_GET 命令，并把返回的值显示在右边的显示框中。该命令获取节点的 network transmit 的值。低 3bit 表示 network transmit count，高 5bit 表示 network transmit interval。

SDK 默认值定义如下:

```
: #define TRANSMIT_CNT_DEF          (5)
: #define TRANSMIT_INVL_STEPS_DEF   (2)
```

Note:transmit count(5)和 network transmit interval(2)组合起来就是 0x15.

在输入框中，修改 network transmit 的值，然后按键盘的“Enter”键，会发送 CFG_NW_TRANSMIT_SET，对应的参数是输入框里面的值

“Relay”按钮

点击该按钮会发送 CFG_RELAY_GET 命令，并把返回的值显示在右边的显示框中。该命令获取节点的 relay 使能开关的值。

在输入框中，修改该值，然后按键盘的“Enter”键，会发送 CFG_RELAY_SET，对应的参数是输入框里面的值

“Friend”按钮

点击该按钮会发送 CFG_FRIEND_GET 命令，并把返回的值显示在右边的显示框中。该命令获取节点的 friend feature 的使能开关的值。

在输入框中，修改该值，然后按键盘的“Enter”键，会发送 CFG_FRIEND_SET，对应的参数是输入框里面的值

“Proxy”按钮

点击该按钮会发送 CFG_GATT_PROXY_GET 命令，并把返回的值显示在右边的显示框中。该命令获取节点的 proxy feature 的使能开关的值。

在输入框中，修改该值，然后按键盘的“Enter”键，会发送 CFG_GATT_PROXY_SET，对应的参数是输入框里面的值

“Lightness”按钮

点击该按钮会发送 LIGHTNESS_GET 命令，并把返回的值，由 0-65535 的刻度转换为 0-0x64 的刻度后显示在右边的显示框中。

在输入框中，修改该值，然后按键盘的“Enter”键，会发送 LIGHTNESS_SET，对应的参数是输入框里面的值

“C/T”按钮

点击该按钮会发送 LIGHT_CTL_TEMP_GET 命令，并把返回的值，由 800-20000 的刻度转换为 0-0x64 的刻度后显示在右边的显示框中。

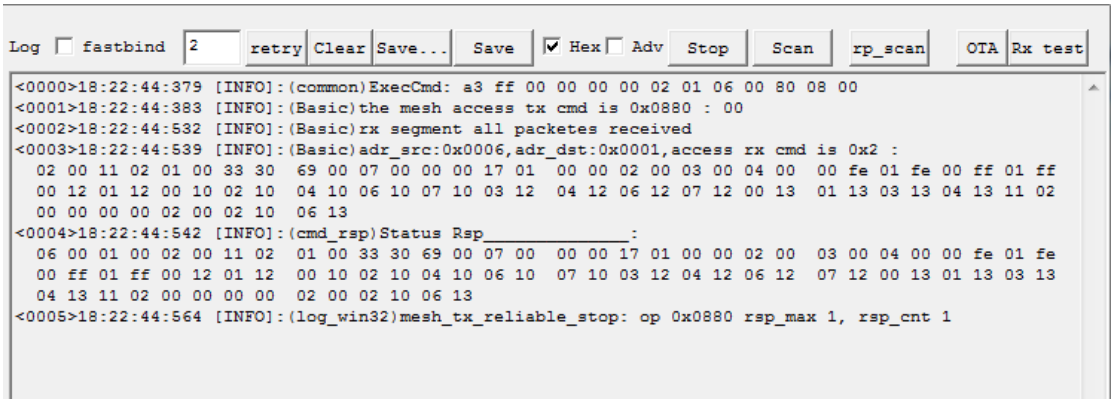
在输入框中，修改该值，然后按键盘的“Enter”键，会发送 LIGHT_CTL_TEMP_SET，对应的参数是输入框里面的值

“RFU”

Reserve for future

“GetCPS”按钮

点击该按钮会发送 COMPOSITION_DATA_GET 命令，并把返回的值显示在 log 窗口中。



“DelNode”

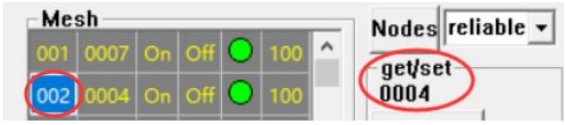
点击该按钮会发送 NODE_RESET 命令，把当前节点踢出当前网络。踢出成功后，节点的红色 LED 会闪烁 8 次，并有执行 reboot 的操作。

4.6 Time model 的操作

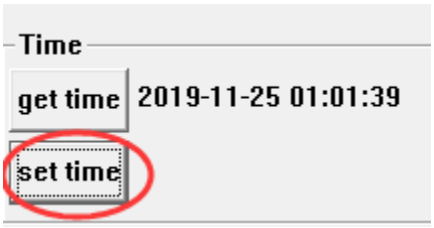
1) 节点的固件端 time model 默认是关闭的，需要设置 MD_TIME_EN 为 1.

   另外，gateway 8269 默认也是关闭，需要打开；gateway 8258 默认是打开的。

2) 双击选择要操作的节点



3) 点击“set time”按钮，工具会把 PC 当前的时间通过 “TIME_SET” 命令发送给节点。此时 time 显示如下，并且时钟会自动刷新。



备注:time set 命令在发送的时候，主要的参数解析如下:

```
typedef struct{
    u32 TAI_sec;      // 32bit is enough for 2000 ~ 2099 year
    u8 TAI_sec_rsv;
    u8 sub_sec;
    u8 uncertainty;
    u16 time_auth       :1;
    u16 TAI_UTC_delta   :15;
    u8 zone_offset;
}time_status_t;
```

- ✧ TAI_sec:是相对于 0 时区的值。
- ✧ zone_offset:要配置为当前的时区，单位是 15minute。

举个例子:北京时间 2019/1/1 09:00:00 (时区:东 8 区)的配置方式。

```
void tx_cmd_time_set_local_sample()
{
    // beijing: 2019/1/1 09:00:00 (time zone: east 8)
    s8 zone_hour = 8;    // Positive numbers are eastwards
    mesh_UTC_t UTC = {0};
    UTC.year = 2019;
    UTC.month = 12;
    UTC.day = 4;
    UTC.hour = 10 - zone_hour;  // translate to 0 time zone.
    UTC.minute = 0;
    UTC.second = 0;
    u32 TAI_sec = get_TAI_sec(&UTC);

    time_status_t time_set = {0};
    time_set.TAI_sec = TAI_sec;
    time_set.zone_offset = get_time_zone_offset(zone_hour*60);

    access_cmd_time_set(0xffff, 1, &time_set);
}
```

上述函数只是介绍 TAI 和本地时间的转换过程，一般是不会这么用的。因为，time set 命令是在手机 APP 或者 PC 上发送的，这两者都有 API 直接获取当前的 TAI_sec 和 zone_offset。比如我们 PC 的上位机获取方式如下:

(需要偏移 OFFSET_1970_2000，是因为 PC 的基准年份是 1970 年，SIG MESH 定义的基准年份是 2000 年)

```
void CTLMeshDlg::OnBnClickedSetTime()
{
    if(0 != Sel_Ele_Check()){
        return ;
    }
    time_status_t settime={0};
    //mesh_UTC_t set_utc;
    CTime time = CTime::GetCurrentTime();
    u32 nTSeconds2 = (u32)time.GetTime();
    settime.TAI_sec=nTSeconds2-OFFSET_1970_2000;
    TIME_ZONE_INFORMATION tz;
    u32 dwRet = GetTimeZoneInformation(&tz);
    settime.zone_offset = get_time_zone_offset(-tz.Bias);
    if(is_support_model_dst(mesh_sel,SIG_MD_TIME_S,1)){
        access_cmd_time_set(mesh_sel, 0, &settime);
    }else{
        LOG_MSG_INFO (TL_LOG_COMMON, 0, 0, "Node not support time model",0);
    }
}
```

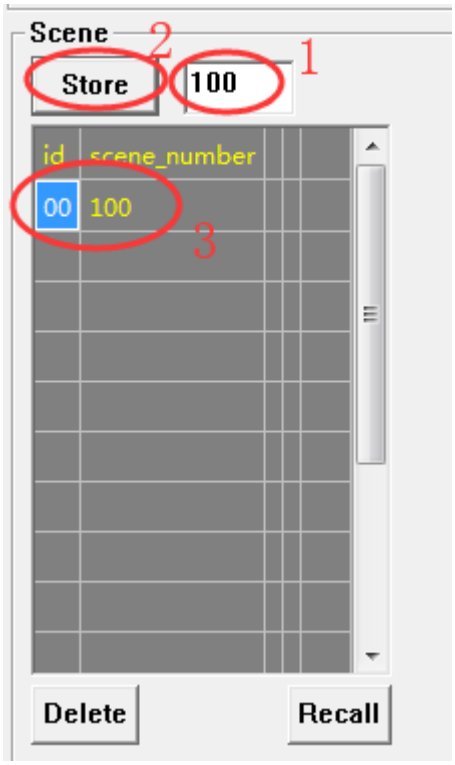
4.7 Scene model 的操作

1) 固件端 scene model 默认是关闭的，需要设置 MD_SCENE_EN 为 1.

   另外，gateway 8269 默认也是关闭，需要打开；gateway 8258 默认是打开的。

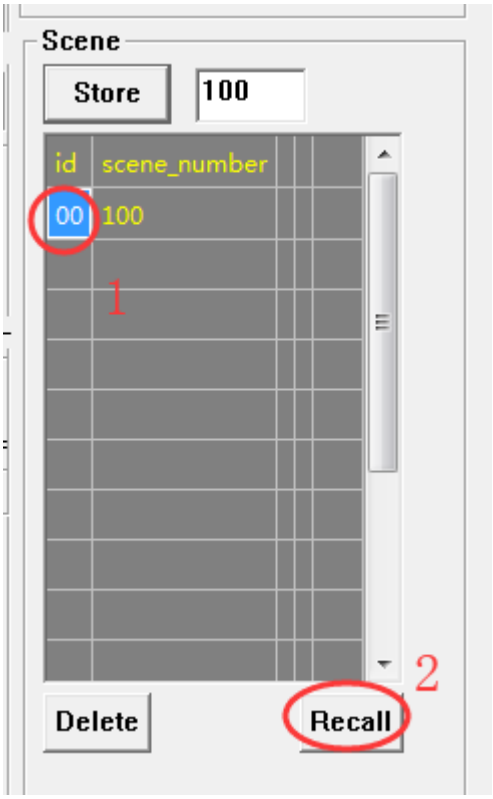
2) 双击选择要操作的节点。
3) 通过 UI 或者 INI 命令，把节点的状态调节到场景期望的配置。比如 generic onoff set 以及 lightness set 命令等。
4) 输入 scene number，然后点击"Store"，就会发送场景添加命令(SCENE_STORE)，把节点当前的状态设置为对应的场景 ID。并在列表中列出已经配置好的场景 ID，如下图"3"所示。

Note:场景添加命令(SCENE_STORE)只有 scene number，没有灯的状态信息，节点在收到场景添加命令后，会自动把当前的状态信息，比如 onoff ，lightness 等保存起来作为该 scene 的状态。

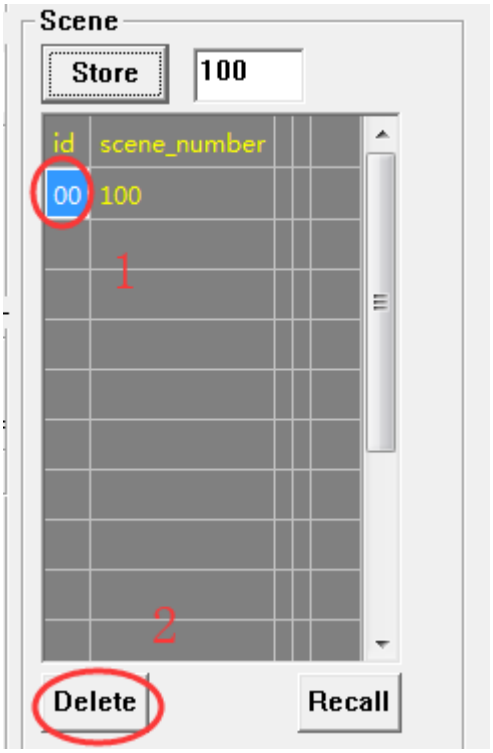


5) Recall 场景，即把灯的状态设置为场景定义的状态。双击下图"1"所示的序号，然后点击"Recall"按钮即可。

Note:Recall scene 后，灯的状态发生改变，但是灯节点的状态没有上报，因为没有配置 publish status，如果需要 UI 能刷新，则需要配置相关 model 的 publish 参数。



6) 修改场景。目前没有专门的修改命令，通过 scene store 的方式进行修改即可。
7) 删除场景。双击下图"1"所示的序号，然后点击"Delete"按钮即可。

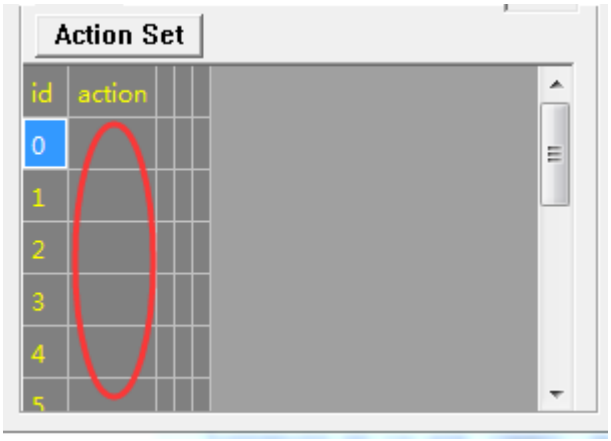


4.8 Scheduler model 的操作

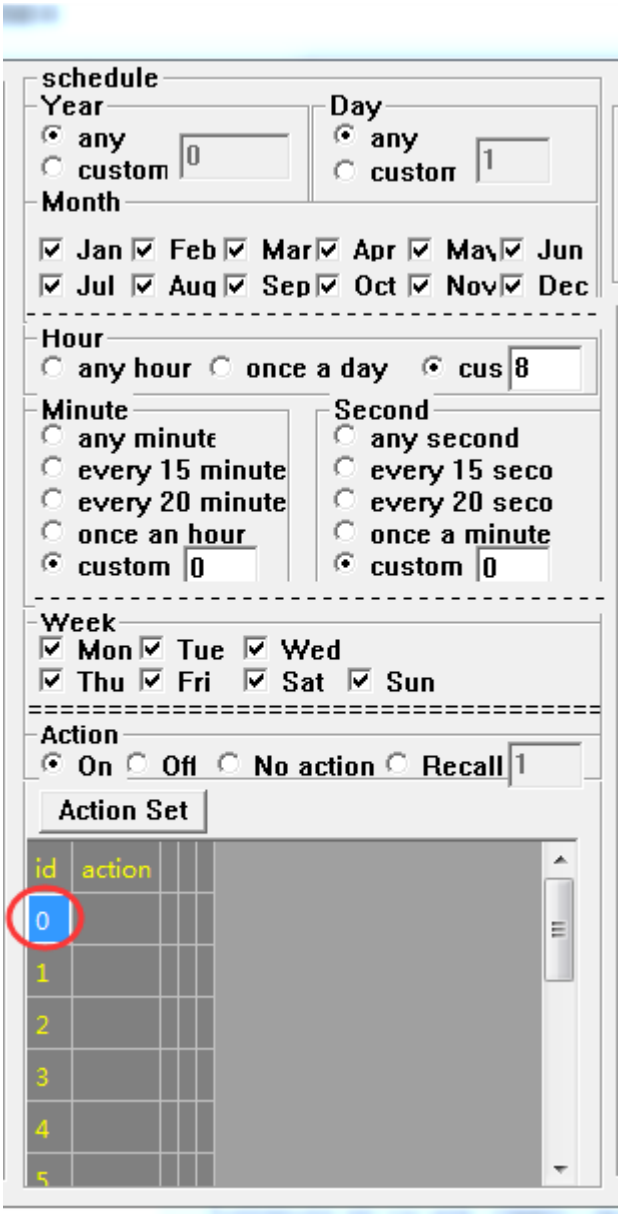
1) 参数介绍(更详细的信息请参考 spec):
   - ✧ Year:any 表示每一年;custom 表示指定某一年,base 是 2000 年,即 0 表示 2000 年，19 表示 2019 年.....
   - ✧ Month:可以单选，多选，全选。Note:不选和全选都表示全选。

- ✧ Day:any 表示每一天；custom 表示指定某一天。
- ✧ Week:可以单选，多选，全选。Note:不选和全选都表示全选。
- ✧ Hour:any 表示每一小时；once a day 表示一天的某一个小时响应一次，这个值是随机数，且每一天都重新生成这个随机数；custom 表示指定某一小时。
- ✧ Min:any 表示每一分钟；every 15 表示 0/15/30/45 响应；every 20 表示 0/20/40 响应；once an hour 表示一小时的某一个分钟响应一次，这个值是随机数，且每一小时都重新生成这个随机数；custom 表示指定某一分钟。
- ✧ Second:规则同 Min。

2) 双击选择要操作的节点，如果下图“action”一栏，如果为空，则表示该节点的这个 id 的 schedule 还未进行过配置。



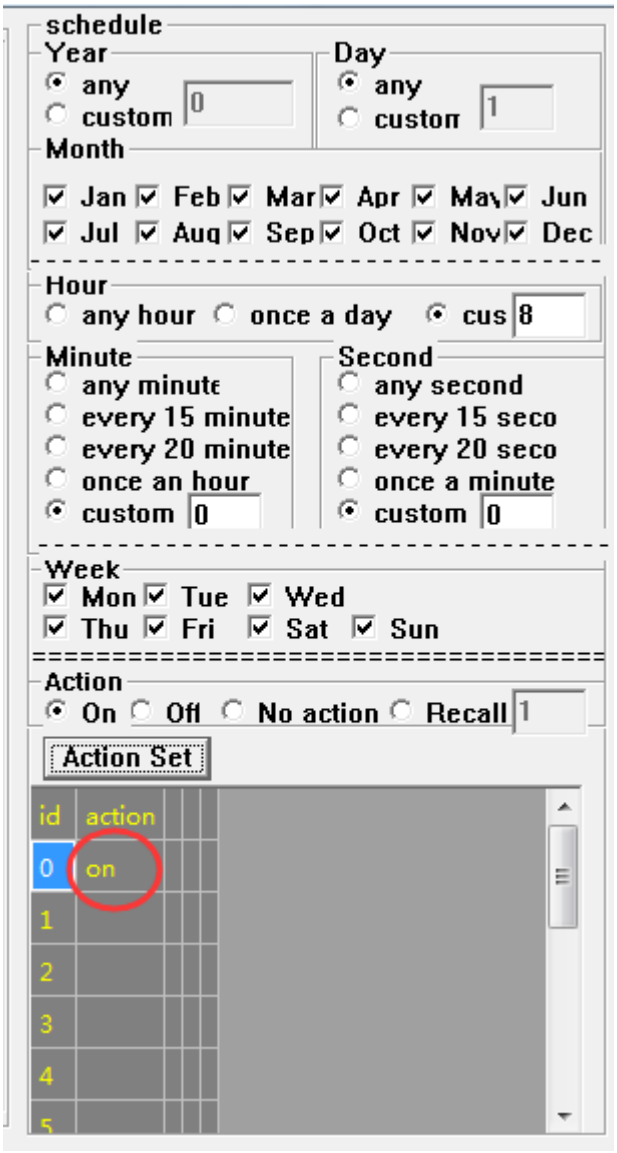
3) 单击”id”所在的列，选择要配置的 scheduler 的 id(SIG 定义最多 16 个，取值范围是 0-15)，被选中的 id 会已蓝色底显示。并且该 id 对应的 schedule 参数会被刷新到上方的 UI。



4) 用户根据需要修改 schedule 参数，然后点击“Action Set”按钮，发送 SCHD_ACTION_SET 进行配置。

因为 UI 显示区的限制，在列表里面只显示 action 参数，对应的是“on”，“off”，“no action”，“recall”，如果什么都不显示，表示该 schedule 还未进行过配置。

如果需要查看某个 schedule id 的详细信息，单击“id”一栏的某个值即可。



5) Schedule 删除。

SIG 目前未定义专门的删除命令。可以通过设置该 schedule 的 action 为“No action”即可。

5. 厂测模式

5.1 厂测模式的目的

厂测模式的目的是用于生产中，不需要进行组网，就能进行一些常用控制，比如 ONOFF，亮度，色温控制等。目前 gateway 和 GATT master dongle 支持厂测模式。APP 暂时不支持。

5.2 厂测模式的参数

- ✧ unicast address 默认是 MAC 的低 15 bit，如果低 15bit 为 0，则使用 1 作为 unicast address。
- ✧ network key，app key，device key，iv index 使用编译的默认值。

5.3 厂测模式的默认能测试的命令

厂测模式能控制的 model 由这个数组 factory_test_model_array[]定义，configure model 能使用的命令由 factory_test_cfg_op_array[]定义。

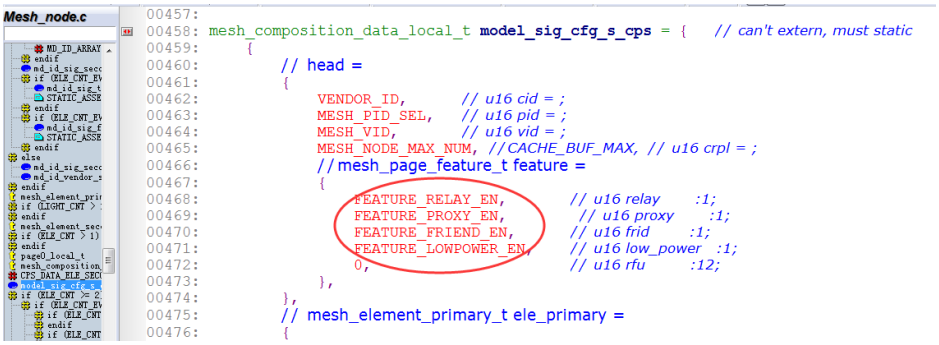
```
const u16 factory_test_model_array[] = {
    SIG_MD_G_ONOFF_S, SIG_MD_LIGHTNESS_S, SIG_MD_FW_UPDATE_S,
    SIG_MD_LIGHT_CTL_S, SIG_MD_LIGHT_CTL_TEMP_S, SIG_MD_LIGHT_HSL_S,
    SIG_MD_LIGHT_XYL_S
};
const u16 factory_test_cfg_op_array[] = {COMPOSITION_DATA_GET, NODE_RESET};
```

厂测模式下的控制，不需要组网，直接按“4.5 控制对应的节点”章节操作即可。需要注意，要使所有节点，包括 gateway 或者 master dongle 都没有进行过组网动作。

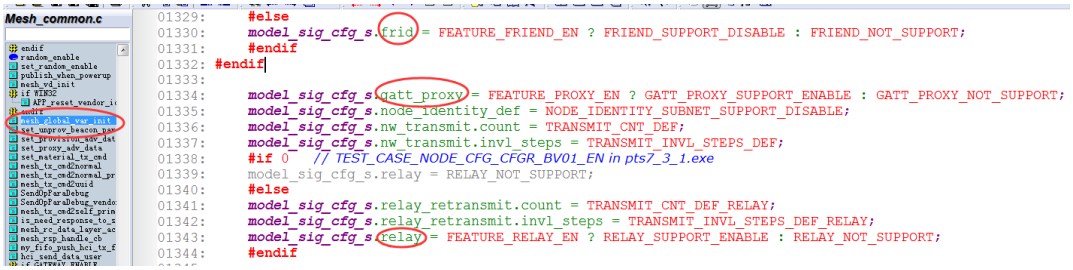
6. SDK 常用的几个模块

6.1 配置 mesh SDK 的默认 feature

1) Mesh 节点在 composition data(model_sig_cfg_s_cps.page0.head.feature)里面描述了该节点支持的 feature，SDK 的初始化如下：



2) composition data 里面只是定义了是否支持，另外在处于“支持”的状态下，可以配置是否开启。详见 mesh_global_var_init()里面配置的地方：model_sig_cfg_s.frid、model_sig_cfg_s.relay、 model_sig_cfg_s.gatt_proxy。



另外，在节点正常工作的过程中，支持通过命令来开启或者关闭这几个 feature，分别是 CFG_FRIEND_SET、CFG_RELAY_SET、CFG_GATT_PROXY_SET。

当不支持对应的 feature 时，model_sig_cfg_s 相关的参数需要设置为 SUPPORT_DISABLE。

3) 我们各个编译工程的 feature 的默认情况，详见第一章“SDK 概述”的“Demo Project” 章节.

6.2 Share model 的介绍

SIG Mesh spec 定义每个 model 都可以独立配置组号，所以，目前每个 model 都有一份独立的组号数据，最大存储个数是 8 个(SUB_LIST_MAX)，如下图。

```
typedef struct{
    u16 ele_adr;     // use as primary address for model_sig_cfg_s_t
    u8 no_pub    :1; // means not support publish function
    u8 no_sub    :1; // means not support subscription function; must before pub and sub par
    u8 pub_trans_flag :1; // transition process was ongoing flag.
    u8 pub_2nd_state  :1; // eg: lightness and lightness linear.
    u8 rsv2;
    bind_key_t bind_key[BIND_KEY_MAX];
    u8 pub_uuid[16];
    cb_pub_st_t cb_pub_st;   // no need to save, fix later
    u32 cb_tick_ms;          // no need to save, fix later
    u16 pub_adr;     // pub_adr and pub_par must existed if sub_list existed //  offset:32
    mesh_model_pub_par_t pub_par;
    u8 rfu3[1];
    u16 sub_list[SUB_LIST_MAX];     // pub_adr, pub_par, sub_list must follow com if existed
    u8 sub_uuid[SUB_LIST_MAX][16];
}model_common_t;
```

但是在某些应用里面，对某一个 model 绑定组号的时候，还希望对其他 model 也自动做组号绑定，减少组号配置时间。这个时候，就需要用到 share model 功能，对应的宏开关是 SUBSCRIPTION_SHARE_EN，打开这个宏开关后，把需要一起自动绑定的组号，sig model 放在 sub_share_model[]这个数组，vendor model 放在 ub_share_model_vendor[]里面就可以了。

其他细节，请参考“全局配置文件说明”章节里面的 sub_share_model[]这个数组的介绍。

6.3 Heartbeat 的演示

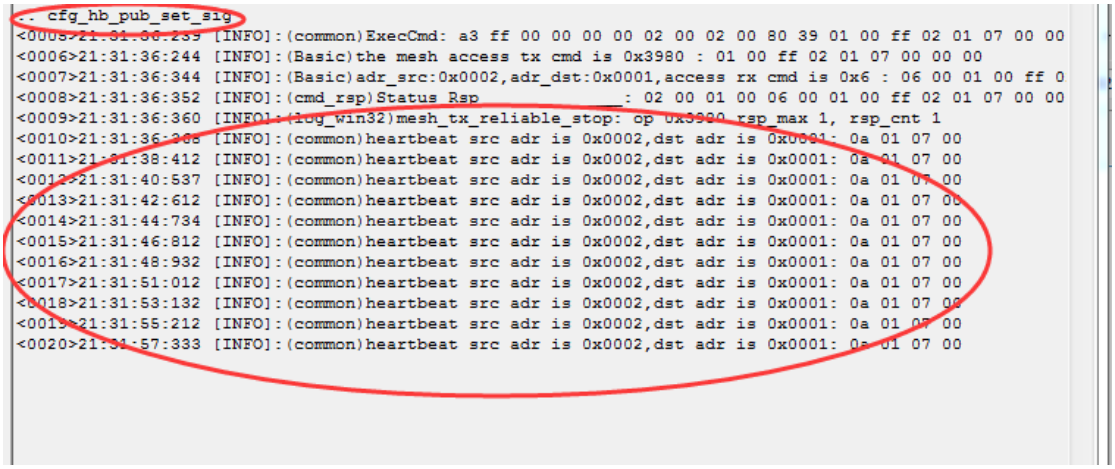
默认情况下，不发送 heartbeat message，通过 HEARTBEAT_PUB_SET 命令配置。发送成功后，节点就会开始发送 heartbeat message。以下是示例:每 2 秒发送一次 heartbeat message，对应的 INI 命令:

```
CMD-cfg_hb_pub_set_sig
=a3 ff 00 00 00 00 00 00 02 00 80 39 01 00 ff 02 05 07 00 00 00
```

该命令配置的参数解析如下:

- 80 39:op code
- 01 00:即表示 heart beat 的 destination address 是 0x0001。
- ff:CountLog，0xff 表示数量无穷大
- 02:PeriodLog，周期等于 2 的(02-1)次方即 2 秒。
- 05:InitTTL，设定 heartbeat message 发送的时候，使用的 TTL 值。
- 07 00:features，只要 relay，friend，proxy feature 这 3 个中的任意一个的状态有变化(在 enable 和 disable 之间切换)就会立即上报 heartbeat message。
- 00 00:NetKeyIndex

在上位机工具中可以看到心跳包。



- ✧ 0a:heartbeat 的 opcode，注意 heartbeat 是 control message。
- ✧ 01:InitTTL，值和 heartbeat set message 里的参数相同。
- ✧ 07:Features，值和 heartbeat set message 里的参数相同。

6.4 Mesh 接收和发送自定义广播包

自定义广播包的发送

sdk 初始化时调用 bls_set_advertise_prepare (app_advertise_prepare_handler)注册广播包发送回调函数，发广播前允许用户访问和修改广播包内容，sdk 默认 10ms 调用一次广播包发送函数。如果用户想发送自定义的广播包，在 gatt_adv_prepare_handler 里参照 mesh 可连接广播包的发送即可。通过 clock_time_exceed 软件计时方式控制广播包间隔，rf_packet_adv_t * p 指向待发送的广播包，用户根据广播包格式修改 p 指向的广播包内容（可参考 set_adv_provision（））， 最后置返回值 ret 为 1 即可，表示要发送广播包。

接收和过滤可连接广播包

sdk 在 rf rx 中断调用 adv_filter_proc()函数对收到的广播包进行过滤，函数返回 0 表示丢弃收到的广播包，返回 1 表示不过滤(接收并压入 blt_rxfifo)。默认情况下,过滤所有可连接广播包。如果用户想接收可连接广播包，打开宏 USER_ADV_FILTER_EN，在 user_adv_filter_proc()函数里对用户关心的广播包进行判断并返回 1，不建议所有的可连接广播包都返回 1，因为此方式不对广播包做任何过滤，所有广播包都会压到 blt_rxfifo 里，包括周围非 mesh 的其他 BLE 产品发出的广播包也会被接收进来，我们的接收 buffer 可能会不能存储那么多数据，导致需要关心的 mesh message 丢失，进而影响 mesh 的收包效果。

blt_sdk_main_loop () 会检查 blt_rxfifo，如果检测到有数据需要处理，会调用 app_event_handle(),用户在此回调函数的 if(LL_TYPE_ADV_NONCONN_IND != (pa->event_type & 0x0F))分支下处理收到的可连接广播包即可。如下图:

```
int app_event_handler (u32 h, u8 *p, int n)
{
    static u32 event_cb_num;
    event_cb_num++;
    int send_to_hci = 1;

    if (h == (HCI_FLAG_EVENT_BT_STD | HCI_EVT_LE_META))      //LE event
    {
        u8 subcode = p[0];
        #if MI_API_ENABLE
        telink_ble_mi_app_event(subcode,p,n);
        #endif
    //------------ ADV packet -------------------------------------------
        if (subcode == HCI_SUB_EVT_LE_ADVERTISING_REPORT)    // ADV packet
        {
            event_adv_report_t *pa = (event_adv_report_t *)p;
            if(LL_TYPE_ADV_NONCONN_IND != (pa->event_type & 0x0F)){
                return 0;
            }

            #if DEBUG_MESH_DONGLE_IN_VC_EN
            send_to_hci = mesh_dongle_adv_report2vc(pa->data, MESH_ADV_PAYLOAD);
            #else
            send_to_hci = app_event_handler_adv(pa->data, ADV_FROM_MESH, 1);
            #endif
        }
```

7. Vendor model 的使用

7.1 添加 vendor model 说明

一般来说没必要增加 model，目前的 SIG model 已经做好，vendor model 可以直接使用已经添加的 vendor model:VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S，添加 op code(即下一章的 vendor) 即可。

除非是要在现有的 vendor model 上 publish 多个 status，才有需要考虑增加 model。目前不建议添加。如果需要添加，请参考目前的结构添加 model，比如参考 MD_SCENE_EN 的宏括起来的相关代码，或者联系我们。

7.2 添加 vendor 命令的注册说明

本文描述的 Command 与 opcode 是相同概念, op code(1BYTE) + vendor id(2BYTE).

Vendor model 的 op code，总共只有 64 个。注意，不是每一个产品类型各有 64 个，而是整个网络同一个 vendor id 的所有产品总共只有 64 个。当 MESH_USER_DEFINE_MODE 选择 MESH_NORMAL_MODE 时，telink 需要预留使用 32 个，即 0xC0---0xDF，客户使用从 0xE0---0xFF。所以请注意节省使用，建议做子命令的方式。如果确实使用超过 32 个，在这种情况下，telink 某些自定义的功能可能就无法打开，所以请联系我们。

另外 vendor 命令的参数区，最大是 377byte，但是大于 8 byte，SIG mesh 底层就会自动采用分包的方式发送。如果是经常使用的控制命令，不建议大于 8byte。

Vendor modle 的注册参考代码 vendor_model.c 的

mesh_cmd_sig_func_t mesh_cmd_vd_func[] = {... ...}的实现,截图如下:

```
#if DEBUG_VENDOR_CMD_EN
{VD_LIGHT_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set, VD_LIGHT_ONOFF_STATUS},
{VD_LIGHT_ONOFF_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_get, VD_LIGHT_ONOFF_STATUS},
{VD_LIGHT_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set, STATUS_NONE},
{VD_LIGHT_ONOFF_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C, cb_vd_light_onoff_status, STATUS_NONE},
#endif
```

注意，mesh_cmd_vd_func[]是 const 类型的数组，所以这个数组是只读,不可改写. 可以节省 RAM 的空间.

另外，如果添加的命令码需要 TID 字段，则还需要另外在 is_cmd_with_tid_vendor()里面注册，详见 3.2.2 的举例说明部分。

7.2.1 mesh_cmd_sig_func_t 介绍

```
typedef struct{
    u16 op;
    u16 status_cmd;
    u32 model_id_tx;
    u32 model_id_rx;
    cb_cmd_sig2_t cb;
    u32 op_rsp;
}mesh_cmd_sig_func_t;
```

✧ op:新增命令的 opcode，不管是 SIG 还是 vendor 命令，都统一用 u16 表示，其中 vendor 命令不需要填写 vendor id 这两个 byte，library 底层会自动添加。

✧ status_cmd:如果该 opcode 是对应某个 acknowledge request command 的 status command，比如 VD_LIGHT_ONOFF_STATUS，则该值写 1；否则写 0。当 model_id_rx 是 client model 的时候，status_cmd 都会是 1. 这个 status_cmd flag 在 Library 会使用到。

✧ model_id_tx:发送该命令对应的 model ID。比如进行 publish status 的时候，需要根据 op 查表 mesh_cmd_vd_func[]，得到这个 model_id_tx，然后通过 mesh_find_ele_resource_in_model()获取到 model 对应的全局变量资源，比如 model_sig_g_onoff_level.onoff_srv，然后再进一步获取这个 model 里面的 model_sig_g_onoff_level.onoff_srv->com. pub_adr 以及其它 publish 参数，然后才能执行 publish status 操作。

✧ model_id_rx:接收该命令对应的 model ID，如果该 node 的 composition data 没有对应的 model id，则不处理这个 opcode。

当支持这个 model 的时候，还需要根据这个 model 里面的参数判断 是否已经绑定了对应的 app key，当 destination address 是 group address 的时候，是否有订阅对应的 group 等。

✧ cb:收到该命令时，调用的回调处理函数 mesh_rc_data_layer_access_cb()→p_res->cb()，客户在这个回调函数里处理自己的应用即可。

✧ op_rsp:如果该 opcode 是 request command，需要 status 回复，则该处写对应的 status op code，否则写 STATUS_NONE。发送 request command 的一端，当发送命令后，用来判断是否收到对应的 status response。

7.2.2 增加 acknowledge command(即 request command with status response)

以 VD_LIGHT_ONOFF_SET 为例:

1) 在 mesh_cmd_vd_func[]中添加:

```
    {VD_LIGHT_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
&cb_vd_light_onoff_set, VD_LIGHT_ONOFF_STATUS}
```

2) 该命令需要 TID 字段(transmit ID)，所以需要在 is_cmd_with_tid_vendor()中添加对应的分支，以及标识 TID 字段在 access payload 中对应的位置。

在 library 里面会有一个统一管理 TID 的全局变量 mesh_tid，当发送命令的时候，会自动加一，并拷贝到命令参数区的 TID 字段。使用举例请参考[Vendor modle 命令格式]

TID 用途:如果在一定时间内(目前默认是 6 秒)，收到重复的 TID，则不执行对应的动作，但是 response 会回复。这个识别的动作在 library 完成，上层应用直接判断 cb_par->retransaction 这个 flag 即可。

TID 一般是 控制灯的状态的命令，才会使用。因为 TID，是为了避免在一定的时间内收到 retry 的时候，会重复执行，导致渐变时间不对，比如当节点收到 OFF 命令，要求延时 1 秒钟到达 OFF 状态，当渐变执行到 0.8 秒的时候，收到了 retry 命令，如果执行这个命令的话，渐变时间需要重新计时 1 秒，这样看到的现象是 1.8 秒后节点才能处于 OFF 的状态。另外也可能会导致灯的闪烁，比如，同一时间有两个 app 对同一个节点进行控制，其中一个执行 generic on，另外一个执行 generic off，并且有执行 retry 动作(message 的 sequence number 不相同，但是 TID 相同)， 这个时候，我们预期是，light 只执行一次 on 和一次 off，最终状态是 on 还是 off 由 最后收到的命令决定。如果有 TID 的识别的话，可以实现预期的需求。但是如果没有 TID 的识别，灯可能会执行了几次 onoff 动作，这样就会有闪烁的问题。

SIG 的标准命令里面也只有类似 onoff set，level set 、lightness set ，CT set 等命令的时候，才会使用 TID。

像一些 lightness get 等 命令，以及 config model 里面的 config set/get 命令都是不需要使用 的。

对于 vendor 命令，没必要添加的时候，尽量不要添加，因为本来有效字节就比较少，没必要再浪费一个 byte。

3) 编写 cb_vd_light_onoff_set()函数，调用 light_onoff_idx()执行开关灯动作。
4) 因为该命令是需要 ack 回复的命令，所以编写对应的 ack 函数 vd_light_onoff_st_rsp()；并在 vd_light_onoff_st_rsp()里面调用 mesh_tx_cmd(VD_LIGHT_ONOFF_STATUS,……)来进行 ack 回复。

注意:

收到命令后回复 status，需要调用 mesh_tx_cmd_rsp()，因为在多个 network key，app key 的网络中，收包的时候，用什么 key 进行解密，回复 status 的时候就必须用对应的 key 进行加密。所以使用 mesh_tx_cmd_rsp()而不是 mesh_tx_cmd2normal_primary()。

mesh_tx_cmd2normal_primary() 是默认都统一用第一个 key 进行发送，一般用于主动发送命令，所以回复 status 的时候，不能直接使用这个函数。

5) 封装发送 VD_LIGHT_ONOFF_SET 命令的接口 vd_cmd_onoff()；

rsp_max 表示需要回复的节点的个数。设置方法:当 adr_dst 为 unicast 时，该值设为 1 或 0 都可以(建议设置为 1)；当 adr_dst 为 group 时，根据 APP 数据库中的记录，设置为 group 拥有的 element 个数。

7.2.3 增加 Unacknowledge command

以 VD_LIGHT_ONOFF_SET_NOACK 为例:

1) 在 mesh_cmd_vd_func[]中添加:

```
{VD_LIGHT_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, &cb_vd_light_onoff_set, STATUS_NONE}
```

2) 该命令需要 TID 字段(transmit ID)，参考 acknowledge command 的添加方法。
3) 编写 cb_vd_light_onoff_set()函数(和 VD_LIGHT_ONOFF_SET 共用)，参考 acknowledge command 的添加方法。
4) 该命令不需要 ack 回复。
5) 封装发送 VD_LIGHT_ONOFF_SET 命令的接口 vd_cmd_onoff()；

参考 acknowledge command 的添加方法。

7.2.4 Publish 函数注册

当通过 CFG_MODEL_PUB_SET 命令设置 light 节点的 model 的 publish 参数后，该 model 就有了 publish 功能，当 model status 变化的时候会自动 publish 一个 status message 到 publish 参数配置的 publish address，另外 publish 参数还可以配置周期发送参数等，详见 spec 关于 publish 命令的参数定义[4.3.2.16 Config Model Publication Set]。

为了实现上述自动 publish 的功能，需要为该 model 注册发送 status message 的 publish 函数，如下图:

```
void mesh_model_cb_pub_st_register()
{
......
    MODEL_PUB_ST_CB_INIT(model_vd_light.srv, &vd_light_onoff_st_publish);
......
}
```

当状态变化，或者 publish 周期时间到之后，mesh stack 会回调该函数发送 status message。

8. 全局配置文件说明

8.1 mesh_config.h

PROXY_HCI_SEL

开发调试用，开发者不需要关注，默认选择 PROXY_HCI_GATT 即可。

DEBUG_VENDOR_CMD_EN

开关 vendor model 的 debug 命令。默认打开。

FAST_PROVISION_ENABLE

Fast provision 是私有模式，可对多节点同时进行组网，执行快速批量以及支持 relay 组网。默认不打开。

MESH_USER_DEFINE_MODE

定义 provision 时的认证模式，MESH_NORMAL_MODE 为 no OOB 模式，其他为 static OOB 模式，详细参考《平台接入》章节的介绍。

SUBSCRIPTION_SHARE_EN

私有模式。目的是在收到对 onoff model 设定组号的命令后，固件端自动对 sub_share_model_sig[]和 sub_share_model_vendor[]里面列出来的 model 也添加该组号。

PROVISION_FLOW_SIMPLE_EN

和标准组网流程一样，也是依次对每个节点进行配网，即同一时间只有一个节点在配网。只是在 node 收到 app key add 后，自动对每一个 model 都执行 key bind 的动作。Provisioner 端不再需要发送 key bind 命令。简化组网流程，减少组网时间。

AIS_ENABLE / MI_API_ENABLE

请参考本文件《平台接入》章节。

LIGHT_TYPE_SEL

选择灯的类型，目前几种类型的灯是互斥的关系。请参考《LIGHT_TYPE_SEL 介绍》章节。

以下控制 model 开关的宏，比如 MD_LIGHTNESS_EN，使能时打开的是 client 还是 server model，或者是两者都打开，取决于 MD_SERVER_EN，MD_CLIENT_EN，以及 MD_CLIENT_VENDOR_EN 这 3 个开关宏。具体规则详见本章节这三个宏的介绍。

LIGHT_TYPE_CT_EN

色温灯相关 model 的开关，包含 Light CTL Server，Light CTL Setup Server，Light CTL Temperature Server，Light CTL Client。

LIGHT_TYPE_HSL_EN

彩色灯(HSL) 相关 model 的开关，包含 Light HSL Server，Light HSL Hue Server，Light HSL Saturation Server，Light HSL Setup Server，Light HSL Client

MD_LIGHT_CONTROL_EN

Lighting Control 相关 model 的开关，默认关闭。包含 Light LC Server，Light LC Setup Server，Light LC Client。

MD_LIGHTNESS_EN

Lightness 相关model的开关，包含 Light Lightness Server，Light Lightness Setup Server，Light Lightness Client。

MD_LEVEL_EN

Level 相关 model 的开关。每一个状态量都可以对应有一个 level model。

MD_MESH_OTA_EN

Mesh OTA 相关 model 的开关，默认关闭。

MD_ONOFF_EN

Generic OnOff 相关 model 的开关，默认开启

MD_DEF_TRANSIT_TIME_EN

Generic Default Transition Time 相关 model 的开关，默认支持，

MD_POWER_ONOFF_EN

Generic Power OnOff 相关 model 的开关，默认支持，和 MD_DEF_TRANSIT_TIME_EN 同时关闭或者打开，因为这两个 model 的参数存在同一个 flash 扇区。

MD_TIME_EN

Time Model 相关 model 的开关，默认关闭

MD_SCENE_EN

Scene Model 相关 model 的开关，默认关闭

MD_SCHEDULE_EN

Schedule 相关 model 的开关，默认关闭，和 MD_TIME_EN 同时关闭或者打开，因为 schedule 依赖于 time。

MD_PROPERTY_EN

Property 相关 model 的开关，包含 Generic User Property Server，Generic Admin Property Server，Generic Manufacturer Property Server，Generic Client Property Server，Generic Property Client。

MD_LOCATION_EN

Location 相关model的开关，包含 Generic Location Server，Generic Location Setup Server，Generic Location Client

MD_SENSOR_EN

Sensor 相关 model 的开关，包含 Sensor Server，Sensor Setup Server，Sensor Client。

MD_BATTERY_EN

Battery 相关 model 的开关，包含 Generic Battery Server，Generic Battery Client。

MD_SERVER_EN

等于 1 时，打开上述介绍的处于使能状态的 server model 包含 SIG 和 vendor 的，比如 lightness server 和 VENDOR_MD_LIGHT_S。

MD_REMOTE_PROV

Remote provision 相关 model 的开关，默认关闭。

MD_CLIENT_EN

等于 1 时，打开上述介绍的处于使能状态的 client SIG model。比如 lightness client。

MD_CLIENT_VENDOR_EN

等于 1 时，打开 client vendor model:VENDOR_MD_LIGHT_C。

MD_VENDOR_2ND_EN

等于 1 时，打开第二个 vendor server model。VENDOR_MD_LIGHT_S2。一般 vendor model 只要一个就够了。

Note:
client model 的开关。对于节点一般是不需要 client model，所以为了节省 Ram，灯端默认关闭。Client model 由 MD_CLIENT_EN，MD_CLIENT_VENDOR_EN 这两个宏开关分别控制，是因为有些灯节点需要打开 vendor 的 client model。但是 SIG 的 client model 不需要打开。

FACTORY_TEST_MODE_ENABLE

产测模式，默认开启。为方便产线测试，未 provision 情况下可以对节点使用默认 key 进行开关，调亮度等简单操作。

MANUAL_FACTORY_RESET_TX_STATUS_EN

通过 5 次上电的复位操作执行后，是否发送 NODE_RESET_STATUS 指令通知 gateway，或者 app。

KEEP_ONOFF_STATE_AFTER_OTA

该宏开关用于决定 OTA 重启后，是否要保持重启前的灯的 onoff 状态。

ELE_CNT_EVERY_LIGHT

每一个 light 的 element 个数。比如一个色温灯需要两个 element，大部分的 model 都会放在第一个 element 上，只有 Light CTL Temperature Server 以及对应的 level model 存在于第二个 element 上。

需要两个 element 的原因是因为 lightness 和 CTL Temperature 都可以通过 level model 相关的命令来控制，如果只有一个 element address，在接收到 level set 命令后，就没办法知道是要控制 lightness 还是 CTL Temperature。

要注意和 LIGHT_CNT ，ELE_CNT 的区别。

LIGHT_CNT 是说一个蓝牙模组上同时存在几个相同的灯珠，比如两个色温灯。

ELE_CNT = ELE_CNT_EVERY_LIGHT * LIGHT_CNT，表示该 node 的总 element 个数。在配网时，占用的 element address 个数也等于 ELE_CNT。

FEATURE_FRIEND_EN

设置是否支持 Friend feature。

FEATURE_LOWPOWER_EN

设置是否支持 Low Power feature。

FEATURE_PROV_EN

Provision 开关，需打开。

FEATURE_RELAY_EN

设置是否支持 Relay feature

FEATURE_PROXY_EN

设置是否支持 Proxy feature

MAX_LPN_NUM

设置一个 friend 节点最大能支持几个 low power 节点，目前为 2，如需修改，建议小于 10(我们测试的最大数量是 10)。不建议改太大，主要是考虑 LPN 节点太多时，friend 给多个 LPN 回复 response 时，发包窗口可能产生冲突的概率增加，导致回复有延时，LPN 节点功耗会增加。另外 RAM 的开销也会增加。

USER_DEFINE_SET_CCC_ENABLE

必须打开。用于给 APP 控制 slave 节点是否上报 notify 或 indication。

SEND_STATUS_WHEN_POWER_ON

设置上电时是否发亮度状态 message，默认发送地址是 0xffff。

8.2 mesh_node.h

SUB_LIST_MAX

每个 model 最多支持的订阅地址个数(即组号个数) 。目前，用户不可修改，因为在 library 中有使用该宏。

BIND_KEY_MAX

每个 model 能 bind key 的个数的最大值。目前用户不能修改，因为在 library 中有使用该宏。

SCENE_CNT_MAX

能配置的最大的 scene 数量。用户可以修改。

8.3 app_mesh.h

8.3.1 宏设定介绍

TRANSMIT_CNT_DEF

设置 message 默认的 transmit cnt，即每个命令的重传次数，

次数 = TRANSMIT_CNT_DEF + 1.

TRANSMIT_INVL_STEPS_DEF

设置 message 默认的 transmit interval，即重传的间隔。

重传的间隔 = (TRANSMIT_INVL_STEPS_DEF + 1) * 10ms + (0--10) ms.

TRANSMIT_CNT_LPN_ACCESS_CMD

对于 LPN 节点，control 命令的的 transmit cnt 由 TRANSMIT_CNT_DEF 定义，比如 friend request，friend poll 等，对于其他 message 则由 TRANSMIT_CNT_LPN_ACCESS_CMD 定义，比如 onoff status.

TRANSMIT_CNT_DEF_RELAY，TRANSMIT_INVL_STEPS_DEF_RELAY，

Relay 的重传次数和间隔。

MESH_ADV_CMD_BUF_CNT

设置 发送 message 的 buffer size，不包含 relay message

MESH_ADV_BUF_RELAY_CNT

设置 relay message 的 buffer size

SEC_NW_BC_INV_DEF_100MS

设置组网成功后，发送 security beacon 的 interval，单位 100ms。

8.3.2 函数介绍

mesh_tx_cmd(material_tx_cmd_t *p)

通用的发送命令的函数

1) 参数介绍:

```
type typedef struct{
union{  //指向参数地址
u8 *par;
u8 *p_ac;
};
union{  //参数的长度
u32 par_len;
u32 len_ac;
};
u16 adr_src; //源地址
u16 adr_dst;//目的地址
u8* uuid; //指向虚拟地址
model_common_t *pub_md; // 指向 model 参数
u32 rsp_max; //需要回复的节点的个数
u16 op; // 命令码
u16 nk_array_idx; // network_key index
u16 ak_array_idx; // app_key index
u8 retry_cnt; // retry 次数
}material_tx_cmd_t;
```

参数值由调用它的函数 mesh_tx_cmd2normal_primary(u16 op, u8 *par, u32 par_len, u16 adr_dst, int rsp_max)带的参数决定。

2) 返回值

返回值返回 0 表示命令执行成功；

返回非 0 表示发送失败，比如当前还在发送命令，不能接受新的命令(即 busy 状态)，以及参数非法，等。

int mesh_tx_cmd_primary(u16 op, u8 *par, u32 par_len, u16 adr_dst, int rsp_max)

该函数是把 adr_src 固定为 ele_adr_primary，然后对 mesh_tx_cmd()进行封装。

8.4 app_provision.c

u8 is_provision_success():

获取节点是否处于配网成功状态。

u8 is_provision_working()

获取节点是否处于正在配网过程。

8.5 mesh_node.c

is_own_ele()

判断 adr 是否是本身的 element address。

8.6 mesh_common.c 文件介绍

HCI fifo

hci_tx_fifo，hci_rx_fifo 是定义和外设传输数据的 fifo，比如 gateway 节点和 gateway 上位机的 USB 通讯。

mesh_get_proxy_hci_type()

定义 proxy 的类型，默认是 PROXY_HCI_GATT。PROXY_HCI_USB 是开发 debug 模式，用户不需要使用。

mesh_tid_save()

TID 保存函数，比如 generic on/off 等命令需要用到 tid，不执行 deep sleep 模式的情况下是不用保存的，上电初始化为 0 即可。有 deep 模式的，比如 switch 等，因为每次按键都会初始化所有变量，包括 tid，所以需要保存。

adv_filter_proc()

IRQ RX 收到校验正确的广播包后，通过这个函数进行过滤，比如可连接广播包直接丢弃。详见 code。

const u16 sub_share_model[]

```
    const u16 sub_share_mode= {
        SIG_MD_G_ONOFF_S, SIG_MD_G_LEVEL_S, SIG_MD_LIGHTNESS_S,
SIG_MD_LIGHTNESS_SETUP_S,
        SIG_MD_LIGHT_CTL_S, SIG_MD_LIGHT_CTL_SETUP_S, SIG_MD_LIGHT_CTL_TEMP_S,
        SIG_MD_LIGHT_HSL_S, SIG_MD_LIGHT_HSL_SETUP_S, SIG_MD_LIGHT_HSL_HUE_S,
SIG_MD_LIGHT_HSL_SAT_S,
        SIG_MD_SCENE_S, };
```

参考 mesh_config.h 章节里面对 SUBSCRIPTION_SHARE_EN 的介绍。

Model 默认绑定关系，在设置订阅地址（分组）的时候，这几个 model 是同时生效的。

具体的调用流程:

收到 CFG_MODEL_SUB_ADD-> mesh_rc_data_layer_access_cb() -> mesh_cmd_sig_cfg_model_sub_set()->share_model_sub_by_rx_cmd()->share_model_sub( ).

entry_ota_mode()

当收到 OTA start 命令后，会回调此函数。

ota_condition_enable()

允许 GATT OTA 的条件。当 GATT 连接成功，并且收到 set proxy filter 后，pair_login_ok 会被置为 1.(Note:收发 set proxy filter 需要使用 network key 进行加解密)

proc_telink_mesh_to_sig_mesh()

检测 ota 前的 firmware 是 SIG mesh 还是其他已经定义的 SDK，比如 telink mesh 如果不是 SIG mesh，则表示进行产品切换，需要执行参数初始化。

mesh_ota_reboot_proc()

mesh ota 完成后，延时 1.5 秒再重启。

调用方式:main_loop() -> mesh_loop_process()->mesh_ota_reboot_proc()

mesh_ble_connect_cb

GATT connect 成功之后，会回调此函数。

mesh_ble_disconnect_cb

GATT disconnect 之后，会回调此函数。

updata_para_change_MTU()

在连接成功后，适当的时间申请更改 BLE 连接参数请求，主要避免在 discovery 和组网期间发起连接参数更新。

gatt_adv_prepare_handler()

1) relay_adv_prepare_handler()

relay buffer 是独立的，和主动发包用的是不相同的 fifo，因为 relay buffer 可以在主动发包 transmit interval 间隙发包。

发包优先级: 主动发包 > relay > 可连接广播包。

2) 其他是 GATT 相关的广播包。

app_advertise_prepare_handler ()

当 BLE 协议栈底层允许发广播包时，会回调此函数，当前任务中如果有需要发送广播包(包括可连接广播包，beacon 包等)，对参数 p 指向结构体赋值即可。

发包优先级:Friend Node 收到 LPN 的 poll 后回复给 LPN 的 message > 主动发送的 network message > relay > 可连接广播包.

1) get_adv_cmd():

返回值为等待发送的 mesh message 数据包指针，非零表示有数据包需要发送:包括 MESH_ADV_TYPE_MESSAGE、SECURE_BEACON 类型的 MESH_ADV_TYPE_BEACON。

2) mesh_adv_cmd_set()

把等待发送的数据包拷贝给 BLE 协议栈。

3) p_bear->trans_par_val:

即 spec 中提到的 retransmit 的概念。包含 transmit count 和 transmit interval。

4) mesh_rsp_random_delay_step

节点收到目标地址是 group 的地址时，在回复 response 时需要增加让 Random delay，详见 mesh_rc_data_layer_access_cb()的介绍。

5) adv_retry_flag

用于一些需要取消 network transmit interval，连续发包的情景，比如 LPN 建立 friend ship 后发送的 poll 等。

app_l2cap_packet_receive ()

当 BLE 协议栈收到有 payload 的时候，回调此函数，然后调用 blc_l2cap_packet_receive() 函数进行数据分析。调试时，开发者可以把该数据打印出来，便于分析。

chn_conn_update_dispatch()

暂未使用。

sim_tx_cmd_node2node()

每隔 3 秒发送 unreliable 的开关灯命令。做 demo 演示用。

usb_id_init()

USB ID 配置，当多个 dongle 接在同一台 PC 的时候，必须设置不同的 ID，PC 才能识别是不同的设备。

ble_mac_init()

当 MAC 的参数位置，低 4byte 为 0xFFFFFFFF，会随机生成一个 MAC 并保存起来。

mesh_scan_rsp_init()

初始化的时候填充 scan rsp 的固定字段，比如 mac address。因为在建立数据库的时候，需要用到 mac，并且 IOS 系统不能直接从广播包数据的 AdvA 字段获取，所以需要在 scan response 的内容里面标识出来。

mesh_scan_rsp_update_adr_primary()

当 primary adr 有变更(比如进行 provision 时)，以及上电初始化时，会回调该函数更新 scan rsp 中的 adr_primary 字段。

publish_when_powerup()

上电，经过一个 random 时间(publish_powerup_random_ms)之后发送相应的 status，通知 app 或者 gateway 节点上线。

mesh_vd_init()

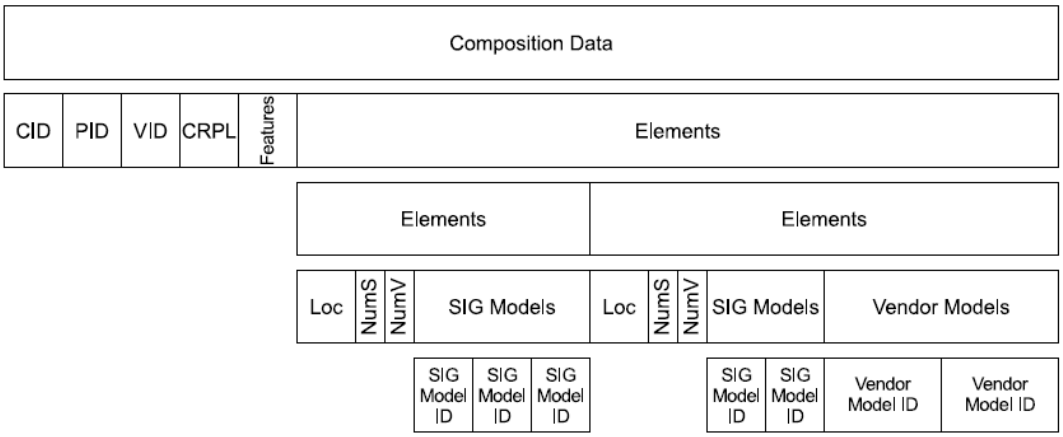
多个 project 的关于 mesh 相关的公共处理部分，在 mesh_init_all()调用。

mesh_global_var_init()

全局结构体变量初始化函数，在读取存储在 flash 里面的相关参数前运行。用于设定编译默认值。如果 flash 里面有相关参数，则以 flash 里面的为准。

model_sig_cfg_s_cps

即 composition data，相关定义请参考 model_sig_cfg_s_cps 结构体定义，以及 spec 相关资料。



set_unprov_beacon_para()

- p_uuid:为设置 uuid 的指针，长度为 16 字节。
- p_info:为设置 oob_info 的指针，长度为 2 字节。
- p_hash:为设置 URI 的 hash 值的指针，长度为 4 字节。
- uri_para:为 uri 连接的指针，最大长度为 40 字节。
- uri_len:为实际用到的 uri 部分的数据的长度，最大长度不能超过 40。

以上参数用来配置未 provision 节点的 beacon 包。

set_provision_adv_data()

- p_uuid:为设置 uuid 的指针，长度为 16 字节。
- oob_info:为设置 oob_info 的指针，长度为 2 字节。

以上参数用来配置未完成 provision 时的广播包部分的参数。

set_proxy_adv_data()

- p_hash:为设置 hash 的指针，长度为 8 字节。
- p_random:为设置 random 的指针，长度为 8 字节。
- node_identity 表示的是广播包的类型。

如果 node_identity 为 0，表示发送的广播包类型为 advertising with Network ID，此时 p_hash 和 p_random 参数均不生效。

如果 node_identity 为 1，表示发送的广播包类型为 advertising with Node Identity，p_hash 和 p_random 参数的设置才能生效。

以上参数用来配置已经完成 provision 之后，发送 proxy 连接的广播包。

uart_drv_init()/usb_bulk_drv_init()

串口和 USB 初始化，通过宏 HCI_ACCESS 选择串口或 USB，blc_register_hci_handler 注册回调函数。用户可通过调用 my_fifo_push_hci_tx_fifo 往 hci_tx_fifo 推送需要上报的数据。

set_material_tx_cmd()

设置发包的参数:

1) op:vendor op code，只需要输入一个 byte，vendor id 不需要输入，会自动填充。
2) rsp_max:只对有 status 回复的 command 有效，用于判断是否收到了足够的 status 个数。当 destination address 是 unicast 时，该值是 0 或 1(0 或 1 的作用相同，因为是收到 status 的时候才进行判断)，当 destination address 是 group 时，该值是属于这个 group 的节点个数。
3) retry_cnt:只对有 status 回复的 command 有效，当命令发出一段时间后，未收到指定 rsp_max 个数的 status，则触发 retry flow，retry 的次数由 retry_cnt 决定。
4) uuid:当需要发送的 destination address 是 virtual address 时，需要输入 uuid，否则为 0.
5) nk_array_idx:因为 mesh 支持多个 netkey，此处是设定使用 netkey 数组里面的 key 的数组索引值，注意不是 组网时分配的 global netkey index.
6) ak_array_idx:因为 mesh 支持多个 appkey,此处是设定使用 appkey 数组里面的 key 的数组索引值，注意不是 组网时分配的 global appkey index
7) pub_md:当需要执行 publish status 时，需要设置为 model 的指针，因为在发送 message 时需要使用 pub_md-> pub_par 里面的 mesh_model_pub_par_t 参数，比如 ttl，network transmit，network count，appkey index，而不是使用这些参数对应的默认值。当不是执行 publish status 时，pub_md 设为 0.

mesh_tx_cmd2normal_primary()

节点端主动发送命令调用 API。参数区详见 set_material_tx_cmd()的相关介绍。

SendOpParaDebug_vendor()

当是 WIN32 模式时，包括 gateway，app，par_tmp[2:3]的解析，详见 ini 格式的解析。

is_need_response_to_self()

当自己收到从自己发过来的需要 status 回复的命令时,该函数用来设定是否需要回复 status。

mesh_rc_data_layer_access_cb()

当节点端收到一个发给自己的命令(条件:model 支持，destination address 匹配)，将会回调该函数。

1) Vendor Op code 在 VD_OP_RESERVE_FOR_TELINK_START 和 VD_OP_RESERVE_FOR_TELINK_END 之间的，是预留给 telink 的 opcode，客户不应该使用。
2) mesh_need_random_delay :当节点收到目标地址是 group 的地址时，在回复 response 时需要增加让 Random delay，尽量避免多节点在同一时刻回复。
3) p_res->cb: 这个是每个 op code 对应的回调函数，对应的是 mesh_cmd_sig_func[]->cb 以及 mesh_cmd_vd_func[]->cb.

mesh_rsp_handle_cb()

用于上报 gateway 收到的 status message 给上位机。上报的方式包括 USB 或 UART。

hci_send_data_user()

hci tx fifo 的 buffer data 区，前面两个 byte 是 len，第三个是 data type，用于识别数据是哪种类型。此处类型为 HCI_RSP_USER，其他类型请参考 hci_type_t.

mesh_tx_reliable_stop_report()

gateway 在发送 reliable 命令，达到结束条件后的回调函数。

app_hci_cmd_from_usb()

在 blt_sdk_main_loop() 函数中，回调 app_hci_cmd_from_usb()处理从上位机收到的命令，并通过 app_hci_cmd_from_usb_handle()进行命令解析和执行。

app_hci_cmd_from_usb_handle ()

buff 对应的是 ini 格式的数据，详见 ini 章节的详细介绍。

8.7 cmd_interface.h 文件介绍

access_cmd_get_level()

获取 element 的 level 值。该函数是把 opcode 设为 G_LEVEL_GET，然后对 mesh_tx_cmd()进行封装。

access_cmd_set_level()

设置 element 的 level 值。该函数是对 mesh_tx_cmd()进行封装。当 ack 为 1 时，opcode 设为 G_LEVEL_GET；当为 0 时，opcode 设为 G_LEVEL_SET_NOACK。该命令用到的 tid 参数由内部统一进行管理实现，上层开发不需要关心。

access_set_lum()

通过输入 lum 的方式(范围是 0~100)来设置 level 值。该函数是对 access_cmd_set_level ()进行封装。

access_cmd_onoff()

设置 element 的 onoff 值。该函数是对 mesh_tx_cmd()进行封装。当 ack 为 1 时，opcode 设为 G_ONOFF_GET；当为 0 时，opcode 设为 G_ ONOFF _SET_NOACK。该命令用到的 tid 参数由内部统一进行管理实现，上层开发不需要关心。

8.8 vendor_model.c 文件介绍

该文件介绍 vendor model 对应的命令码的发送和收到该命令码后执行的相应回调函数等。

注意事项:一般来说对于节点端(非 provisioner 角色)只会使用一个 vendor id，并且会在 composition data 里面显示出来， Vendor model 的 op code，总共只有 64 个。注意，这个不是每一个产品类型各有 64 个，而是整个网络同一个 vendor id 的所有产品的所有 vendor model 的 op code 加起来总共只有 64 个。所以请注意节省使用，能做成子命令的时候，尽量做成子命令。

另外，因为 telink 也会有需求使用 vendor op code 来实现一些自定义的 feature，所以目前暂定 0xC0—0xDF 是预留给 telink 使用，0xE0—0xFF 是留个客户使用。

vendor 命令码的注册

详见 3.2 章节的介绍。

mesh_search_model_id_by_op_vendor()

通过 opcode 在 mesh_cmd_vd_func[]数组里面查找对应的资源。用户了解即可，不需要更改该函数。

vd_cmd_key_report()

上报按键事件。在 8258_mesh_switch 工程中使用。

int SendOpParaDebug(u16 adr_dst, u8 rsp_max, u16 op, u8 *par, int len);可以认为是和 mesh_tx_cmd_primary()是等效的。

is_cmd_with_tid_vendor()

该函数判断并返回该 opcode 是否需要带 tid，如果有则返回 1，并把 tid 在参数区的位置通过 tid_pos_out 来返回。否则返回 0。

8.9 mesh_test_cmd.c 文件介绍

该文件用于存放测试命令的实现。

9. 8258 MESH 工程介绍

9.1 app_config_8258.h

PCBA_8258_SEL

选择 PCBA，默认选择 48pin 的 dongle 板。另外两个是开发板。

FLASH_1M_ENABLE

当芯片内置是 1M flash 的时候，需要把该宏打开，因为 flash map 不一样，另外，出厂的时候，MAC 默认放在 0Xff000，而 512K 是放在 0x76000.

HCI_ACCESS

设置 hci 的接口，等于 HCI_USE_USB 时使用 USB， 等于 HCI_USE_UART 时使用 UART。默认不需要使用 HCI 收发数据。

UART_GPIO_SEL

设置 uart 的 IO

HCI_LOG_FW_EN

firmware 默认不打开调试信息，如有需要打开调试信息用户可打开，打开方式参考 log 输出章节。

ADC_ENABLE

设置是否使能 ADC。

ONLINE_STATUS_EN

私有 mesh SDK 的 online status 功能。实时发送状态数据，实时性能比 publish 的机制更优化。

DUAL_MODE_ADAPT_EN

SIG mesh + ZigBee 双模

DUAL_MODE_WITH_TLK_MESH_EN

SIG mesh + 私有 mesh SDK 双模。

TRANSITION_TIME_DEFAULT_VAL

默认的渐变参数。包括上电的时候的渐变参数，以及当收到支持渐变参数的命令，比如 generic onoff，但是收到的命令又没有渐变参数时，灯端会按照 TRANSITION_TIME_DEFAULT_VAL 来执行渐变动作。默认渐变时间 1 秒，如需关闭渐变，把 TRANSITION_TIME_DEFAULT_VAL 设置为 0 即可，具体解析格式请参考 trans_time_t.

SW1_GPIO/SW2_GPIO

Dongle 板的 2 个按键，做测试 button 的 IO，默认没有使用。

如果使用的时候，需要改 IO，则改完 IO 后，还需要同步修改对应的 PULL_WAKEUP_SRC_XXX 和 XXX_INPUT_ENABLE。

PWM_R/ PWM_G/ PWM_B/ PWM_W

设置 PWM 对应的 IO

GPIO_LED

定义 led 指示灯的 IO，比如组网完成，恢复出厂设置等，进行闪灯动作的 IO 定义。

9.2 app.c 文件介绍

9.2.1 广播包以及广播响应包的定制

广播包

可连接广播包:目前可连接广播包的格式已经由 SIG MESH 的 spec 定义完所有字段，具体定义格式，请参考 PB_GATT_ADV_DAT 结构体，或者 spec 相关资料。

广播响应包

用户可通过 mesh_scan_rsp_init()修改 u8 tbl_scanRsp [] = {} 这个数组来定制广播响应包，广播响应包最长有 31 个 BYTE，目前只使用了一部分，客户可在 mesh_scan_rsp_init()里面对 rsv 字段赋值，填充自己想要的信息。

```
typedef struct{
 u8 len;
 u8 type;
 u8 mac_adr[6];
 u16 adr_primary;
 u8 rsv_telink [10];   // not for user
 u8 rsv_user[11];
}mesh_scan_rsp_t;
```

9.2.2 fifo 部分配置

```
MYFIFO_INIT(blt_rxfifo, 64, 16);
MYFIFO_INIT(blt_txfifo, 40, 32);
```

配置 BLE 协议栈底层的收包 buff 和发包 buff。如果有 RAM 不足时，可修改，但一般不建议更改。

9.2.3 app_event_handler ()

回调处理函数:当 BLE stack 收到:adv (包含可连接广告包，beacon 包等)、connect request 包、BLE 连接参数更新包、BLE 连接断开，等，事件后会回调该函数进行处理。

目前 SIG MESH 通讯用的所有 beacon，都是在(subcode == HCI_SUB_EVT_LE_ADVERTISING_REPORT) 这个分支里面处理。

其他事件的处理，请参考对应的 code，目前仅做简单的 LED 指示灯处理。用户有需要，可以添加相关功能。

HCI_SUB_EVT_LE_ADVERTISING_REPORT

接收到广播包的处理分支，客户可以在该分支下添加对应的事件以及处理的函数。

HCI_SUB_EVT_LE_CONNECTION_COMPLETE

蓝牙连接上之后产生的事件回调，在蓝牙连接上之后，BLE 协议栈底层会回调到这个分支。

HCI_CMD_DISCONNECTION_COMPLETE

蓝牙连接断开之后，BLE 协议栈底层会回调到这个分支。

9.2.4 main_loop ()

- ✧ mesh_loop_proc_prior():实时性要求相对较高的函数。
- ✧ blt_SDK_main_loop ():BLE 协议栈的 main_loop 函数
- ✧ proc_led():led 指示灯事件处理函数。
- ✧ factory_reset_cnt_check():factory reset 处理函数。支持上电 5 次的方式进行复位。具体请参考 factory reset 章节。
- ✧ mesh_loop_process():SIG mesh 相关的 loop 函数，包括 reliable 命令的 retry 机制、segment ack 超时回复、TID 超时检测机制等。
- ✧ sim_tx_cmd_node2node():提供一个定时发送开关命令的 demo 演示接口。

9.2.5 user_init()

- ✧ proc_telink_mesh_to_sig_mesh():sig mesh 和 telink mesh 相互 OTA 时，需要对参数进行初始化，因为两者间的参数格式是不兼容的。目前的做法是，当检测到 mesh 类型变化时，会把新 mesh 用到的参数区都进行清除。
- ✧ bls_ll_setAdvParam():广播包间隔等参数定义，目前暂时不建议用户修改。
- ✧ blc_ll_setAdvCustomedChannel():广播 channel 定制，sig mesh 要求使用标准的 37/38/39，但是测试过程中，可以更改该 channel，方便调试。
- ✧ bls_ll_setAdvEnable(1);使能广播包的发送。
- ✧ rf_set_power_level_index (MY_RF_POWER_INDEX);默认设置发送功率为 3dbm。
- ✧ mesh_init_all():sig mesh 相关的初始化

9.2.6 void proc_ui()

该函数主要是做些 UI 方面的处理，如 button 检测函数，以及对应的测试代码。

9.3 app_att.c 文件介绍

pb_gatt_provision_out_ccc_cb()

使能 provision out 部分的发送使能。只有将 provision_Out_ccc 设置为 01 00 才能使得 mesh 节点能够正常的返回指令。

pb_gatt_Write ()

对应的 uuid 为 my_pb_gatt_in_UUID 的回调函数，用于处理 provision 命令的回调函数。

proxy_gatt_Write()

处理 proxy 命令的回调函数，其中 proxy 的指令分为三种，分别为 MSG_PROXY_CONFIG，MSG_MESH_BEACON，MSG_NETWORK_PDU 命令头。

- ✧ MSG_PROXY_CONFIG：用于配置 proxy 通信的白名单和黑名单。
- ✧ MSG_MESH_BEACON：控制 beacon 指令（notify）的接收。
- ✧ MSG_NETWORK_PDU：用于控制开关灯的指令。

attribute_t my_Attributes[]

SIG_mesh 中的服务列表，里面包含基本的 att，以及关于 SIG_mesh 部分的 att 的内容。

9.4 light.c 文件介绍

修改 IO 口

修改 PWM_R/ PWM_G/ PWM_B/ PWM_W 对应的 IO 口即可。

```
#define PWM_R       GPIO_PC2                //red
#define PWM_G       GPIO_PC3                //green
#define PWM_B       GPIO_PB6                //blue
#define PWM_W       GPIO_PB4                //white
typedef struct{
    u32 gpio;
    u8 id;      // pwm id
    u8 invert;  // pwm 的翻转属性
    u8 func;    // 采用第一功能 PWM 还是 第二功能 PWM
    u8 rsv[1];
}light_res_hw_t;
light_res_hw_t light_res_hw[LIGHT_CNT][4];
```

light_res_hw 定义 PWM IO 的属性。

func:有 3 个 PWM 管脚有两种 PWM 输出，包括 GPIO_PC1，GPIO_PC4，GPIO_PD5.此处定义是使用第一功能还是第二功能。

定义 PWM 的宏的时候，我们默认把 dongle 板或者开发板上的四个 led 都列出来了，也就是 RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B/ RES_HW_PWM_W，但是针对不同类型的灯类型，会选用某几路就可以了，不需要都用完，这个由 light_res_hw 来设置。比如:

LIGHT_TYPE_CT:选用 RES_HW_PWM_R 和 RES_HW_PWM_G，其中红色代表暖色灯珠，绿色代表冷色灯珠。

LIGHT_TYPE_HSL:RES_HW_PWM_R, RES_HW_PWM_G, RES_HW_PWM_B 分别对应 RGB 灯珠，在调节灯光的时候，会在 dim_refresh()中把 HSL 转换为 RGB 值，来驱动 LED。

LIGHT_TYPE_LPN_ONOFF_LEVEL:使用 RES_HW_PWM_R,由于 retention 低功耗的特性，目前只能控制 onoff。

LIGHT_TYPE_PANEL:默认 LIGHT_CNT 等于 3，占用 3 个 element address，有 3 个 onoff server model，3 个 onoff model 分别对应 RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B 这三个灯珠。

PWM 频率设定:

修改 PWM_FREQ 即可。

注:如果编译的时候，STATIC_ASSERT(PWM_MAX_TICK < 0x10000)报错，是因为 pwm 的 tick 溢出了。因为 pwm 的 tick 是 16bit 的，PWM 时钟默认是 PLL 时钟，当 PWM_FREQ 设置过小时，pwm 的 tick 会溢出，此时可以设置 PWM 分频，即设置 PWM_CLK_DIV_LIGHT。一般情况下，不需要配置。

ct_flag

ct_flag 只有 在 LIGHT_TYPE_CT_HSL 模式的时候才有作用。该模式有 CT 和 HSL 两种灯珠，但是同一时间只有一种灯珠在点亮。ct_flag 等于 1 表示，当前是 CT 灯的状态，ct_flag 等于 0 表示，当前是 HSL 灯的状态。

light_res_sw_save

这个变量包含了所有和灯的状态相关，且需要保存的参数。比如 lightness，色温，等。注意，保存的格式都是转换为 generic level 之后才保存的。(取值范围是 -32768 ~ 32767).

采用 level 保存的原因是:(1) 所有的状态值都可以和 level 转换. (2)level 的刻度是最精细的。(3)同一个状态量只能存储一个参数，比如色温值，不能同时存储 色温值，又同时存储色温转换为 level 之后的值。因为有多份数据可能会出现数据不同步的问题。

综上考虑，都是按 level 来存储。否则会有精度丢失的问题。

亮度值和 PWM 值的非线性对应关系

开发者可根据实际的灯的特性，修改 rgb_lumen_map[]数组。

// 0-100% (pwm's value index: this is pwm compare value, and the pwm cycle is 255*256)

const u16 rgb_lumen_map[101] = {}

mesh_global_var_init_light_sw()

第一次上电的时候，设置的初始值，相当于编译默认值。当 flash 有存储对应的参数后，则以 flash 存储的为准。

light_res_sw_load()

从 flash 加载灯的状态的参数。

light_pwm_init()

读取灯的状态参数数后，调用该函数。初始化 PWM register 后，先设置为 OFF 的状态，然后再看是否需要打开，以及是否需要渐变参数。

light_par_save_proc()

为了避免频繁的写 flash，灯的状态改变 3 秒后才会存储到 flash。

light_dim_set_hw()

设置 PWM 的输出，当节点收到 G_LEVEL_SET，G_LEVEL_SET_NOACK 等命令后，会执行。

Idx 和 idx2 对应 light_res_hw[idx][idx2].

idx:对应 light count 的索引值，比如，一个蓝牙模组上有两个色温灯模组时。

Idx2:灯珠的索引值。

light_dim_refresh ()

当灯的状态发生变化后，调用该函数来刷新 PWM 输出值。在该函数里面，用户能直接获取具体的 lightness，色温值等，客户可以根据这个值按自己的调光算法来输出 PWM 值。

默认的算法是，把标准的 lightness，色温值换算为 0—100 的刻度，然后从 rgb_lumen_map[101]查表获取到对应的 PWM 输出值。

get_light_pub_list()

当灯的状态发生变化后，查询所有需要 publish 的 status。

temp_to_temp100()

把 800—20000 的色温值转换为 0—100 的刻度。

temp100_to_temp()

把 0—100 的刻度转换为 800—20000 的色温值

light_g_level_set_idx_with_trans()

```
typedef struct{
 s32 step_1p32768;     // (1 / 32768 level unit)
 u32 remain_t_ms;   // unit ms: max 26bit: 38400*1000ms
 u16 delay_ms;        // unit ms
     s16 present;            // all value transfer into level, include CT.
     s16 present_1p32768;// (1 / 32768 level unit)
     s16 target;
}st_transition_t;
```

收到 level set/lightness set 等命令时，如果有渐变需求，则通过该函数来设置渐变参数。st_transition_t 里面的 “1p32768”表示把一个 level 刻度划分为 32768 份,也就是该值的 unit 是 1/32768，此处的做法主要是避免浮点运算。 提高运算效率。

step_1p32768:表示渐变过程中，每经过 LIGHT_ADJUST_INTERVAL 时间，变化的量。

remain_t_ms:命令中 remain 的值转换为 ms

delay_ms:命令中 delay_ms 的值转换为 ms

present:渐变过程中，level 的实时值。

present_1p32768:present 的余数部分，单位是 1/32768 level 刻度。

target:目标 level。

light_transition_proc()

渐变轮询处理函数。当渐变结束，即当前 level 达到目标 level 时，如果该渐变是由 scene load 触发，则会调用 scene_target_complete_check(i); 来标记场景已生效。

另外，当渐变结束时，还要检查并发送 publish status。

led_onoff_gpio()

当处于 deep retention sleep 或者 deep sleep 模式时，如果需要在 sleep 期间保持输出，只能通过设置上下拉的方式来输出。因为此时 pwm 已停止工作，gpio function，gpio output enable，gpio output 等数字 register 已经不工作。只有设置上下拉等 analog register 在工作。

proc_led()

led 指示灯事件轮询处理函数。

rf_link_light_event_callback ()

led 指示灯注册事件，用该方式，闪灯是在 main_loop 中执行，不影响其他事件的处理。

当处于 LPN 模式时，因为要进入 deep 模式，不能一直轮询执行 proc_led()，导致闪烁很慢，不符合预期。所以，目前暂时的做法是，需要 led 指示时，需要设置较快的闪烁参数，然后一直轮询 proc_led()，当闪烁结束后，再执行其他函数。一般的 LPN 产品是不需要 LED 的，只是开发时使用。如果有需要，还需要另行处理。

主要闪灯事件介绍:

LGT_CMD_SET_MESH_INFO(LGT_CMD_PROV_SUC_EVE):表示 light 节点端，provision 成功后的闪灯事件.

LGT_CMD_FRIEND_SHIP_OK:LPN 和 friend 建立 friend ship 完成，LPN 会产生该事件。

LGT_CMD_SET_SUBSCRIPTION:收到修改 subscription address 的 message.

LGT_CMD_BLE_ADV:BLE 连接断开事件。仅 Debug 模式使用，默认不打开。

LGT_CMD_BLE_CONN:BLE 连接成功事件。仅 Debug 模式使用，默认不打开。

LGT_CMD_SWITCH_POWERON:switch 上电闪灯一次。

LGT_CMD_SWITCH_PROVISION:switch 进入配网模式。

LGT_CMD_SWITCH_CMD:switch 发送按键命令。

PROV_START_LED_CMD:gateway 开始对节点发起 provision flow。

PROV_END_LED_CMD:gateway 对节点发起 provision flow 已完成。

LGT_CMD_DUAL_MODE_MESH:dual mode 模式执行切换 mode。

show_ota_result()

OTA 结束后的闪灯指示处理函数。

show_factory_reset()

执行恢复出厂设置后的闪灯指示处理函数。

如何导入客户自己的调光算法:

默认的调光算法:参考前面 light_dim_refresh()的介绍。

客户需要修改调光算法的话，修改 light_dim_refresh() 即可，即在里面获取到标准的值，然后调用自己的调光算法即可:

Lightness:

```
st_transition_t *p_trans = P_ST_TRANS(idx, ST_TRANS_LIGHTNESS);
u16 lightness = get_lightness_from_level(p_trans->present);
```

CT value:

```
u16 temp = light_ctl_temp_prensent_get(idx);
```

HSL(RGB) value:

参考 light_dim_refresh()里面的获取方法，根据调光算法需求，取得 HSL.h/HSL.s/HSL.l 或者 RGB.r/ RGB.g/ RGB.b。

Onoff:onoff 是通过 lightness 来决定的。

10. Provisioner（Gateway）工程介绍

10.1 provisioner 的功能介绍

10.1.1 adv-bearer 和 gatt-bearer

Provisioning 是将一个未分配的节点加入到 mesh 网络中的，provisioner 能够将未分配的节点加入到 mesh 中成为 mesh 网络中的节点。在 provision 的过程中主要分配 network key、IV index、unicast adr，其中 network key 和 IV index 是决定是否是处于同一个网络的关键参数，unicast adr 为在网络中分配的地址。provisioner 可以通过网络参数和 unicast adr 去访问对应的节点进行相应的控制，例如开关灯，调节亮度之类的。

provision 的链接信道分为两种:一种是通过广播包实现 adv-bearer 信道上的通信，本章节主要介绍 adv-bearer 部分的内容。另外一种是采用 BLE 连接实现，通过 gatt-bearer 实现，gatt-bearer 的实现方式详见 7.2 章节中的 BLE 连接和加灯，以及 SIG_mesh 对应 Android 和 iOS 的 app 可实现对应的功能。

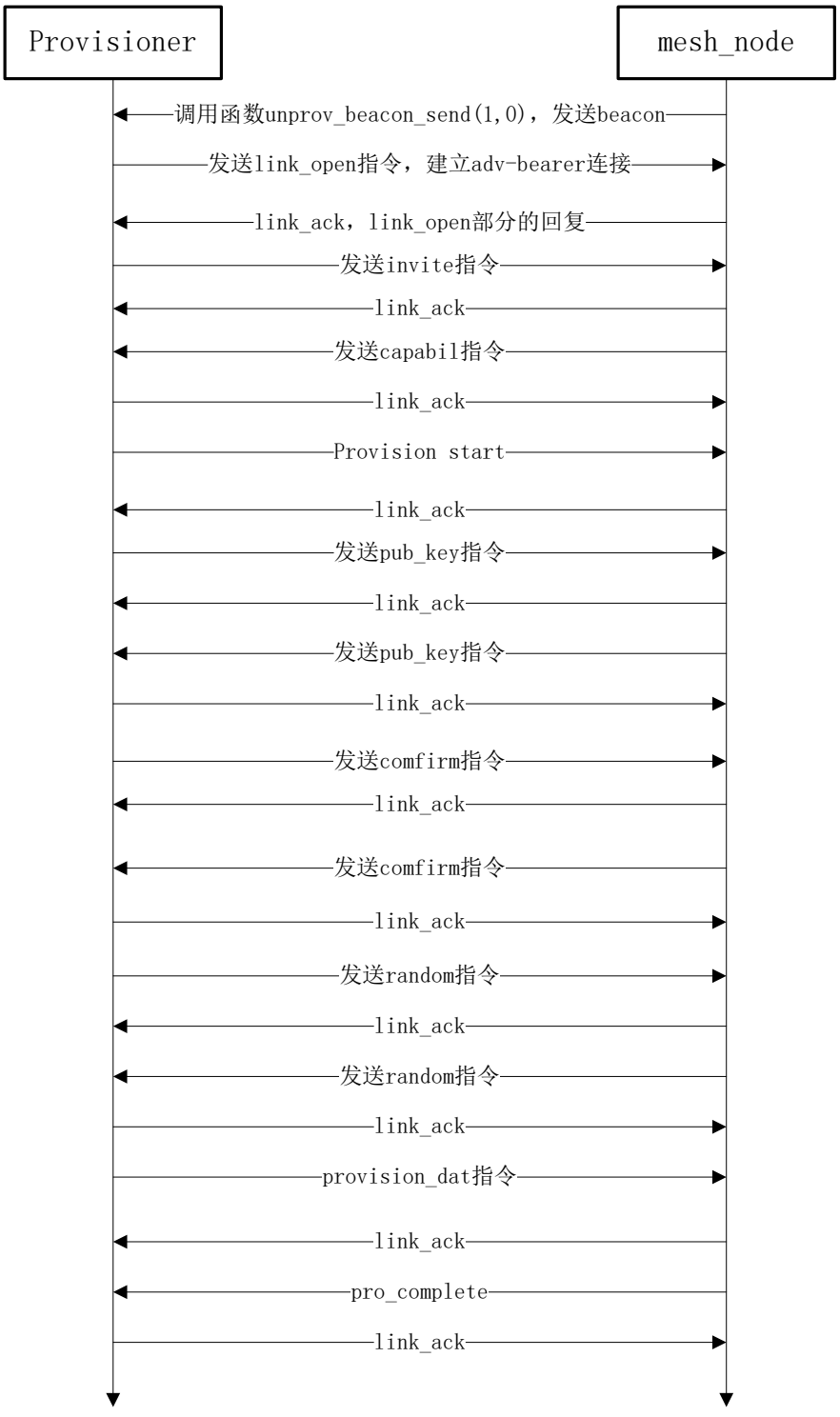
10.2 provisioner 的原理

10.2.1 provisioner 的命令交互

provisioner 通过 adv-bearer 将 unprov node（未配对节点）加入到网络中，采用蓝牙的 37,38,39 信道和 unprov node 进行通信，通过相互交互 random，key 之类的参数（详见 Mesh_v1.0.pdf 中 5.3）来交换网络参数和地址的分配最终达到将 unprov node 加入到网络中。

10.2.2 adv provisioner 的时序图

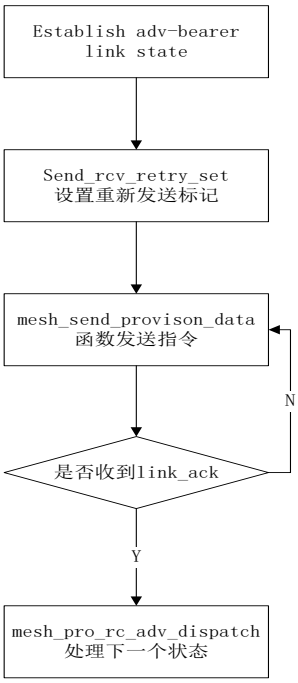
图 10-1 adv provisioner 时序图



在 provision_dat 指令中包含 network key、IV index、unicast adr 这 3 个网络参数，实现组网的功能。

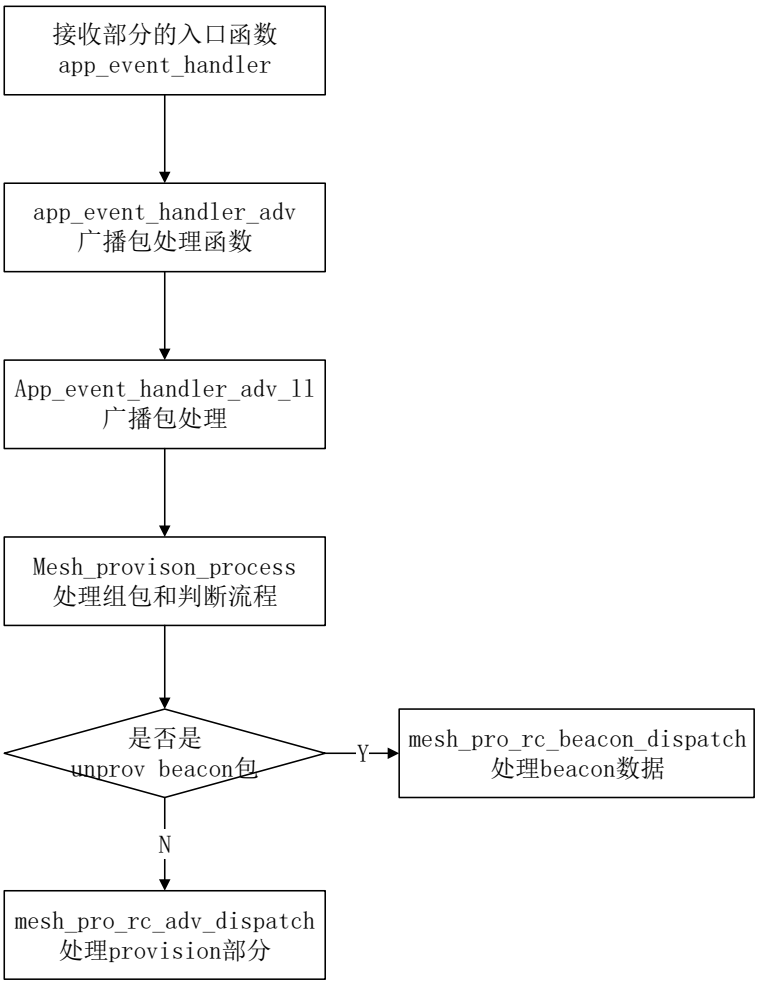
adv-provision 的函数发包部分的函数关系调用图:

图 10-2 adv-provision 发包部分的函数关系调用图



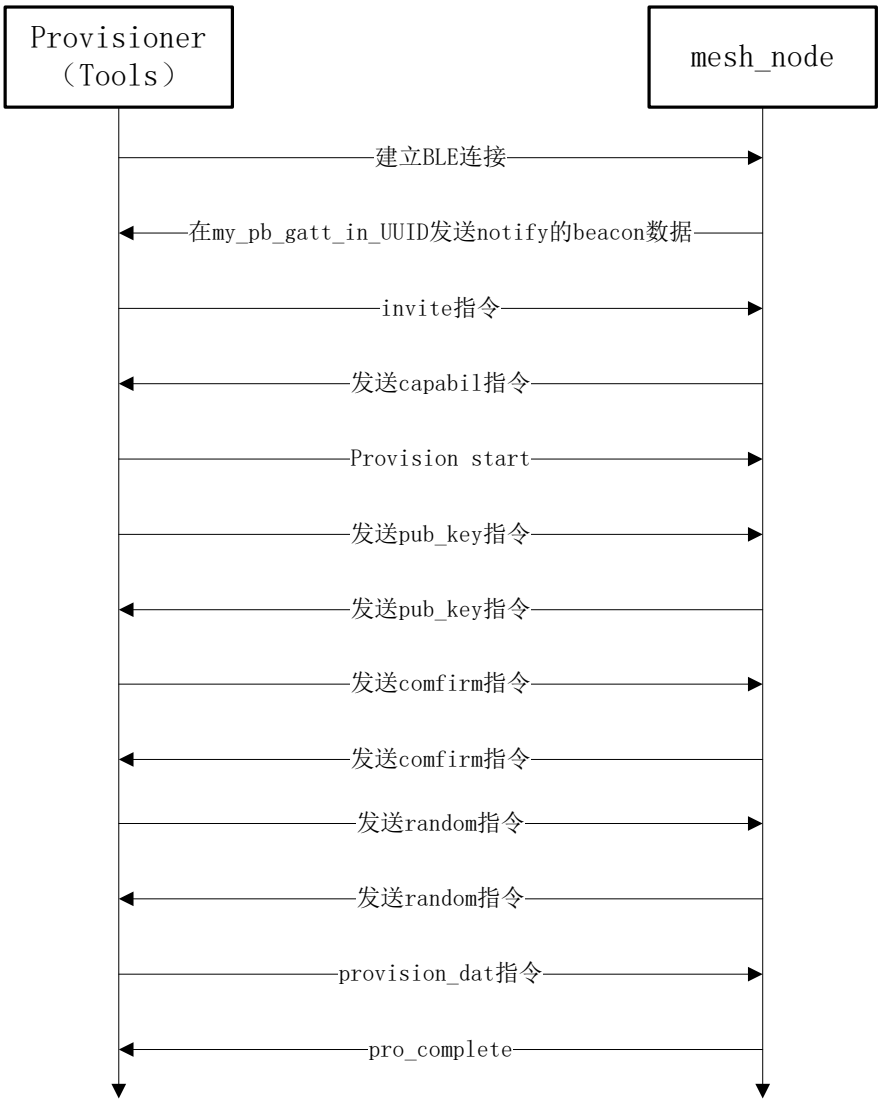
adv-provision 的函数收包部分的函数关系调用图:

图 10-3 adv-provision 收包部分的函数关系调用图



10.2.3 gatt provisioner 的时序图

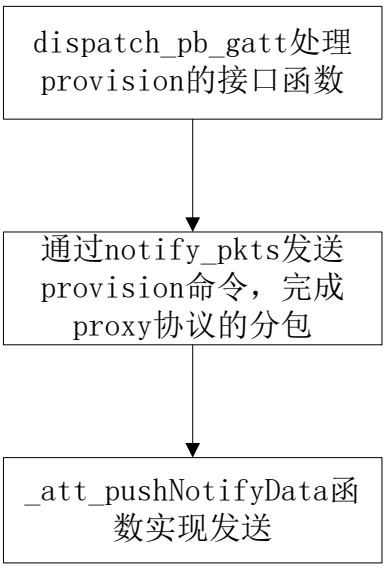
图 10-4 gatt provisioner 时序图



通过 gatt-provision 的方式能够更加快速的实现 provision 的功能，其中 int pb_gatt_Write (void *p)为 gatt_provision 部分的入口函数。

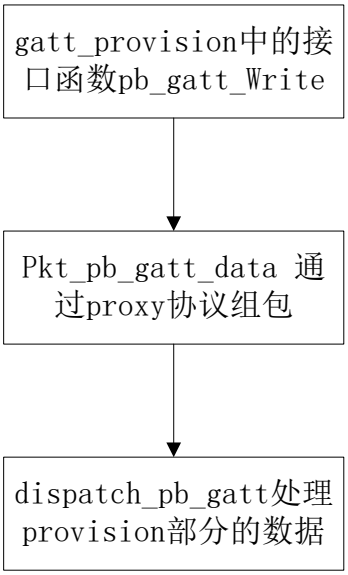
gatt_provision 的发包函数入口:

图 10-5 gatt_provision 发包函数入口



gatt_provision 的收包函数入口:

图 10-6 gatt_provision 收包函数入口



10.3 app.c 文件介绍

目前在 provisioner 工程中需要定制修改的内容暂时仅仅只限于 app.c 文件，需要对 app.c 中的内容做定制化的修改。

广播包以及广播响应包的定制

参考《8258 MESH 工程介绍》。

fifo 部分配置

参考《8258 MESH 工程介绍》。

HCI（USB/UART）上报数据

my_fifo_push_hci_tx_fifo (u8 *p, u16 n, u8 *head, u8 head_len)

p:指向所发数据地址

n:数据长度

head:标识头

head_len:标识头长度（没有则填 0）

调用 my_fifo_push_hci_tx_fifo (u8 *p, u16 n, u8 *head, u8 head_len)往 hci_tx_fifo 送数据，在发送回调函数里会把 hci_tx_fifo 的数据送出去。回调函数已在 user_init 里注册:

- ✧ UART: blc_register_hci_handler (blc_rx_from_uart, blc_hci_tx_to_uart);
- ✧ USB: blc_register_hci_handler (app_hci_cmd_from_usb, blc_hci_tx_to_usb);

app_event_handler ()

参考《8258 MESH 工程介绍》。

main_loop ()

参考《8258 MESH 工程介绍》。

user_init()

参考《8258 MESH 工程介绍》。

proc_ui()

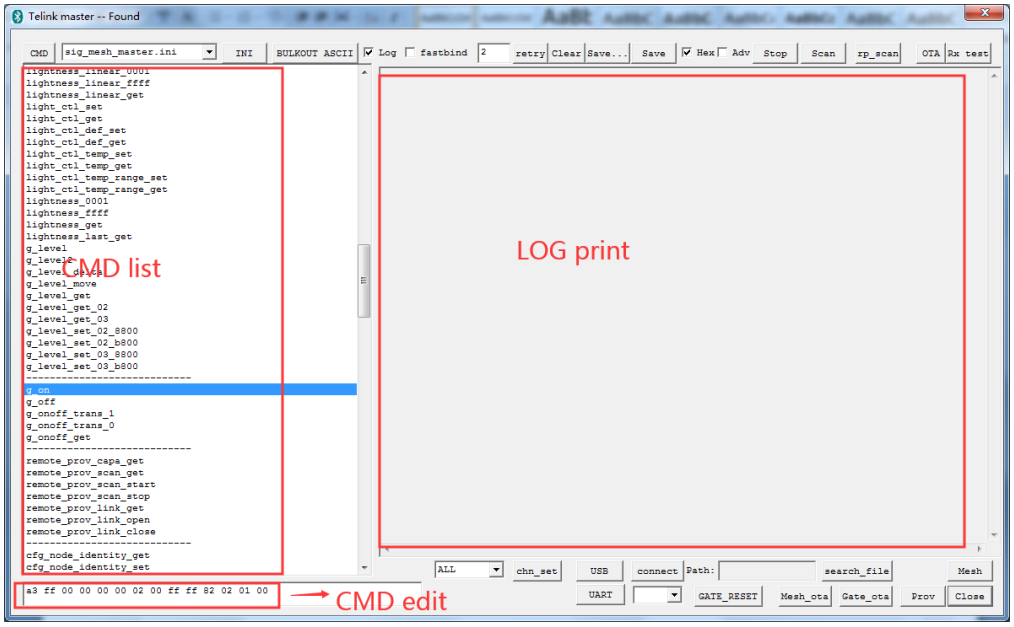
在 proc_ui 函数中配置每 40ms 扫描下 IO 口检测 IO 的变化。最终是通过这个接口 access_cmd_onoff 函数来发送开关灯的指令。每间隔 100ms 发送一条开关灯的指令，按 SW1 发送开灯的指令，SW0 发送关灯的指令。

10.4 网关（provisioner）操作和接口

对于网关，协议栈是在网关端运行的。入网的节点信息都存在网关 flash 地址为 FLASH_ADR_VC_NODE_INFO（0x3f000）的位置，由于一个 sector 为 4K，目前最多保存 200 个节点信息。如果超过 200 个节点无法继续将对应的节点加入到网络中，需调用 HCI_GATEWAY_CMD_CLEAR_NODE_INFO，即“e9 ff 06”清除 gateway 存贮的节点信息。

10.4.1 SIG_MESH_TOOL ini 文件格式说明

Gateway 的操作会用到“SIG_MESH_TOOL”，用户可以通过点击界面上的 button 控制，或双击主界面左边的命令列表，或编辑 edit 控件后按 enter 键下发命令。网关收到后会在 app_hci_cmd_from_usb_handle 的相应分支作处理。



ini 文件里命令格式分为 SIG model 和 vendor model。

10.4.2 SIG model 命令格式，g_all_on 为例:

1		2	3	4	5	6	7	8	9	10	11
Flag		nk_idx	ak_idx	reliable retry cnt	eliable rsp_max	dst	op	Onoff	TID	Transition time	Delay
e8	ff	0000	0000	00	00	ff ff	82 02	1	0	nc	nc

前 2 个字节是标识符，为 telink 私有化定义，是通信的包头识别，gateway 为 0xE8FF，app(包含手机 app 及 kma dongle 上位机)为 0xA3FF

后面参数结构如下:

```
typedef struct{
    u16 nk_idx;         //netkey index
    u16 ak_idx;         //app_key index
    u8 retry_cnt;       // 应用层 retry 的次数，未收到 rsp_max 个回复时应用层会 retry

    u8 rsp_max;         // 表示期望收到的回复个数
    u16 adr_dst;   // 目标地址
    u8 op;          //操作码，op code 的第一个 byte
    u8 par[MESH_CMD_ACCESS_LEN_MAX]; //op code 的剩余 byte 以及 parameter
}mesh_bulk_cmd_par_t;
```

其中，retry_cnt 在 VC tool 中，如果设置为 0，则表示使用"retry"控件的值，默认是 2，

。VC tool 会自动把 INI 数据中 retry_cnt 的值改为 2。如果设置为 0xFF，则表示不需要 retry，即 VC tool 会自动把 INI 数据中 retry_cnt 的值改为 0。

另外，nk_idx，ak_idx， rsp_max 更详细的介绍，请参考本文档 set_material_tx_cmd() 的介绍。

Note:TID，如果值为 0 时，由协议栈来管理和维护。如果 TID 的值不为 0，则使用这个值作为 TID，这样客户就可以自己维护 TID。

10.4.3 Vendor modle 命令格式.以 CMD-vendor_on 为例:

1		2	3	4	5	6	7	8	9	10	11
Flag		nk_idx	ak_idx	reliable retry cnt	eliable rsp_max	dst	op	op_rsp	tid_pos	para[0]	para[1]
e3	ff	0000	0000	02	00	ff ff	C2 1102	c4	2	1	0

与 SIG model 不同的是，多了 op_rsp 和 tid_pos，这 2 个参数只是给上位机本身使用的，是一个伪参数，并不会下发到设备端。

op_rsp:设置 op 对应的 response opcode(不需要包含 vendor id)。

tid_pos:设置 tid 的位置（值为 0 表示没有 tid 字段，1 表示 tid 在 para[0]位置，2 表示在 para[1]位置...），op_rsp 和 tid_pos 的设置要与设备端统一。

增加这两个参数的目的，是因为 provisioner 端需要支持多种 vendor id 的 op code，以及 vendor op 随时会添加，所以无法把这些信息都编译进程序中。所以需要通过 ini 命令的方式添加到里面。

10.4.4节点的烧录

烧录 2 个 8258 dongle:1 个 8258 provisioner 节点(8258_mesh_gw.bin)、1 个 dongle 节点(8258_mesh.bin)。

烧录步骤，请参考“4. 调试工具操作说明”的“下载固件”章节。

10.4.5 provisioner 加灯

网关配合 SIG_mesh_tool 工具的加灯流程和相应命令格式（16 进制表示）。

1) provisioner dongle 插到 USB 口。打开“SIG_MESH_TOOL”选择 tl_node_gateway.ini，标题栏显示 Found 说明找到 gateway 设备（provisioner）。

工具会自动获取网关的 uuid 和 mac 地址，发送的命令格式为:

HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_GET_UUID_MAC。

即 e9 ff + 10

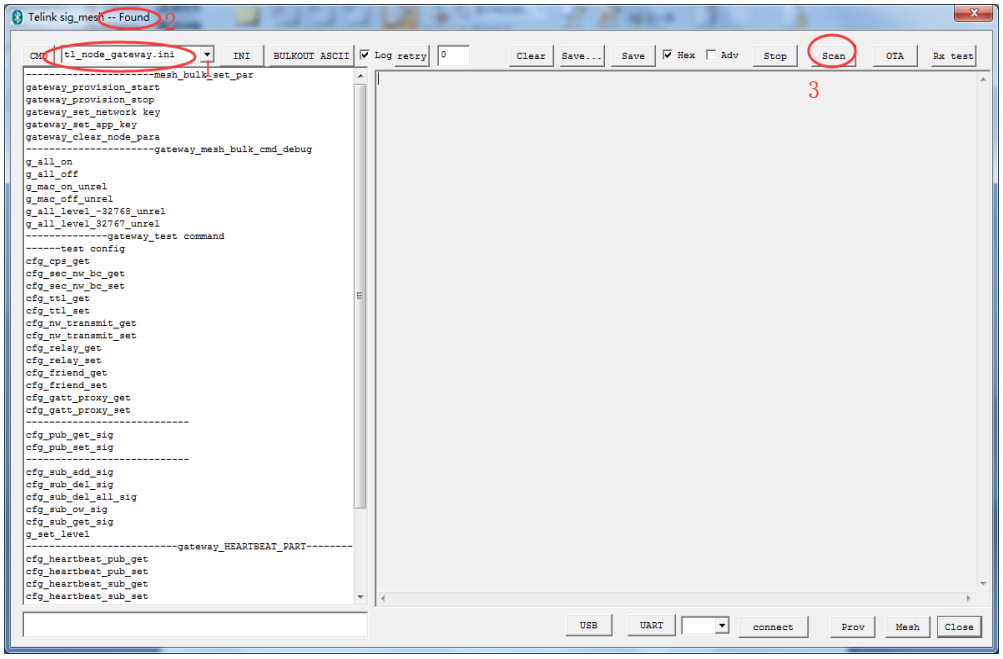
网关收到后会上报 uuid 和 mac，格式为:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_UUID+uuid(16 bytes)+mac(6 bytes), 即:91+99+uuid(16 bytes)+mac(6 bytes)。

2) 8258 dongle 上电，然后点击 scan 搜索设备。

scan 控件对应的命令为 HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_START:e9 ff + 00

stop 控件对应的命令为 HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_STOP:e9 ff + 01



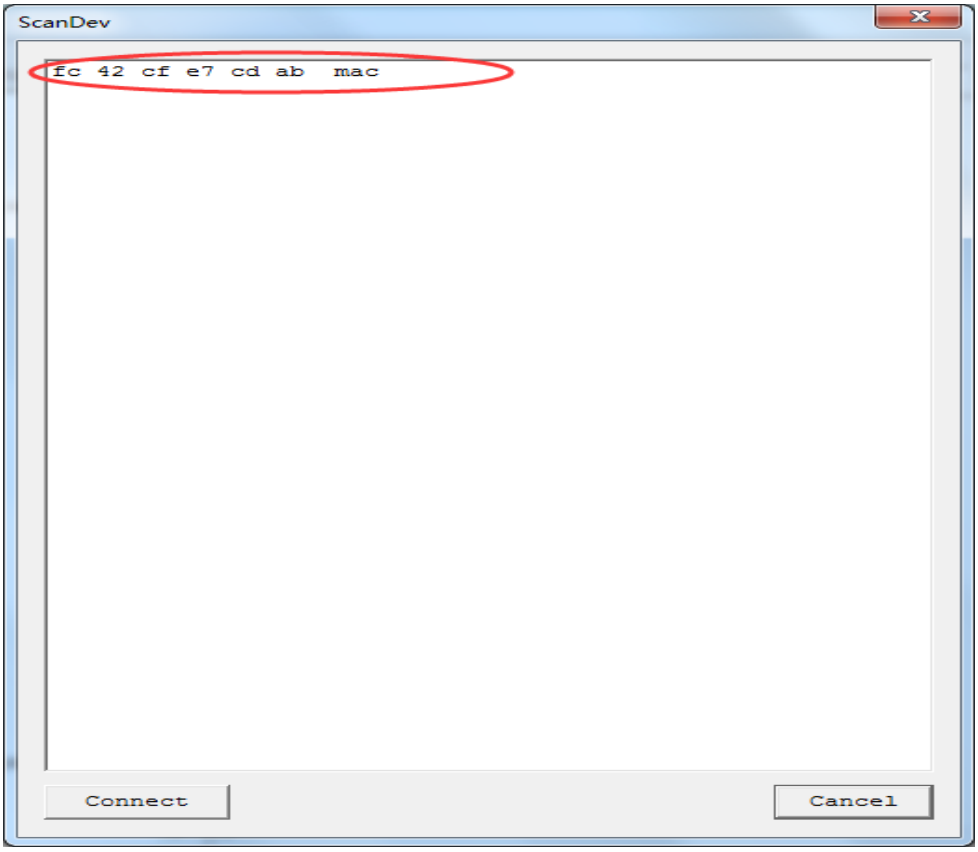
3) 点击 satrt 控件后网关会把收到的 unprovision beacon 上报上来，上报格式为:

TSCRIPT_GATEWAY_DIR_RSP+ HCI_GATEWAY_CMD_UPDATE_MAC+unprovision beacon.即:91+88+mac(6 bytes)+ unprovision beacon.

然后在设备列表显示出来，双击选择需要 provision 的设备，双击对应的命令为:

HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_ADV_FILTER+6 字节 mac 地址。

即:e9 ff + 08 + mac(6 bytes)。



4) 点击 Prov 进入 provision 界面

a.Provision 控件对应的命令为:HCI_GATEWAY_CMD_GET_PRO_SELF_STS。

即“e9 ff 0c”。

b.网关收到此命令后后会返回是否已有配置信息和网关本身的 element 个数，相应的命令格式为:TSCRIPT_GATEWAY_DIR_RSP+ HCI_GATEWAY_CMD_PRO_STS_RSP+provision_flag+ pro_net_info。

即:91 8b + provision_flag+ pro_net_info。pro_net_info 为 25 字节的 provision data。格式如下:

```
typedef struct{
 u8   net_work_key[16];     //network key
 u16   key_index;            //network key index
 union{
       mesh_ctl_fri_update_flag_t prov_flags;
       u8   flags;   // iv update flag
 };
 u8   iv_index[4];   // iv index
 u16   unicast_address;
}provison_net_info_str;
```

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_ELE_CNT+total element:即 91+8c+ total element.

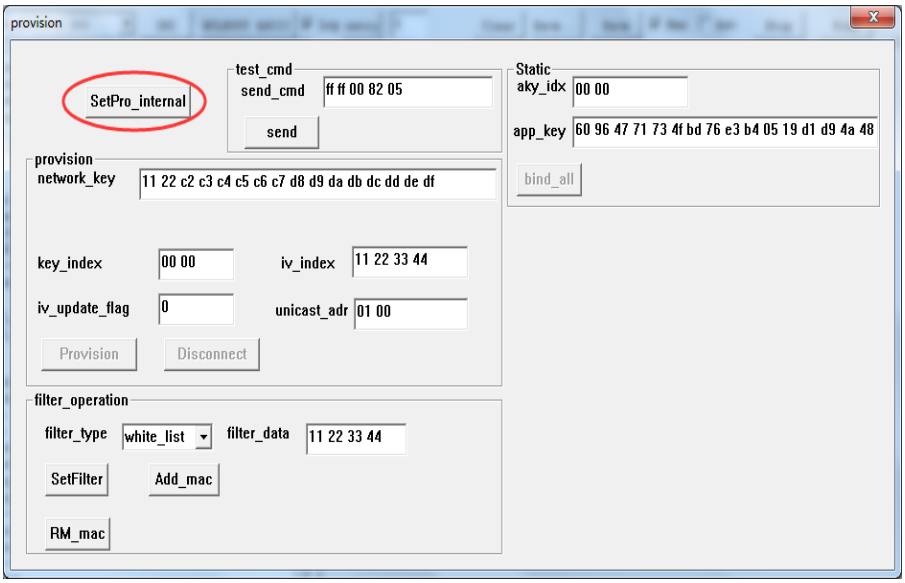
c. 如果 provision flag 标记为 0，说明网关没有配置信息，SetPro Interna 控件使能，在 provision 界面填好 pro_net_info 相关参数后点击 SetPro_interval 设置网关配置信息。对应命令为:HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_PRO_PARA + pro_net_info。

即:e9 ff + 09 + pro_net_info。

HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_PRO_PARA+unicast address +device key:

即:e9 ff + 0d +gateway address+device key

如果 provision flag 标记为 1，说明 gateway 已有配置信息，SetPro Interna 控件禁止，跳过此步骤即可。



5) 点击 Provision 按钮进行 provision。provision 过程中主界面的会打印相关 log，provision 成功后 bind_all 按钮使能。

a.Provision 控件对应的命令为:HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_NODE_PARA+ pro_net_info 。

即 e9 ff+0a+ ro_net_info。

b.Provison 过程中会上报分配给设备的地址，格式为:

TSCRIPT_GATEWAY_DIR_RSP +HCI_GATEWAY_RSP_UNICAST+unicast addr，

即:91 80+unicast address。

c.provision 完成后会上报设备的 information。格式为:

TSCRIPT_GATEWAY_DIR_RSP+ HCI_GATEWAY_CMD_SEND_NODE_INFO+ VC_node_info_t。

即：91+8d+ VC_node_info_t。

VC_node_info_t 定义如下：

```
typedef struct{
    u16 node_adr;      // primary address
    u8 element_cnt;
    u8 rsv;
    u8 dev_key[16];
}VC_node_info_t;
```

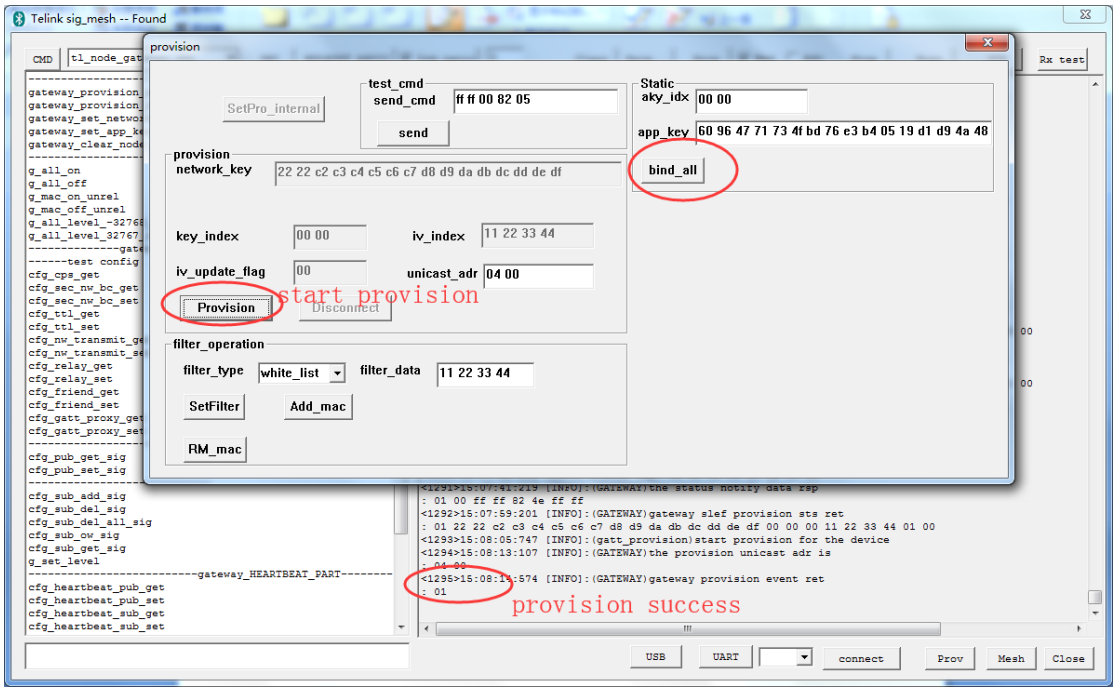
d.provision 完成后会上报设备 provision 状态。格式为:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_PROVISION_EVT+ gateway_prov_event_t。

即:91 + 89 + gateway_prov_event_t.

gateway_prov_event_t 定义如下:

```
typedef struct{
    u8 eve;//1 表示成功
    u16 adr;
    u8 mac[6];
    u8 uuid[16];
}gateway_prov_event_t;
```



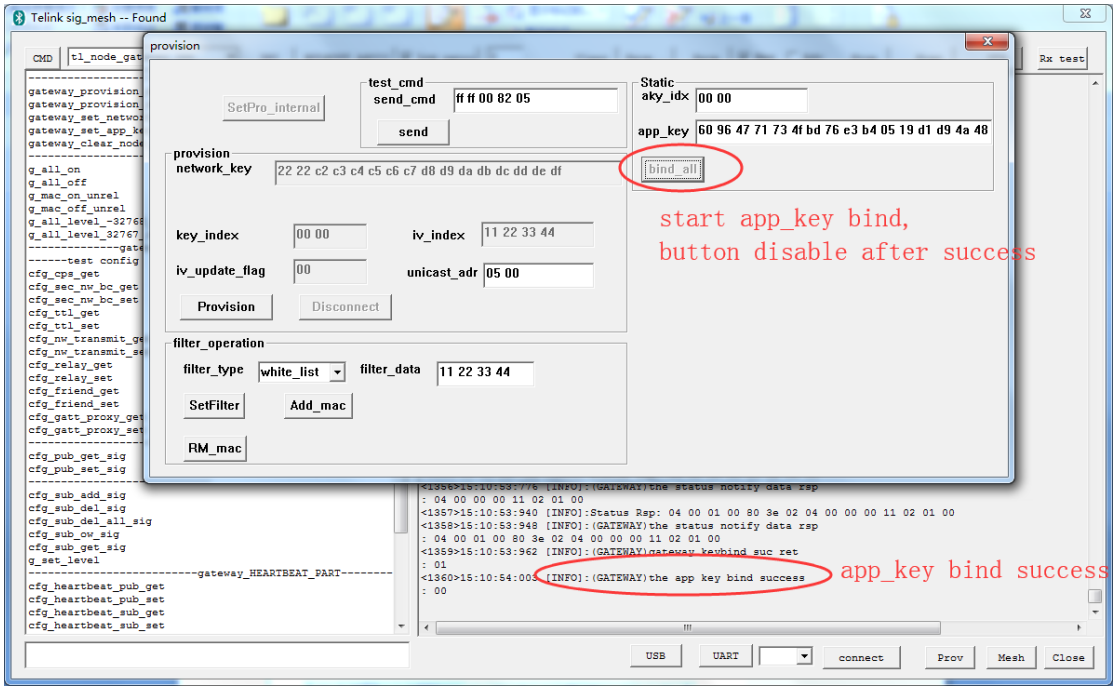
10.4.6 绑定 app_key

Provision 完成后，还需要为 model 绑定 app_key。点击 bind_all 为 model 绑定 app_key。

a.bind_all 对应的命令为:HCI_CMD_GATEWAY_CTL+
HCI_GATEWAY_CMD_START_KEYBIND + fast_bind +

+app_key index(2 byte)+app_key(16 bytes)。

即:e9 ff + 0b + fast_bind + app_key index(2 byte)+app_key(16 bytes)。



fast_bind 为 1 时:网关只会下发 appkey add，被 provision 的设备需打开默认绑定功能（PROVISION_FLOW_SIMPLE_EN 设置为 1）。

fast_bind 为 0 时:网关默认绑定全部 model id，为了节省时间，用户可选择需要绑定的 model id。网关端打开宏 MD_BIND_WHITE_LIST_EN，需要绑定的 model id 详见 Mesh_common.c 文件中 master_filter_list[]，用户可根据需要自行修改。

b.App_key bind 过程中 gateway 会调用 u8 gateway_model_cmd_rsp(u8 *para,u8 len) 返回绑定 model 的状态信息，格式为:TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE +参数。

即:91 + 81 + appkey bind status

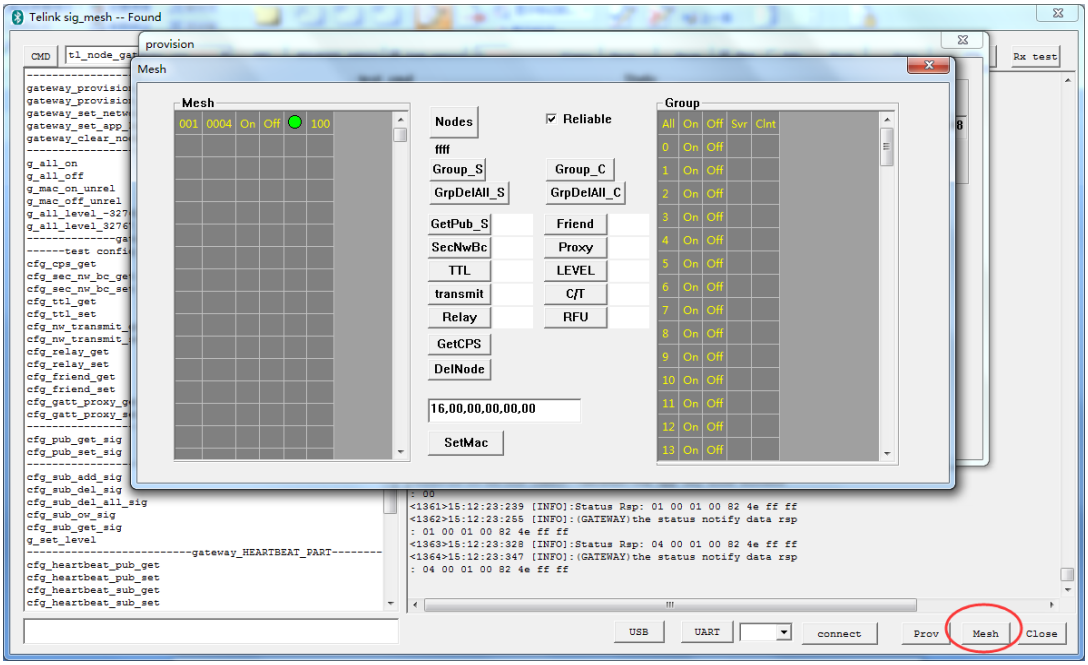
c.App_key bind 完成之后会返回 HCI_GATEWAY_CMD_KEY_BIND_EVT 表示成功还是 time_out。格式为:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_KEY_BIND_EVT +result。

即:91 + 8a + result。（1:success 2:time_out）。

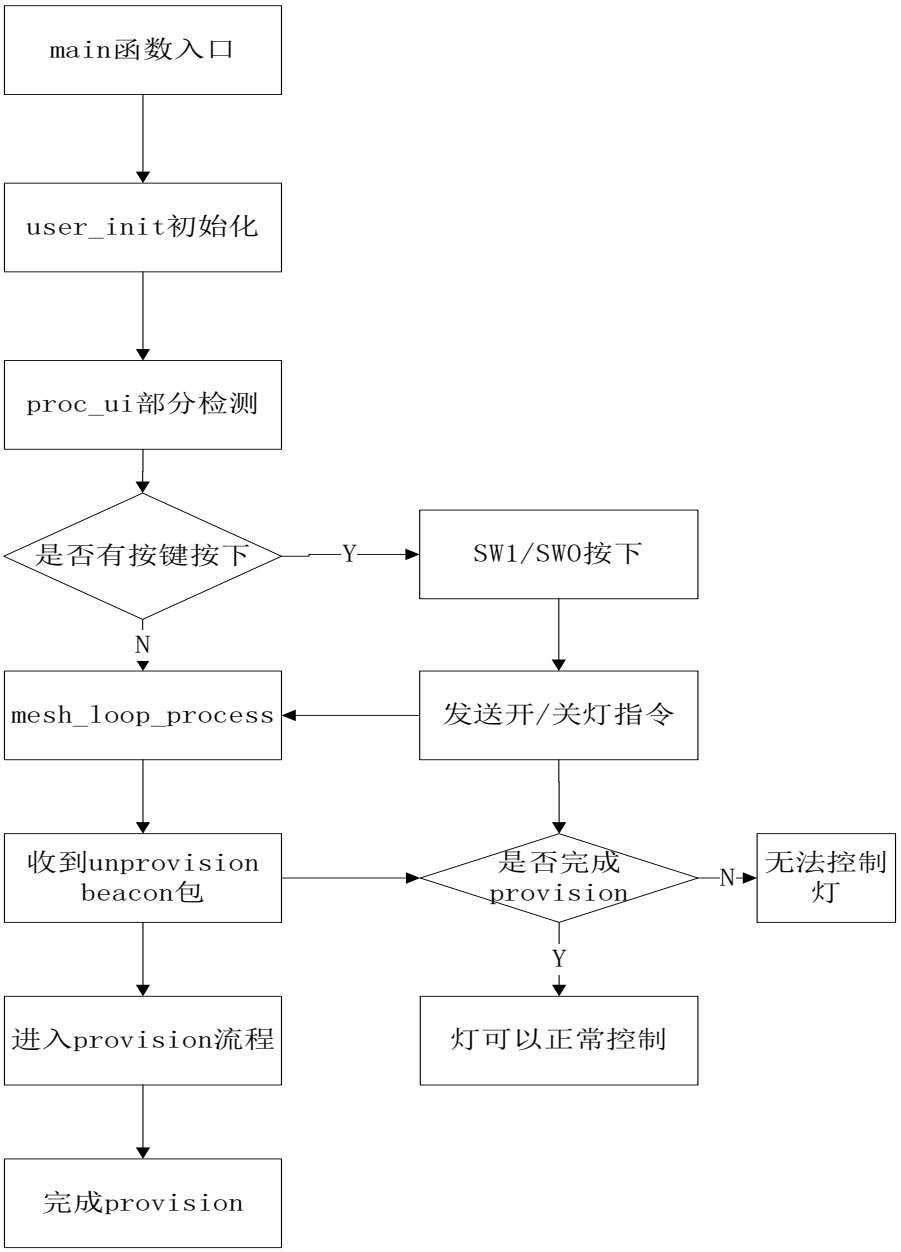
10.4.7 控制灯的开关

绑定 app_key 后点击 Mesh 进入 mesh 界面、或者双击主界面左边的 g_all_on/g_all_off 进行开关灯操作。


Telink sig_mesh -- Found
provision
Mesh
Mesh
001 0004 On Off 100
Nodes
Reliable
ffff
Group_S
Group_C
GrpDelAll_S
GrpDelAll_C
GetPub_S
Friend
SecNwBc
Proxy
TTL
LEVEL
transmit
C/T
Relay
RFU
GetCPS
DelNode
16,00,00,00,00,00
SetMac
Group
All On Off Svr Clnt
0 On Off
1 On Off
2 On Off
3 On Off
4 On Off
5 On Off
6 On Off
7 On Off
8 On Off
9 On Off
10 On Off
11 On Off
12 On Off
13 On Off
CMD
Rx test
USB
UART
connect
Prov
Mesh
Close

10.4.8 provisioner 的控制流程图

图 10-7 provisioner 控制流程图



11. Mesh LPN 工程介绍

11.1 LPN 节点的概念和实现方式介绍

注:本节图片均来自于 SIG MESH spec。

11.1.1 LPN 和 friend 的概念

低功耗（Low-Power）特性:能够以明显较低的接收端占空比在 mesh 网络中运行。通过将无线电接收器启用时间最小化可实现节点功耗的降低，只有在绝对必要时才启动接收器。低功耗节点（LPN，low power node）通过与好友节点（FN，friend node）建立友谊（friendship）关系来实现这一点。

对于 LPN 节点，只能和一个 FN 建立友谊；对于 FN 节点，可以和多个 LPN 节点建立友谊。

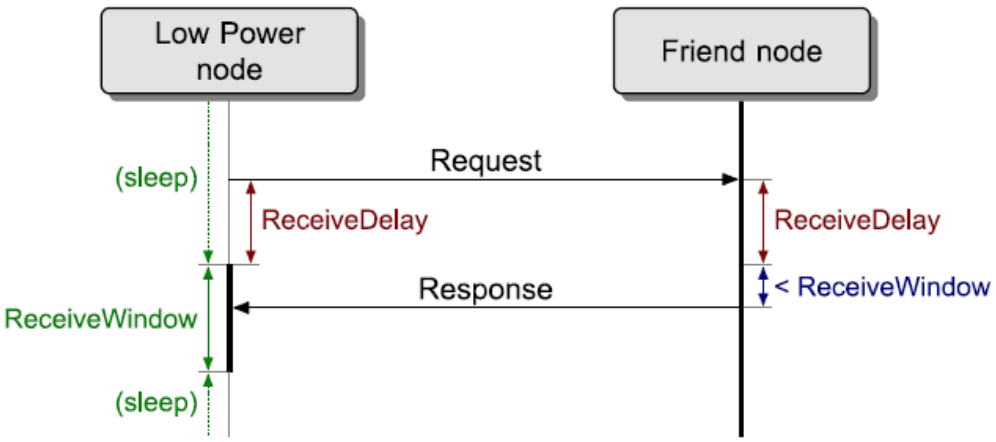
当建立 friendship 后，低功耗节点会以一个比较长的周期对好友节点进行轮询（Poll），查看是否有新消息，如果有，则获取该消息。

好友（Friend）特性:通过存储发往 LPN 的消息，仅在 LPN 明确发出请求时才进行转发来帮助 LPN 运行的能力。

11.1.2 友谊（Friendship）参数

低功耗节点需要找到好友节点，与其建立"友谊"关系。所涉及的流程称为"友谊建立"。我们稍后会探讨此流程。在此之前，先介绍一些有关对 LPN 行为进行管理的关键参数，这些参数被设定于友谊建立过程中。

1) ReceiveDelay 是从 LPN 向好友节点发送请求，到其开始收听响应中间经过的时间。这让好友节点有时间做好响应的准备，并将响应发回。
2) ReceiveWindow 是 LPN 用于收听响应的时间。下图描述了涉及 ReceiveDelay 和 ReceiveWindow 的时序。



3) PollTimeout 设定了 LPN 发送给其好友节点的两个连续请求之间可能经过的最长时间。如果在 PollTimeout 计时器到时之前，好友节点未能收到 LPN 的请求，则友谊关系将被终止。

11.1.3 “友谊”建立

蓝牙 mesh 网络中“友谊”的建立，需要经过以下步骤。

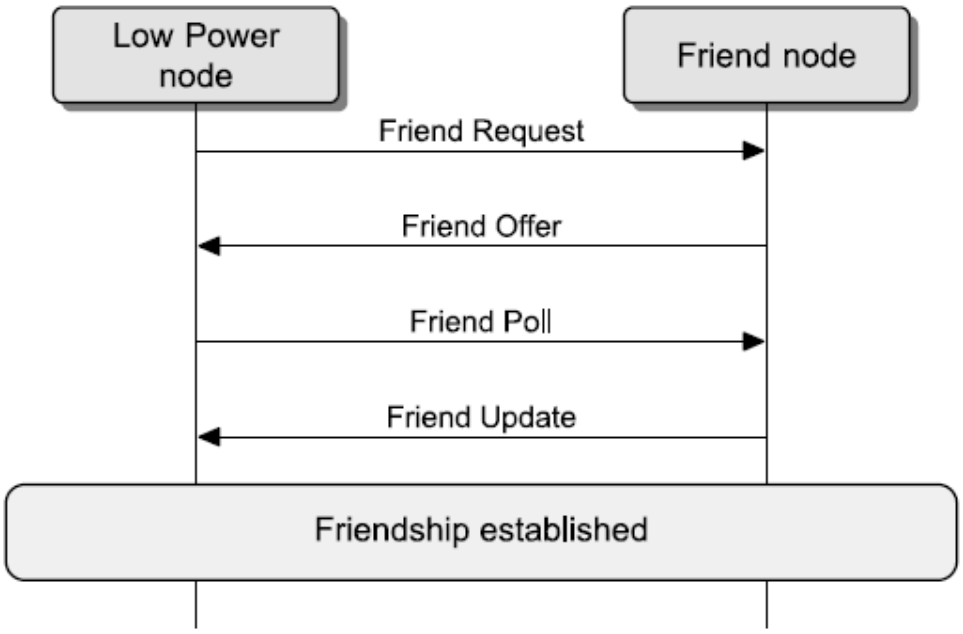
Step 1　LPN 发布一个“好友请求”（Friend Request）消息。该消息不会被中继，因此只有处于直接无线电范围内的好友节点才能处理该消息。不具有“好友”特性的节点会将消息丢弃。“好友请求”消息包括 LPN 的 ReceiveDelay、ReceiveWindow 和 PollTimeout 参数。

Step 2　附近的好友节点若支持“好友请求”消息中特定的要求，将准备一个“Friend Offer”消息，并将其发送回 LPN。该消息包括各种参数，包括支持的 ReceiveWindow 大小、可用的消息队列大小、可用的订阅列表（Subscription List）大小、以及由好友节点测量的 RSSI 值。

Step 3　LPN 接收到“Friend Offer”消息时，通过当前节点采用的算法来选择合适的好友节点。该算法可能会考虑到各种各样的情况。某些设备可能会优先考虑 ReceiveWindow 大小，以尽可能减少功耗；而有些设备则可能会更加关注 RSSI 值，以确保能够与好友节点保持高质量的链路。所采用的精确算法由产品开发者决定。

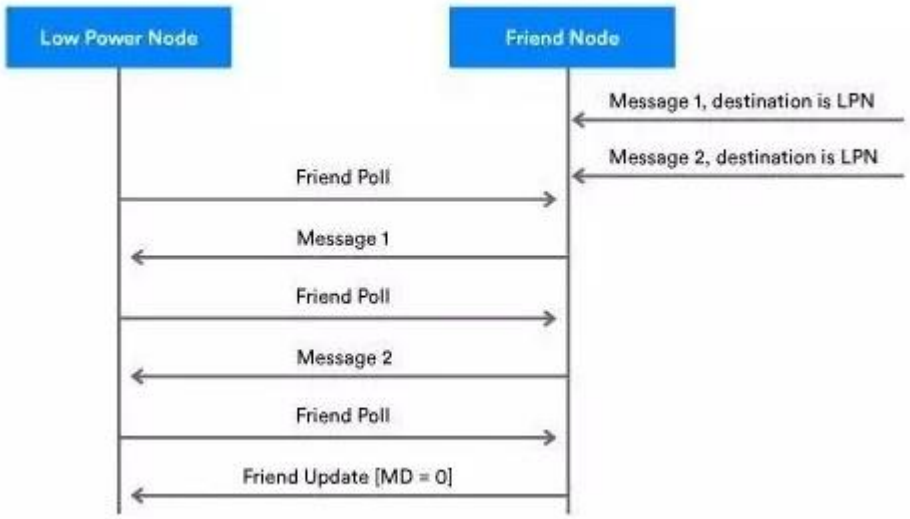
Step 4　选择好友节点之后，LPN 将向好友节点发送一个“Friend Poll”轮询消息。

Step 5　从 LPN 收到“好友轮询”（Friend Poll）消息后，好友节点会回复一个“Friend Update”更新消息，完成“好友”建立流程并提供安全参数。此时“友谊”得以建立。



11.1.4 友谊（Friendship）消息传送

友谊建立之后，好友节点将 LPN 的所有消息存储在“好友队列”（Friend Queue）中，这些消息就是我们所说的“被存储的消息”。下图所示为好友节点和关联 LPN 之间的消息交换。



当好友节点收到一个寻址到该节点的 LPN 的消息时，好友节点会缓冲此消息，将其存储在称为"好友队列"的区域中。在上图中，我们可以看到，好友节点为 LPN 存储了消息 1 和 2。

LPN 会周期性地启用其收发器（transceiver），并向好友节点发送 "好友轮询" 消息，询问是否存储有任何为其缓冲的消息。

好友节点会先将一个被存储的消息发回至 LPN 作为对"好友轮询"（Friend Poll）的响应。

在每次接收到来自好友节点的消息之后，LPN 将会继续发送"好友轮询"消息，直到收到一条"MD（MD =更多数据）"字段设置为 0 的"好友更新"消息为止。这意味着已经没有为 LPN 缓冲的更多消息了。此时，LPN 停止对好友节点的轮询。

11.1.5安全性

主安全资料（Master Security Material）：由网络密钥（NetKey）派生，可被同一网络中的其他节点使用。使用主安全资料加密的消息可被同一网络中的任何节点解码。

好友安全资料（Friend Security Material）：由网络密钥（NetKey）、以及由低功耗节点（LPN）和好友节点生成的额外计数器号码派生而来。使用好友安全资料加密的消息只能由处理该消息的好友节点和 LPN 解码。

使用好友安全材料加密的相应友谊消息：好友轮询（Friend Poll）、好友更新（Friend Update）、好友订阅列表（Friend Subscription List）。

使用主安全资料加密的相应友谊消息：好友清除（Friend Clear）、好友清除确认（Friend Clear Confirm）。

其它从 LPN 发送至好友节点的非控制消息将根据应用设置对应 model publish 参数里面的 credential_flag，来确定是通过主安全资料或好友安全资料进行加密。credential_flag 默认值是 0，也就是使用主安全资料。

11.1.6"友谊"终止

如果在 PollTimeout 计时结束之前，好友节点未收到"好友轮询"、"好友订阅列表添加"或"好友订阅列表删除"消息，则友谊终止。

LPN 可以通过将"好友清除"消息发送给好友节点，以启动友谊终止程序，"友谊"就会被好友节点终止。

11.2 LPN 使用演示

11.2.1硬件准备

此演示基于 GATT master dongle 模式进行，APP 和 gateway 模式的操作步骤和 GATT master dongle 模式类似。注意，gateway 模式时，gateway 节点本身也支持 friend 功能。

一个 8269 GATT master dongle 和 2 个 8258 mesh dongle(一个烧录 8258_mesh.bin, 默认支持 Friend 功能。另一个烧录 8258_mesh_LPN.bin，即 LPN 节点）

LPN 默认支持 generic ONOFF, generic Level,不使能 lightness 和 light CT。

11.2.2 测试步骤

以下提到的和时间相关的宏，客户可以按实际需求更改。

Step 1　mesh friend 节点(FN)上电，并用 SIG_MESH_TOOL 对其进行组网。
Step 2　未配网的 LPN 节点上电，此时 LPN 处于唤醒状态。

LPN 节点上电后，红色 LED 会处于 ON 的状态。未配网状态下，不进入 sleep 状态，目的是可以支持 GATT 组网和 ADV 组网。在此状态下，如果 1 分钟(LPN_SCAN_PROVISION_START_TIMEOUT_MS)之后，还未开始组网流程，则进入 deep sleep 状态，不再发送 ADV，并关闭 LED，目的是节省功耗，避免长时间处于工作高功耗状态。如果已经进入 deep sleep，则需要按下 mesh_lpn_key_map[ ] 里面定义的 SW1, SW2 来唤醒，唤醒后才会重新开始发送 ADV 和开始组网流程。

Step 3　对处于唤醒状态的未配网 LPN 进行组网和 bind key 流程。

Bind key 成功，3 秒钟后(LPN_START_REQUEST_AFTER_BIND_MS)，LPN 会自动重启，然后把 lpn_provision_ok 置为 1,并进入 LPN 模式,开始每 2 秒(FRI_REQ_TIMEOUT_MS)发送一次 friend request 命令。

组网成功后,为了减少处理无效的 network message,降低功耗,LPN 只接收从 FN 通过 friend ship 发过来的信息。如果要接收普通 network message，把 mesh_lpn_rx_master_key 初始化为 1 即可。

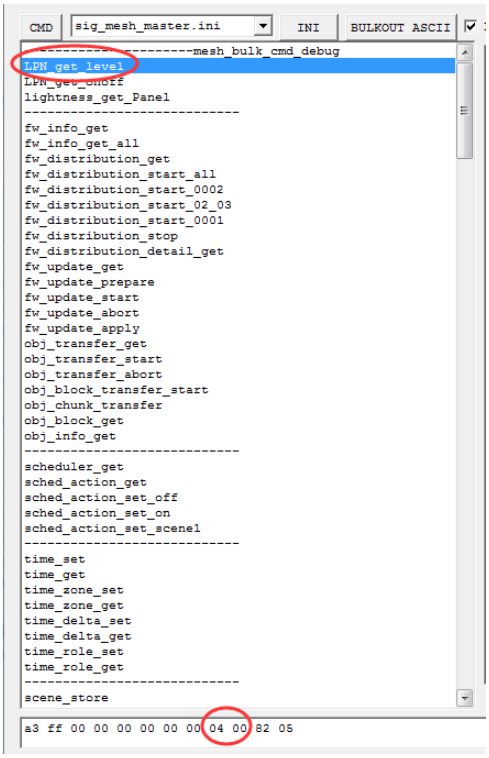
Step 4　当有 FN 存在时，会自动建立 friend ship，只有建立成功(红灯闪烁 3 次)，LPN 才能接收 message。

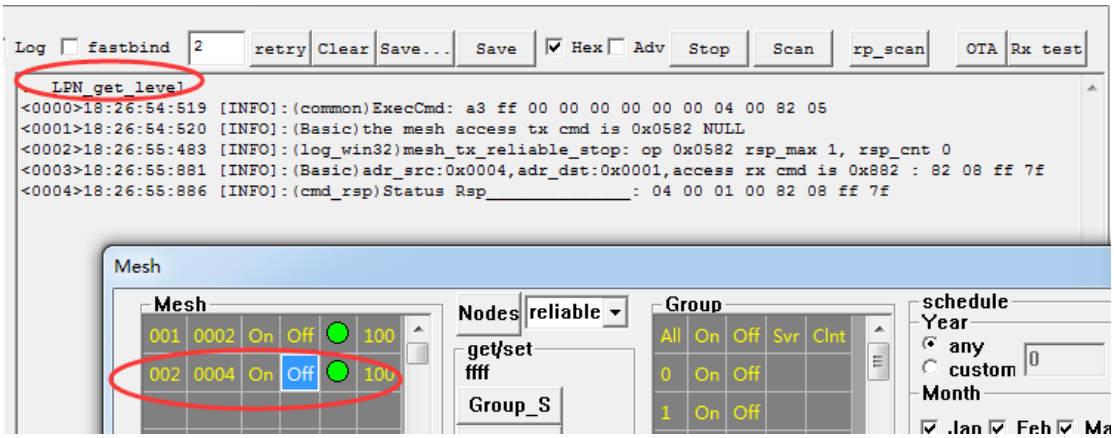
FN 收到 friend request 后，会自动回复 friend offer，然后按 flow 建立 friendship，如果建立成功，会回调 friend_ship_establish_ok_cb_lpn()，并在里面执行红灯会闪烁 3 次(LGT_CMD_FRIEND_SHIP_OK)。然后开始以 2 秒周期(FRI_POLL_INTERVAL_MS)发送 friend POLL。当 FN 收到 POLL 后，如果有需要发送给 LPN 的 cache message，则会把这个 message 发送给 LPN。Cache message(network PDU)的默认最大个数是 4 个(2 ^ FN_CACHE_SIZE_LOG).

假如暂时没有 FN 响应 POLL，LPN 会一直按 2 秒周期发送 friend request。

Step 5　"mesh"窗口显示 LPN 节点及 ONOFF 操作。

首先打开"mesh"窗口，点击"LPN_get_level" INI 命令，下图左下角出现 a3 ff 00 00 00 00 00 00 04 00 82 05，其中 04 00 是 LPN 的 unicast address(如不正确，需要修改)，按回车键，发送该命令，收到 LPN 的 level status 回复后，在 UI 中就会显示 LPN 节点，然后可以对 LPN 发起 ONOFF 操作等，如下图。





因为 LPN 是不接收 destination address 为 0xffff 的 message，所以需要按 unicast 的方式发送命令。如果已经对 LPN 配置了组号，也可以按 group 的方式发送命令。

另外留意，点击了“Nodes”按钮或者重新打开“Mesh”窗口后，VC tool 会把所有的节点置为离线状态，然后发送 lightness get all(destination address 为 0xffff)的命令，重新获取节点状态，但是没有单独给 LPN 节点按 unicast destination address 发送 get 命令，所以需要手动再点“LPN_get_level”命令或者点击 mesh 窗口里面的 ON/OFF 命令才会显示在线状态，否则为离线状态。

Step 6　group 操作，和普通节点的操作一样，请参考“4.5.2　分组控制(即 subscription 的功能演示)”

Step 7　LPN 检测到 FN 断电，自动重新寻找新的 FN。

当 FN 断电， LPN retry 8 次(FRI_POLL_RETRY_MAX)POLL 命令，其中 POLL 的间隔是 170ms(FRI_REC_DELAY_MS + FRI_REC_WIN_MS)，如果 LPN 还是没有收到 FN 的回复，则认为 FN 已断电，会断开 friend ship 并回调 friend_ship_disconnect_cb_lpn()，如果需要执行 led 闪烁操作，也请在该回调函数里面添加。然后重新发送 friend request 寻找新的 friend 节点。

Step 8　目前 demo SDK 1 个 friend node 默认最多同时和两个 LPN 建立 friend ship，如需修改，设置 MAX_LPN_NUM 即可，最大值是 16。

当 LPN 断电后，FN 会检测 10 秒钟(LPN_POLL_TIMEOUT_100MS)，如果还未收到 POLL 命令，则认为节点已断电，此时 FN 会清除该 LPN 的信息。

Step 9　按键发送 ALL ON/OFF 命令。

当 LPN 处于 retention sleep 状态的时候，按下按键 SW2(MESH_LPN_CMD_KEY)唤醒 LPN，然后通过 suspend_handle_next_poll_interval() -> mesh_lpn_wakeup_key_io_get（）检测到该按键，然后执行 test_cmd_wakeup_lpn()函数交替发送 ALL ON/OFF 命令。LPN 主动发送 access layer 命令默认使用主安全资料加密。

Step 10　恢复出厂设置

长按按键 SW1 (MESH_LPN_FACTORY_RESET_KEY) 3 秒钟(LONG_PRESS_TRIGGER_MS) 触发 factory reset.

11.3 app.c 文件介绍

广播包以及广播响应包的定制

参考《8258 MESH 工程介绍》。

fifo 部分配置

参考《8258 MESH 工程介绍》。

app_event_handler ()

参考《8258 MESH 工程介绍》。

main_loop ()

参考《8258 MESH 工程介绍》。

user_init()

参考《8258 MESH 工程介绍》。

proc_ui()

该函数主要是做些 UI 方面的处理，比如 button 检测函数，以及对应的测试代码。目前暂时没有功能。

test_cmd_wakeup_lpn()

当命令按键按下时(目前 demo dongle 是 SW2)，会唤醒程序，并执行 test_cmd_wakeup_lpn()，test_cmd_wakeup_lpn()里面会发出 ON OFF 命令。命令发送完成后进入 sleep。该功能仅做 demo 演示用。

11.4 mesh_lpn.c 文件介绍

mesh_lpn_proc_suspend ()

该函数处理 LPN 的睡眠处理函数。lpn_sleep.op 表示因为什么命令或者事件需要进入睡眠，并在唤醒后处理该事件的后续动作。

比如当 lpn_sleep.op 等于 CMD_CTL_POLL，表示刚刚发送了 POLL message，然后需要进入时间为 receive delay 的 retention sleep，然后唤醒进入 receive window,如下图：

```
void mesh_lpn_proc_suspend ()
{
    ......

    if(lpn_sleep.op && is_lpn_support_and_en){
        ......
        u8 r = irq_disable();
        if((CMD_ST_SLEEP == lpn_sleep.op) || (CMD_ST_NORMAL_UNSEG == lpn_sleep.op)){
            ......
        }else if(CMD_ST_NORMAL_SEG == lpn_sleep.op){      // no need to enter retention deep sleep
            ......
        }else{
            // wake up for rx after send ctrl op, such as POLL,...
            // can't use HANDLE_RETENTION_DEEP_PRE now, because it is from lpn_quick_tx_and_suspe
            suspend_handle_wakeup_rx(HANDLE_SUSPEND_NORMAL);
        }
        irq_restore(r);
    }
}
```

其他的自定义事件有：

CMD_ST_SLEEP：一个 friend ship 的交互周期完成后，进入 2 秒(friend request interval 或者 poll interval)的 retention sleep 状态，然后唤醒进入下一个周期的交互。

CMD_ST_NORMAL_UNSEG：主动发送 access layer 的 unsegment message 后，进入和 CMD_ST_SLEEP 一样的处理方式。

CMD_ST_NORMAL_SEG：主动发送 access layer 的 segment message 后，进入睡眠，等待一段时间后，唤醒，然后接收 block ack，以及处理 segment message 的丢包补发机制等。

CMD_ST_POLL_MD：当发送 POLL 后，收到 FN 回复 MD(more data)为 1，则睡眠 100ms(FRI_POLL_DELAY_FOR_MD_MS)后，唤醒，并继续发送 POLL 接收剩余的 message。

suspend_handle_next_poll_interval()

执行下一个周期的 POLL 的处理函数，里面的有 HANDLE_RETENTION_DEEP_PRE 和 HANDLE_RETENTION_DEEP_AFTER 分支，是因为 825x 在执行 retention sleep 之后，MCU 不是继续往后执行，而是会从 main()→user_init_deepRetn()→suspend_handle_next_poll_interval()执行后续的处理。

mesh_feature_set_lpn()

LPN 的一些可配置参数的初始化。主要是配置 LPN_POLL_TIMEOUT_100MS， 默认是 10 秒。

12. SWITCH 工程介绍

12.1 Switch 的功能介绍

switch 的功能主要是为了在 mesh 节点中加入遥控器的功能。使得 switch 能够控制 mesh 网络中的节点。switch 的节点需要先被 provisioner 加入网络中，然后可以通过 switch 上的按键来控制 mesh 网络中的节点。

12.2 Switch 的原理介绍

switch 是控制 mesh 的遥控器，由于遥控器是低功耗节点，必须触发配对模式才能进行 provision，加入网络中之后，能够控制 mesh 中的节点。

12.3 app.c 文件介绍

广播包以及广播响应包的定制

参考《8258 MESH 工程介绍》。

fifo 部分配置

参考《8258 MESH 工程介绍》。

app_event_handler ()

参考《8258 MESH 工程介绍》。

main_loop ()

参考《8258 MESH 工程介绍》。

user_init()

参考《8258 MESH 工程介绍》。

proc_ui ()

每隔 4ms 扫描一次按键部分。mesh_proc_keyboard 为处理按键部分的接口函数。当 keycode 为 RC_KEY_A_ON 时，switch 发送 all_on（打开网络中所有的灯）的指令。当 keycode 为 RC_KEY_A_OFF 时，switch 发送 all_off（关闭网络里所有的灯）指令。

proc_led()

首先通过 cfg_led_event 配置 LED 闪烁的频率和时间，如 cfg_led_event(LED_EVENT_FLASH_1HZ_4S)表示配置 LED 以 1Hz 频率闪烁 4s。

然后通过 proc_led 控制 LED 闪烁部分的处理。

mesh_switch_init()

mesh_switch_init 中有两部分的设置：

上面那部分是 switch 部分的唤醒 IO 部分的代码设置，以及打开唤醒使能标记位。

下面那部分是 LED 的 IO 部分的设置，默认将 led 管脚设置为 GPIO 模式，100kohm 下拉电阻，并上电闪烁 4 下。

proc_rc_ui_suspend()

休眠函数部分的处理函数为 proc_rc_ui_suspend()。

休眠部分的处理目前设定为广播状态下，如果不发包直接进入 deep 状态，按键唤醒后，发完包之后继续进入 deep 状态。触发配对模式之后 30s 之后进入 deep 状态，链接状态暂时不进入 deep 状态。

休眠部分的处理流程详见本章的 “休眠部分的处理流程图”小节。

kb_scan_key ()

kb_scan_key 为矩阵键盘扫描部分的接口，numlock_status 目前默认为 0，表示全键盘中的 numlock，read_key 为读取的 key 值。

12.4 switch 部分的配置

12.4.1 key table

```
#define    KB_MAP_NORMAL   {\
        {RC_KEY_1_OFF,          RC_KEY_2_OFF,          RC_KEY_1_ON}, \
        {RC_KEY_3_ON,           RC_KEY_3_OFF,          RC_KEY_2_ON}, \
        {RC_KEY_4_ON,           RC_KEY_4_OFF,          RC_KEY_R}, \
        {RC_KEY_A_OFF,          RC_KEY_A_ON,           RC_KEY_UP}, \
        {RC_KEY_L,              RC_KEY_DN,             RC_KEY_M}, }
```

用户可以根据实际驱动脚的 pin 的数量和扫描脚的数量来配置实际的 key_table 部分的内容。驱动脚对应列数，扫描脚对应行数。

12.4.2 配置驱动脚和扫描脚的 IO

```
#define   KB_DRIVE_PINS     {GPIO_PB4,   GPIO_PB5, GPIO_PB6}
#define   KB_SCAN_PINS {GPIO_PE3, GPIO_PE2, GPIO_PE1, GPIO_PE0, GPIO_PD3}
```

根据实际用到的管脚来修改 KB_DRIVE_PINS 和 KB_SCAN_PINS 对应的宏定义。

其次根据对应的 PIN 定制 drive pin 和 scan pin 的 IO 特性：

drive pin 对应的 IO 属性设置：

```
#define PB4_FUNC                            AS_GPIO
#define PB5_FUNC                            AS_GPIO
#define PB6_FUNC                            AS_GPIO
#define PULL_WAKEUP_SRC_PB4             MATRIX_ROW_PULL
#define PULL_WAKEUP_SRC_PB5             MATRIX_ROW_PULL
#define PULL_WAKEUP_SRC_PB6             MATRIX_ROW_PULL
#define PB4_INPUT_ENABLE                    1
#define PB5_INPUT_ENABLE                    1
#define PB6_INPUT_ENABLE                    1
```

scan pin 对应的 IO 属性设置：

```
#define PE3_FUNC        AS_GPIO
```

```
#define PE2_FUNC          AS_GPIO
#define PE1_FUNC          AS_GPIO
#define PE0_FUNC          AS_GPIO
#define PD3_FUNC          AS_GPIO
#define PULL_WAKEUP_SRC_PD3              MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE0              MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE1              MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE2              MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE3              MATRIX_COL_PULL

#define PE3_INPUT_ENABLE          1
#define PE2_INPUT_ENABLE          1
#define PE1_INPUT_ENABLE          1
#define PE0_INPUT_ENABLE          1
#define PD3_INPUT_ENABLE          1
```

假如在驱动管脚部分将 GPIO_PB6 修改为 GPIO_PB7，则需要修改几个部分。

```
1). #define PB6_FUNC     AS_GPIO----->>>#define PB7_FUNC        AS_GPIO
2). #define PULL_WAKEUP_SRC_PB6              MATRIX_ROW_PULL-------->>
    #define PULL_WAKEUP_SRC_PB7              MATRIX_ROW_PULL
3). #define PB6_INPUT_ENABLE                        1   ---------->>
    #define PB7_INPUT_ENABLE                        1
```

12.4.3 switch 开关灯

根据不同的键值发送不同的命令以及做出相应的处理。

参考具体的按键处理程序 mesh_proc_keyboard（）。switch 在加入网络前，无法控制网络中灯的开关。可以通过同时按键 RC_KEY_A_ON 和 RC_KEY_1_OFF 超过 2s，之后触发配对模式，通过 provisioner 将 switch 加入网络，之后就可以通过 RC_KEY_A_ON 和 RC_KEY_A_OFF 来全开/全关网络中的灯。详见本章"switch 操作" 小节。

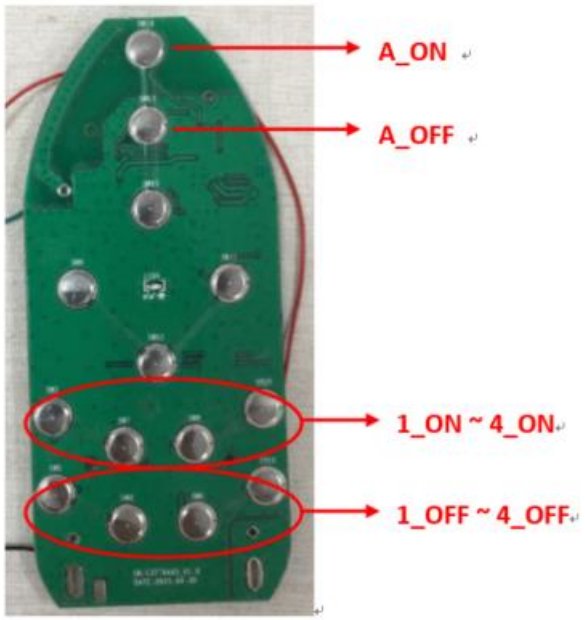
12.5 switch 操作

继续 4.4 中关于 provision 的操作之后，烧录 switch 固件。switch 烧录部分的连接图如下所示：

图 12-1 Switch 烧录链接图

switch 按键的示意图为：

图 12-2 Switch 按键示意图



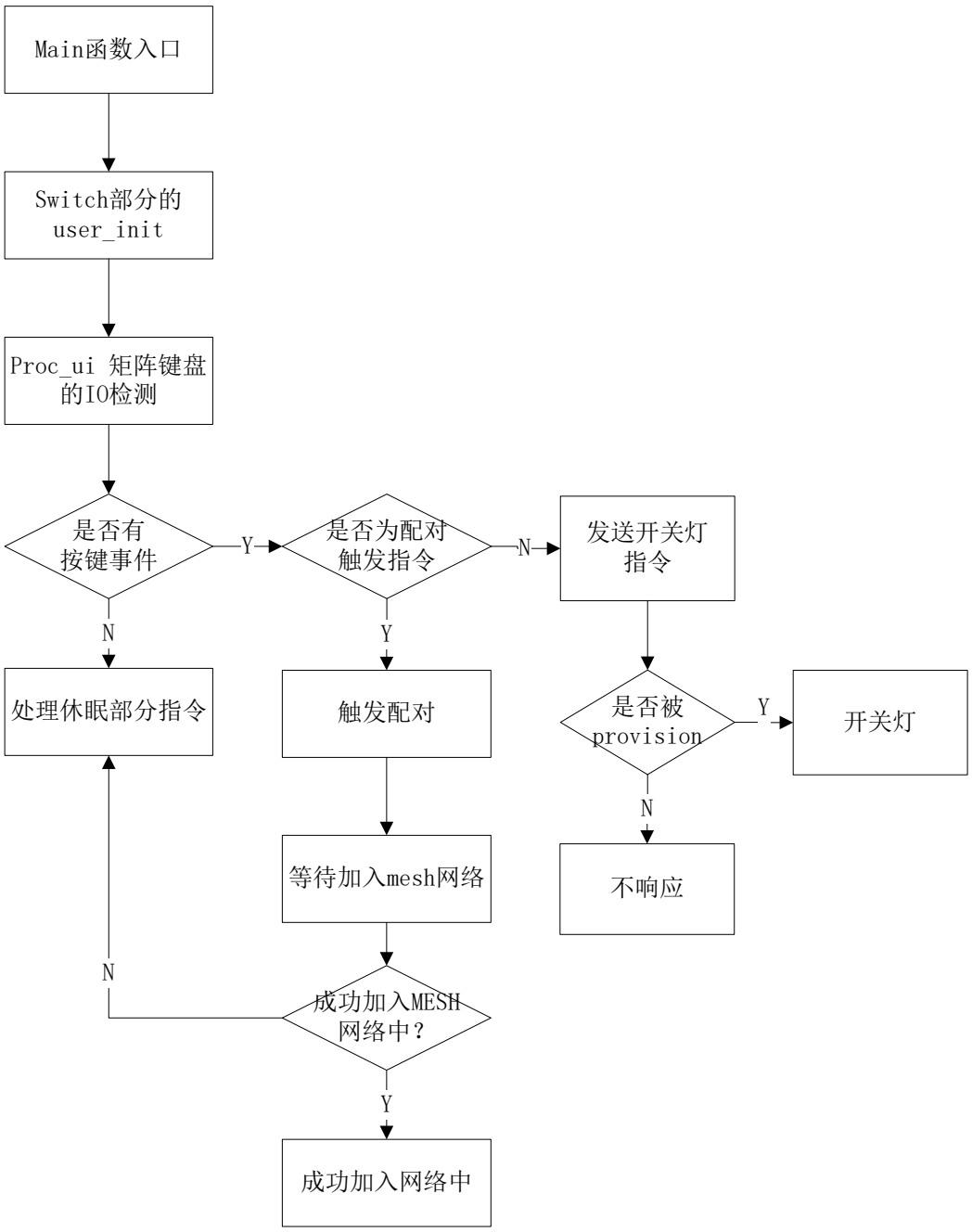
将 switch 节点的设备上电，上电之后由于 switch 属于低功耗节点，所以必须先触发配对模式，使得 switch 通过 provisioner 加入到 mesh 网络中。触发配对模式的方式为同时按压 RC_KEY_A_ON 和 RC_KEY_1_OFF 超过 2s。

switch 上的 led 灯连续闪烁 4 次，表示进入配对模式，provisioner 节点上电，等待 15s 左右，switch 上的 led 连续闪烁 4 次，表示 switch 已经加入到网络。

switch 就可以通过 RC_KEY_A_ON 和 RC_KEY_A_OFF 正常控制网络中的灯节点全开/全关。

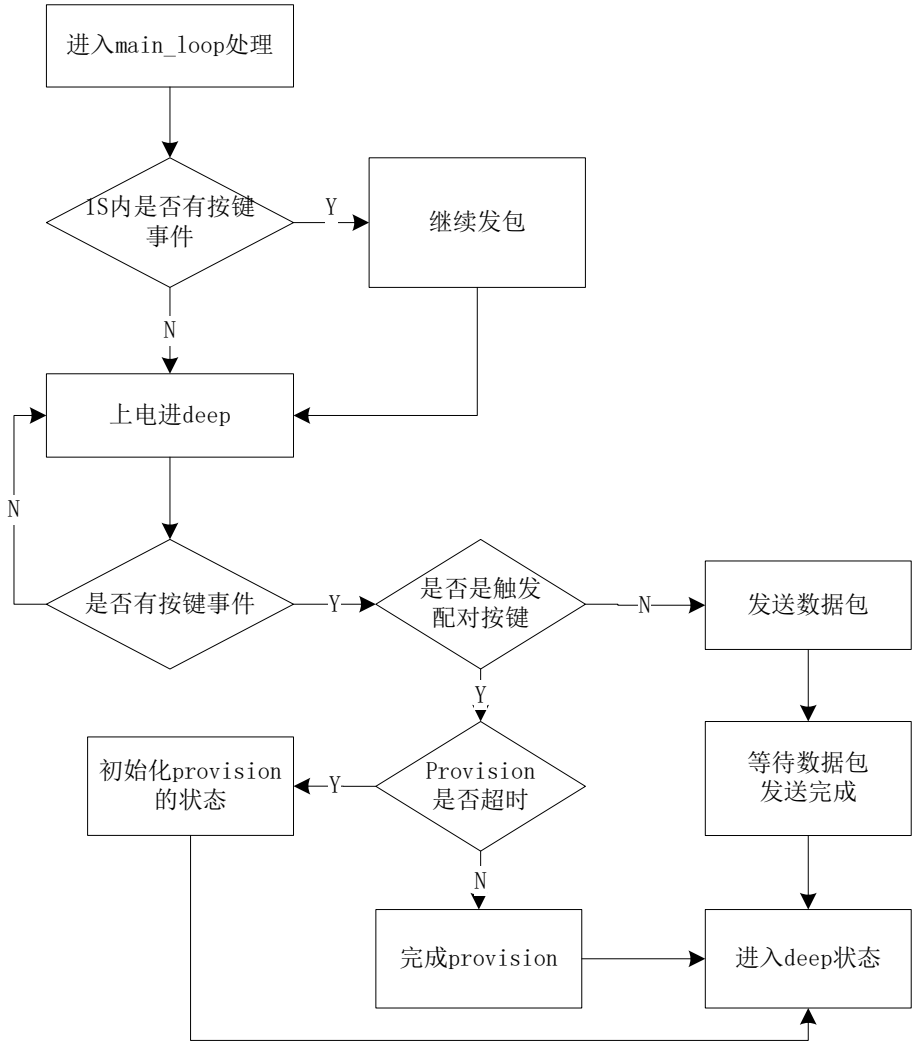
12.6 遥控器的运行流程图

图 12-3 遥控器运行流程图



12.7 休眠处理的流程图

图 12-4 休眠处理流程图



13. 平台接入

当需要接入某个平台的时候，需要配置一些选项，特别是 provision 的方式。通过配置 MESH_USER_DEFINE_MODE 进行选择。

13.1 Normal 模式

13.1.1 No OOB provision 模式

配置方式:

Provision 采用 MESH_NO_OOB 模式。

VENDOR_ID 是 0x0211

测试的时候，直接使用我们的手机 app 或者上位机工具组网即可。

13.1.2 static OOB provision 模式

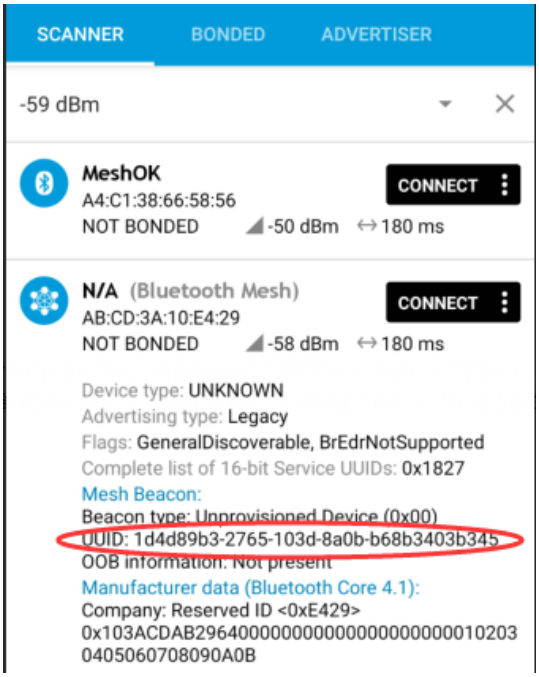
13.1.2.1 Light 节点烧录 Static oob

在烧录 firmware 的时候，直接写 16byte 在 flash 固定位置 FLASH_ADR_STATIC_OOB(例如 0x77800，以实际 code 为准)即可。如果没有烧录(全为 0xff)，则表示使用 no oob 模式。如果需要修改 flash address，修改 FLASH_ADR_STATIC_OOB 这个宏即可。

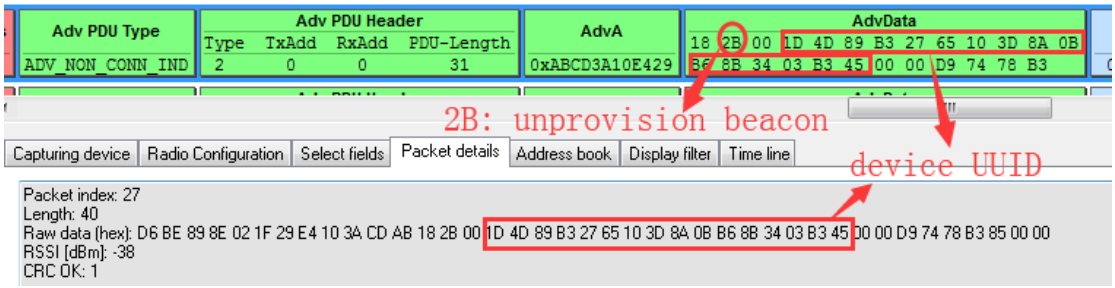
13.1.2.2 light 节点的 Device uuid 获取方法

device uuid 默认是通过 user_prov_multi_device_uuid()→uuid_create_by_mac(tbl_mac,prov_para.device_uuid)生成的，可通过以下几种方法获取：

1) 通过 BDT tool 读取 prov_para.device_uuid
2) 通过 Nordic APP "nRF Connect"获取 unprovision 广播包



3) 通过 TI sniffer 获取 unprovision 广播包



4) GATT provision 的时候，在连接成功的时候，会打印出 device uuid

```
<0027>23:03:16:292 [INFO]:(GattProv)CScanDlg::OnConnect:the device uuid is
: 1d 4d 89 b3 27 65 10 3d 8a 0b 29 e4 10 3a cd ab
```

5) gateway provision 的时候，选择某个 scan 得到的节点的时候，会打印出 device uuid

```
<0005>00:02:49:081 [INFO]:(GattProv)CScanDlg::provision link:the device uuid is
: 1d 4d 89 b3 27 65 10 3d 8a 0b 29 e4 10 3a cd ab
```

13.1.2.3 用户定制 uuid 的方法

如果用户想定制 device uuid，把 NORMAL_MODE_DEV_UUID_CUSTOMIZE_EN 置为 1 即可。

13.1.2.4 Provisioner static oob 数据库

Provisioner 端需要填写节点的 oob 数据到 oob 数据库文件(oob_database.txt)中：

oob 数据库数据格式如下：

device uuid(16byte) + oob(16byte)

例如： 1d4d89b32765103d8a0b29e4103acdab
ff000000000000000000000000000000

字段解析如下：

1d4d89b32765103d8a0b29e4103acdab：被组网节点的 device uuid.

ff000000000000000000000000000000：被组网节点的 oob 数据，即Light Firmware 在 flash 固定位置 FLASH_ADR_STATIC_OOB(例如 0x76300)写入的值。

如有多个节点，换行增加即可，比如：

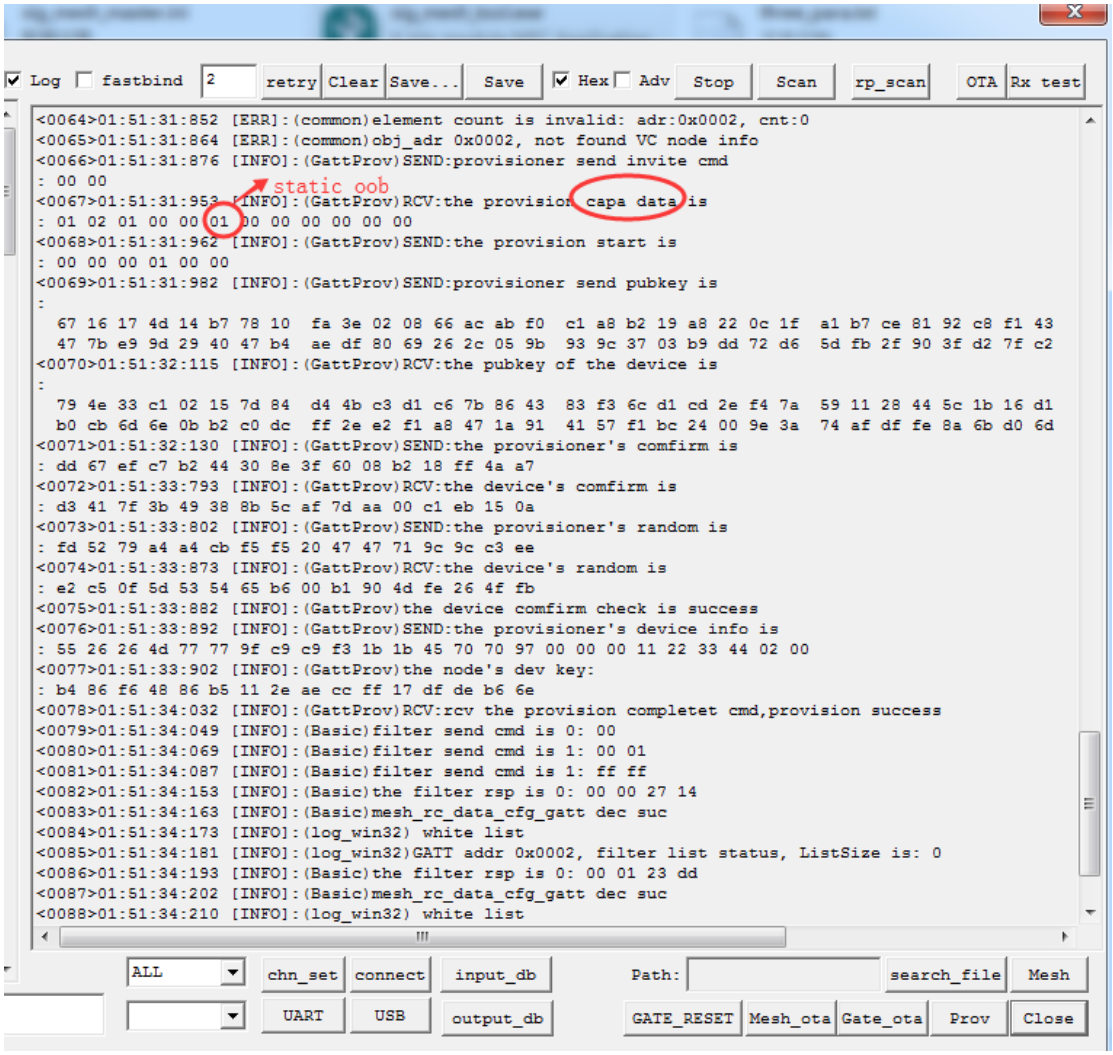
1d4d89b32765103d8a0b29e4103acdab ff000000000000000000000000000000

9f6f6e6e5e90943db9be6d79446a9e37 ffaa000000000000000000000000000000

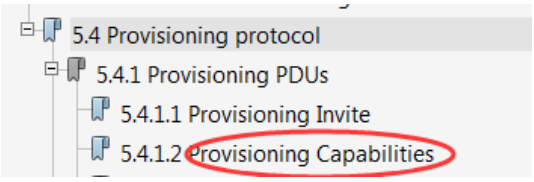
13.1.2.5 测试步骤

按正常流程测试组网即可。

static oob 组网成功的测试结果，使用 GATT master dongle 模式截图如下：



capability data，更详细的解析请参考 spec 以下章节



数据解析如下：

01 02 01 00 00 01 00 00 00 00 00 00

01：Provisioning PDU Type, 01 表示 Provisioning Capabilities

02：Number of Elements

01 00：Algorithms

00：Public Key Type

01：Static OOB Type

00：Output OOB Size

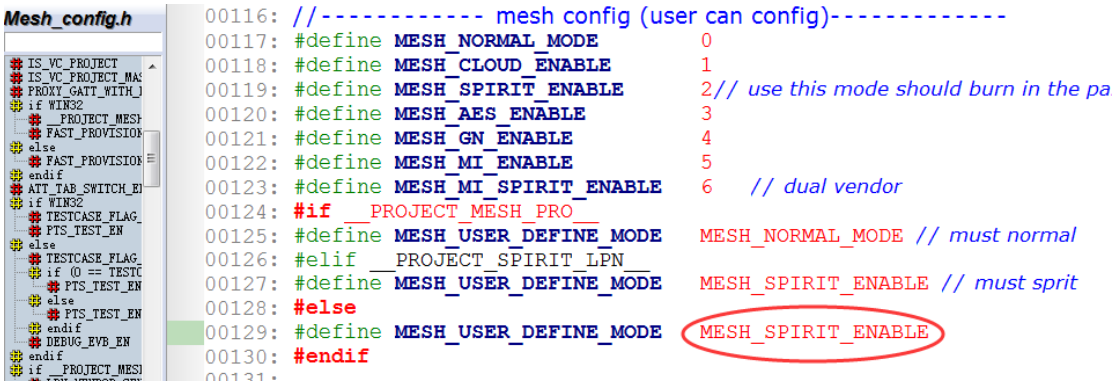
00 00：Output OOB Action

00：input OOB Size

00 00：Input OOB Action

13.2 阿里天猫精灵平台

13.2.1 配置方式

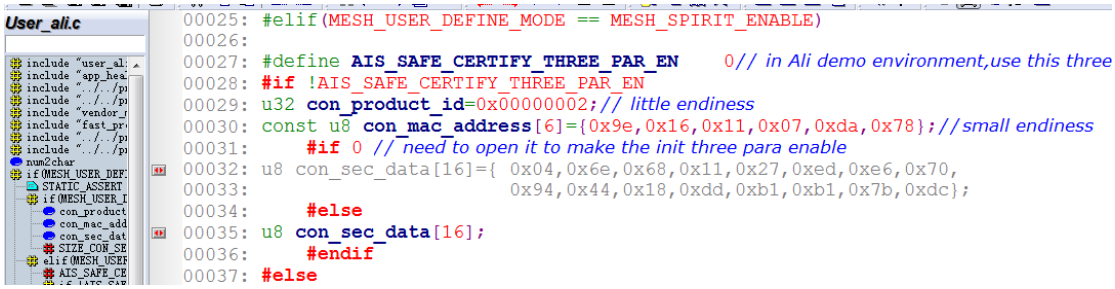


Provision 采用 MESH_STATIC_OOB 模式。

VENDOR_ID 是 0x01A8

13.2.2 向阿里申请三元组

默认的三元组信息在是空的(con_sec_data[16]为空)，具体代码在 user_ali.c 如下图所示：



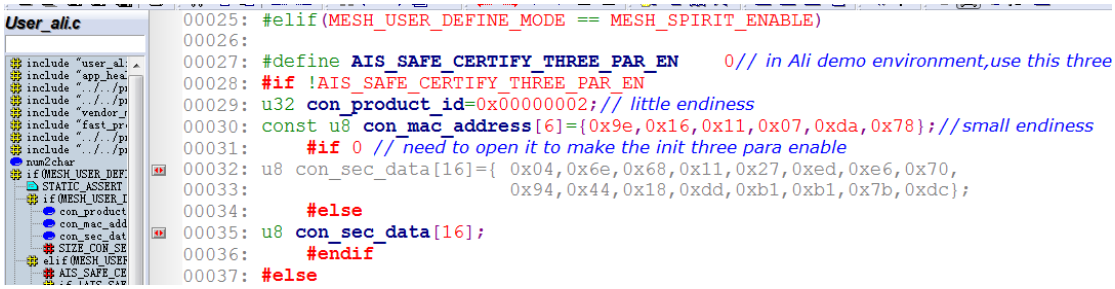
所以客户需要从阿里获取三元组：

(共 24byte：PID(4byte,小端) + MAC(6byte,大端) + secret data(16 byte))，

然后烧录到 FLASH_ADR_THREE_PARA_ADR(0x78000)，代码会自动读取 0x78000 的参数。

13.2.3 使用 SDK 默认的三元组信息

当只是为了做演示，可以使用 SDK 默认的三元组信息，打开方式如下:(使能 user_ali.c 文件里面预设的 con_sec_data[])



但是，因为默认的三元组只有一个，所以测试的时候，只能用于单节点的演示。

13.2.4 通过天猫精灵组网

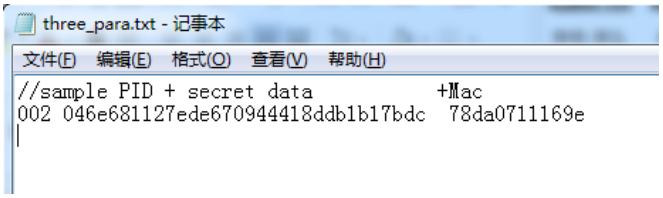
直接通过天猫精灵的语音命令组网就可以了。

13.2.5 通过上位机组网

因为天猫精灵模式默认只支持 static oob 的模式， oob 和三元组是有绑定关系的，所以上位机需要知道三元组的信息才能组网。所以需要把三元组添加到上位机的这个文件：

SIG_MESH_Release_Vxxx\tools\telink-ble-phone\three_para.txt

SDK 默认的三元组信息已经添加在这个文件里面了。添加新的三元组信息的时候，参考这个格式添加即可。



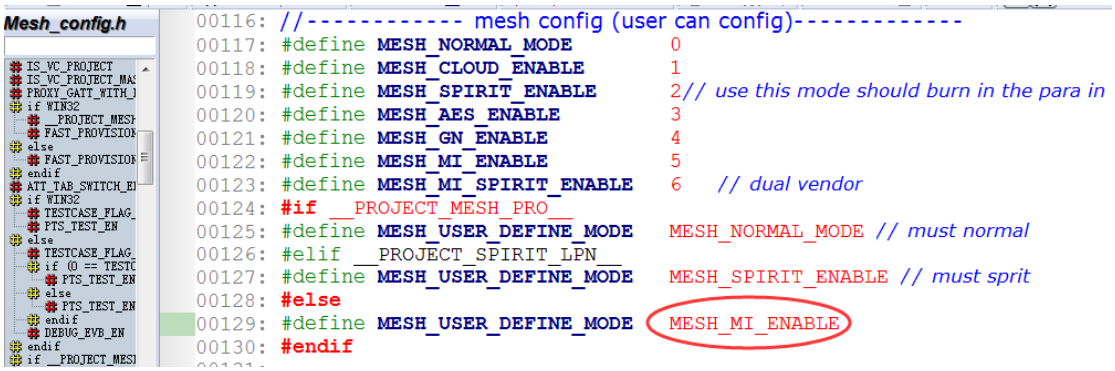
然后直接按“4. 调试工具操作说明”章节里面的“组网部分”进行组网就可以。

13.2.6 同时支持 static oob 和 no oob 模式

天猫精灵模式，节点端默认只响应 static oob 的组网模式，如果需要同时支持 no oob 模式，把 ENABLE_NO_OOB_IN_STATIC_OOB 由 0 改为 1，即可。

13.3 小米小爱同学平台

13.3.1 配置方式



Provision 采用 小米专有 mesh 配网模式。

VENDOR_ID 是 0x038F

13.3.2 认证数据的设定

✧ 研发测试模式:默认采用 SDK 预设的认证数据，dev_cert_pri[]，所以只能用于单节点的测试模式，默认有 6 个证书。因为这几个证书是公开的，如果已经被别人使用，会产生冲突。所以客户最好使用自己申请的证书，然后按生产模式的方式进行测试。

```
#define DEMO_CERT_TYPE0 0
#define DEMO_CERT_TYPE1 1
#define DEMO_CERT_TYPE2 2
#define DEMO_CERT_TYPE3 3
#define DEMO_CERT_TYPE4 4
#define DEMO_CERT_TYPE5 5

#define DEMO_CERT_TYPE  DEMO_CERT_TYPE1
```

✧ 生产模式:当需要生产，或者多节点网络的测试时，需要通过写 flash 的方式，步骤如下

Step 1 把 MI_CER_MODE 的宏定义为 FLASH_CER_MODE，重新生成 firmware。
Step 2 把认证数据烧录到 DEV_SK_FLASH_ADR(0x7f000)的位置。

13.3.3 组网测试

烧录完 firmware 后，请确保 flash 的 MAC 地址(512K flash 是在 0x76000 位置，1M flash 是在 0xFF000 位置)是空的(即全为 0xff)，否则会提示"无法连接"或者"组网失败"。

firmware 在第一次上电启动的时候，会从证书里面把 MAC 提取出来，写到 MAC 地址扇区，并生成一些必要的参数。

节点上电后，直接通过小爱同学进行组网即可。(语音指令示例:"小爱同学，添加设备")。

13.4 双 VENDOR 模式(阿里天猫精灵和小米小爱同学)

13.4.1 功能说明

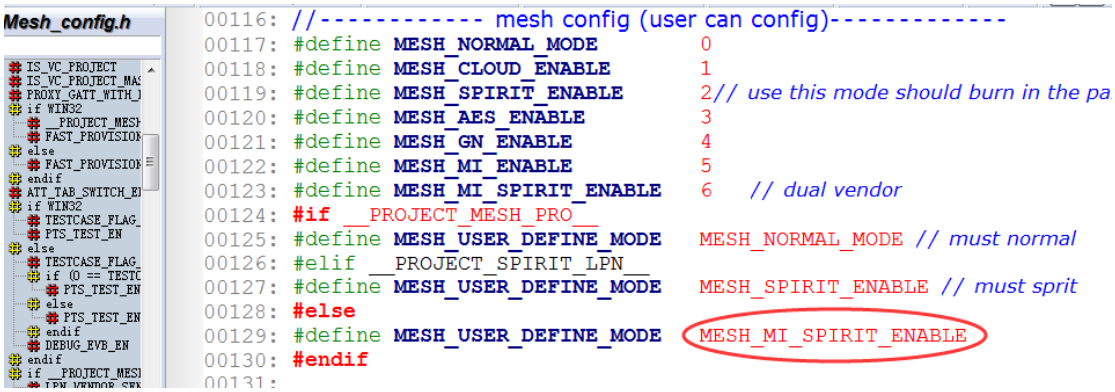
节点在出厂的时候，会同时发送阿里和小米模式的广播包，既可以被天猫精灵组网，也可以被小爱同学组网。一旦组网成功，后续的功能都按照被选择的模式执行，包括 vendor model，以及 transmit count 等参数，参数切换函数详见 mesh_ais_global_var_set()。

未配网状态下的产测功能依照小米模式进行。

已配网成功后，执行踢灯命令，或者恢复出厂设置动作，均可恢复进入双 vendor 模式，重新选择。

当前处于哪一种模式，可以通过 provision_mag.dual_vendor_st 这个变量查看。

13.4.2 配置方式



配网方式以及一些必要的参数设置，请参考本章 “天猫精灵”和“小爱同学”平台的介绍。

14. 恢复出厂配置

14.1 8258_mesh/8269_mesh 节点

14.1.1 函数介绍

Factory reset 的 reset 执行动作，请参考 factory_reset_handle()或者 kick_out()：

```
{
irq_disable();
factory_reset();// 执行 flash 的擦除动作
#if DUAL_MODE_WITH_TLK_MESH_EN
UI_resotre_TLK_4K_with_check();
#endif
show_ota_result(OTA_SUCCESS);   // LED 指示
start_reboot(); // MCU 重启
}
```

如果客户有修改 flash map，或者使用了 customer flash section(默认是 0x7a000—0x7f000，且默认不对该区域执行 erase)，需要重新确认 factory_reset()函数，确认是否有 sector 错误 erase 或者 遗漏 erase。

上电序列检测函数 factory_reset_cnt_check ()：

(a) 上电后， 经过 VALID_POWER_ON_TIME_US(默认 50ms) 时间后，才开始检测上电时序。因为要过滤某些电源上电时产生的脉冲电压。

(b) clear_st 等于 4：

首先通过 reset_cnt_get_idx ()获取断电前的序列，如果是奇数，则表示之前的上电序列不符合预期，直接清除上电序列，重新开始计数。如果是偶数，则表示符合预期。

然后，检查存储的上电序列值是否满足触发 factory reset 的条件，如果满足，则执行 factory reset。如果不满足，则立即对序列值加 1 处理。

(c) clear_st 等于 3，通过 get_reset_cnt()得到上电序列值，获取计时时间，并开始第一阶段的计时。

(d) clear_st 等于 2，第一阶段计时满足要求，通过 get_reset_cnt()得到上电序列值，获取计时时间，并开始第二阶段的计时。

(e) clear_st 等于 1，如果第二阶段计时结束，还未断电，则表示上电时间不符合预期，直接清除上电序。

14.1.2 默认触发动作

低功耗节点，比如 LPN，不能使用上电序列来触发，因为低功耗节点在睡眠的时候无法计时及判断时序。可以考虑通过按键触发等方式，调用 14.1 (1) 章节的函数即可。

非低功耗节点，可以用下面的流程来恢复出厂配置：

Step 1 上电 SIG_mesh 模组，等待 factory_reset_serials []里面设定的最大时间，默认是 30s。

因为超过 30 秒，就不符合 factory_reset_serials[]定义的任意一个上电序列，将清除之前未知的上电记录，确保后续的操作符合第一次上电时序要求)。

Step 2　SIG_mesh 模组重新上电 3 次，要求在上电后 0~3s 内断电(符合 factory_reset_serials[]前 3 个上电序列的要求)。
Step 3　SIG_mesh 模组重新上电 2 次，要求在上电后 3~30s 内断电（符合 factory_reset_serials[]后两个上电序列的要求)。
Step 4　重新上电 SIG_mesh 模组。

在 user_init()里面的 factory_reset_handle()会检测到之前的 5 次上电序列符合触发 "Factory Reset "的要求，红色 led 指示灯以 1Hz 频率闪烁 8s，恢复出厂配置成功。

**注意：因为某些供电模组，断电完成需要一定时间，为了确保 MCU 完全停止，所以要求断电之后，需要等待一段时间再上电，比如 2 秒，等。这个时间需要根据实际的供电模组确认。*

上电时序由下面数组定义:

```
u8 factory_reset_serials[]    =    { 0, 3,
                                     0, 3,
                                     0, 3,
                                     3, 30,
                                     3, 30,};
```

14.1.3 修改为其它上电序列的方法

在 factory_reset_serials[]数组中修改、增加、删除对应的序列即可。要求:

- ✧ 左边的值比右边的小。
- ✧ 增加或者删除时，必须是增加或删除两个值(因为一个序列对应两个值)。

比如，要改为 6 个序列，改为以下方式即可。

```
u8 factory_reset_serials[]    =    { 0, 3,
                                     0, 3,
                                     0, 3,
                                     3, 30,
                                     3, 30,
                                     3, 30,};
```

14.1.4 触发复位动作后，还能恢复到之前 mesh 网络的功能

以上 1----3 介绍的模式是正常的恢复出厂设置的模式。

有些用户触发恢复出厂设置后，如果在一定时间内还未重新配网(比如 30 秒)，希望能自动恢复到之前的网络信息。sdk 中提供了实现此功能，所需要的 2 个 API:

- ✧ mesh_reset_network(u8 provision_enable):把 ram 里的网络信息恢复到默认网络状态，参数 provision_enable 为 1 时表示设备会发 unprovision beacon 和 pb_adv，设备可被重新配网。此时只是 RAM 数据还原了，但是 flash 信息并没有改变。

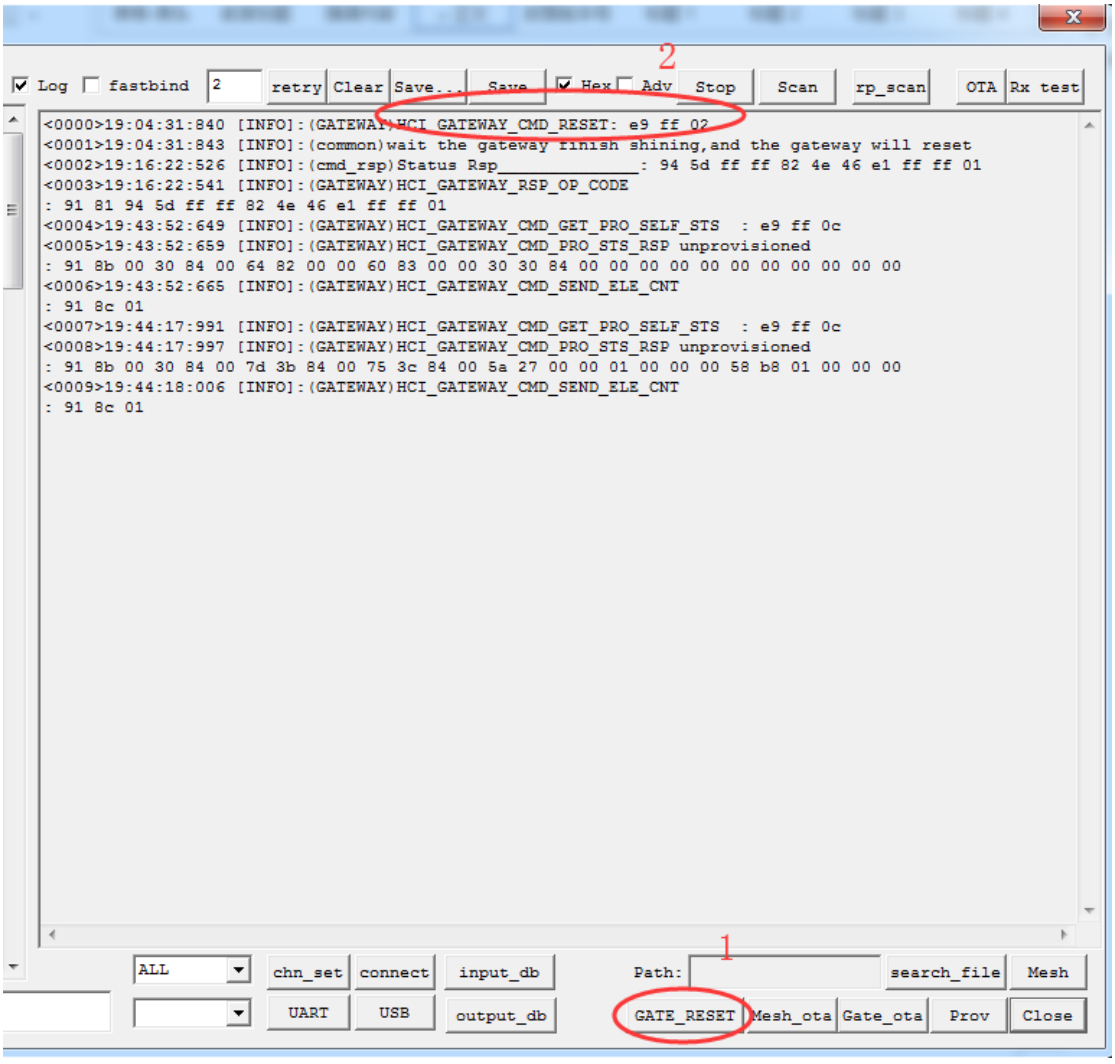
✧ mesh_revert_network():从 flash 里重新加载网络信息。

实现方式:用户触发恢复出厂设置动作进入 kick_out 函数时，直接调用 mesh_reset_network(1)把 ram 里的网络信息恢复到默认网络，不调用 factory_reset()和 start_reboot()。如果用户想恢复网络，直接调用 mesh_revert_network()从 flash 里重新加载 mesh 信息即可

14.2 gateway 节点 + 上位机

gateway 进行复位，需要执行两个动作:一个是 删除 mesh_database.json， 另一个是 把 gateway dongle 的 flash 的参数区也都清除(比如通过 5 次上电的复位方式)。因为有两个动作需要执行，所以为了方便操作，在上位机增加了一个"GATE_RESET" 按钮， 执行这个按钮后，就会执行上述两件事，gateway dongle 把自己的 flash 参数区清除后，会自动调用 start_reboot()软重启。

注意，这个按钮，只是复位了 gateway 本身，并不会发命令出去把其他节点也进行 RESET。



14.3 GATT master dongle + 上位机

master dongle 的 flash 里面是没有存储参数的，所以只要删除 上位机的 mesh_database.json 文件，然后重新打开上位机工具即可。

14.4 LPN 节点

低功耗节点，不能通过 5 次上电的方式进行复位，因为当处于 sleep 状态时，断电后，还需要一段时间才会把电消耗完，这种情况下，就不好确定是否断电成功。所以 LPN 节点一般需要使用按键等方式，触发调用本章第一小节的函数即可。

Demo LPN：长按按键 SW1(MESH_LPN_FACTORY_RESET_KEY) 3 秒以上(LONG_PRESS_TRIGGER_MS) ,会闪灯 4 次，提示复位成功。

14.5 switch 节点

低功耗节点，不能通过 5 次上电的方式进行复位。需要使用按键的方式，长按按组合键:RC_KEY_A_ON + RC_KEY_4_OFF 三秒以上，会闪灯 4 次，提示复位成功。

15. Fast bind 模式(即 PROVISION_FLOW_SIMPLE_EN 模式)

15.1 功能介绍

此模式属于非标准模式，用于加快标准流程的配网速度。具体改善如下：Spec 定义 provision flow 执行完，分配好 unicast address，netkey 等信息后，还需要按顺序发送 get composition data，app key add 以及 对获取到的 composition data 里面的每一个 model 执行 key bind 动作。这个在有些应用场景中不需要那么复杂，在多数情况下一个设备仅仅会有一个 APPkey 而已。所以我们定义了这个 Fast bind 模式。

Fast bind 模式主要是优化 key bind 部分，当 provisioner 发送 app key add 后，不再发送 key bind 命令。节点收到 app key add 后，自行对自身的所有 model 执行 key bind 动作，详见 PROVISION_FLOW_SIMPLE_EN 这个宏括起来的 code

另外，为了更进一步的简化组网流程，我们 demo app 也不需要执行 get composition data 的命令，直接通过组网获取的 device UUID 里面的 PID 来查询数据库来获取对应的 composition data 的所有信息。下图是 firmware 端把 PID 写到 device uuid 的函数。

```
void set_dev_uuid_for_simple_flow( u8 *p_uuid)
{
    simple_flow_dev_uuid_t *p_dev_uuid = (simple_flow_dev_uuid_t *)p_uuid;
    memcpy(&p_dev_uuid->cps_head, &gp_page0->head, sizeof(p_dev_uuid->cps_head));
    memcpy(p_dev_uuid->mac, tbl_mac, sizeof(p_dev_uuid->mac));
    // set uuid
}
```

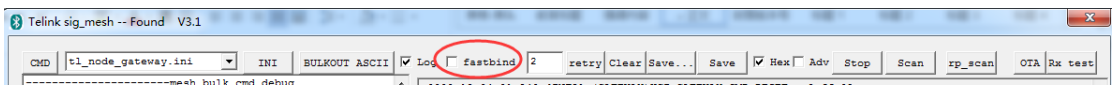
如果，客户不希望通过查询数据库的方式获取 composition data，也可以修改 app 的 flow，增加 get composition data 的命令。

15.2 配置方式

节点 Firmware：把 PROVISION_FLOW_SIMPLE_EN 设置为 1 即可。

15.3 功能演示

15.3.1 上位机的配置：



15.3.2 APP 界面的配置

APP 也要通过控件选择该模式，具体请参考 app 的使用说明。

16. 私有 Fast provision 功能

16.1 功能介绍

区别于 remote provision，只能逐一节点进行配置，网络较大时，组网时间还是过长的问题，我们增加了私有的批量组网模式，即在默认 key 的 network 里面，增加 vendor 命令，比如 VD_MESH_RESET_NETWORK 等，向 0xffff 的目标地址发送 network key，app key，iv index，(只需要发送一次，整个网络都可以同时接收)，然后根据 mac 的不同，逐一分配 unicast address。这种方式可以把多跳范围内的节点都同时组网进来。device key 是按一定规则根据 mac 地址生成，所以不需要通过 mesh 命令单独分配。

另外，在一个已经配置好的 mesh 网络内，还可以继续通过 fast provision 的方式把新购买的未配网设备添加进当前的网络。

16.2 配置方式

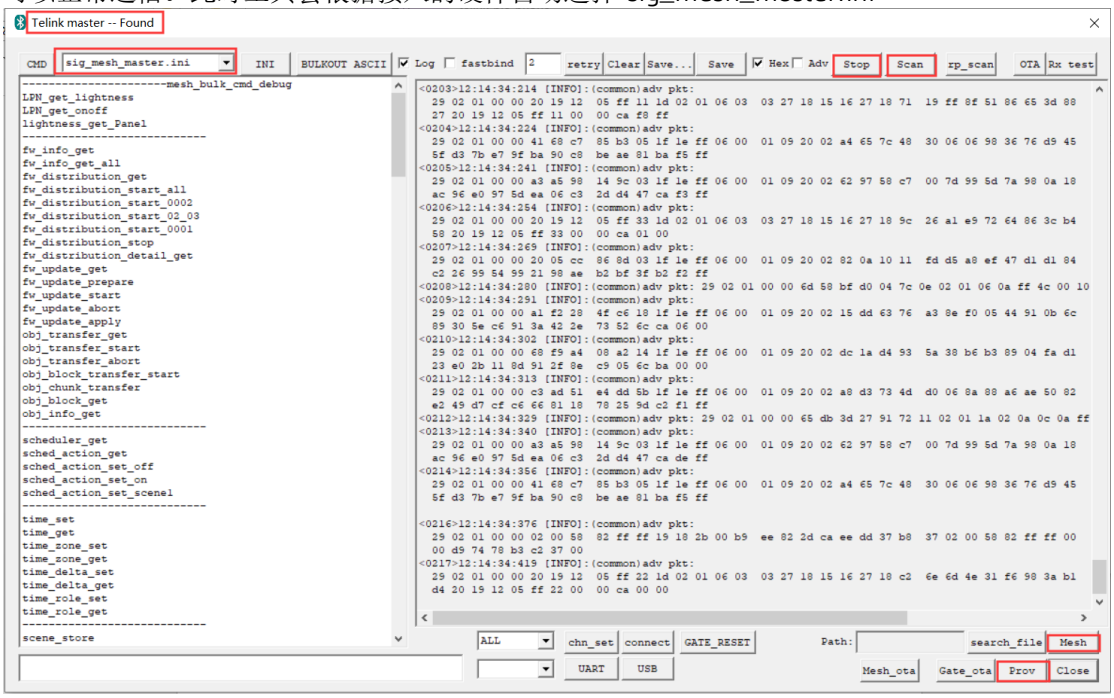
把 FAST_PROVISION_ENABLE 设置为 1。编译 8258_mesh 工程，下载到 3 个（2 个以上）8258 dongle。

目前 GATT master dongle 模式和 APP 默认支持处理 fast provision 功能。 gateway 暂不支持此功能。

16.3 功能演示

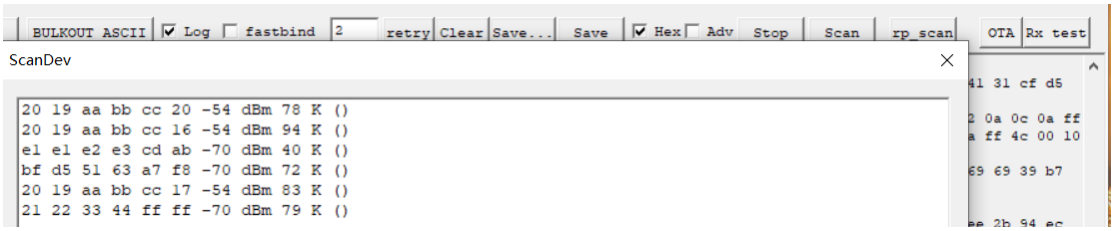
以下 demo 演示了如何将多个（大于 2 个）8258 节点批量加入现存 mesh 网络。

1) 打开“sig_mesh_tool.exe”工具，将烧录好程序的 8269 master Dongle 插到 PC 的 USB 口中。
2) 如下图所示，工具左上角的“Found”表示 8269 Master Dongle 和 PC 工具正常连接，并且可以正常通信。此时工具会根据接入的硬件自动选择“sig_mesh_master.ini”



3) 将 8258mesh 节点上电。

4) 点击右上角的“Scan”按钮，工具将打开一个“ScanDev”窗口，该窗口会显示对应的 MAC 地址列表，包含 rssi 和频率。



“ScanDev”窗口

(5) 双击“ScanDev”窗口中对应的条目，选择节点。8269 Master dongle 模式：双击后会建立 BLE 连接，如果 8269 Master dongle 上的红灯亮起，表示 BLE 连接正常建立。“Stop”按钮用于终止当前的 BLE 连接，8269 Master Dongle 上的白灯亮起，表示 BLE 连接断开。目前只支持单个节点的 BLE 连接。

(6) 点击右下角“Prov”按钮来打开“provision”窗口。按下图顺序点击 1/2/3 的按钮。(注意必须先选择 Fast prov mode，不需要点击 bind_all 按钮)。

几秒钟后，会看到所有节点都在闪烁 3 下，表示都已经组网成功。点击“Mesh”按钮进入 mesh 界面可进行开关灯等操作。

17. 私有 online status 功能演示

17.1 功能介绍

目前 SIG mesh spec 提供的在线离线监测机制，可以通过 heartbeat 以及 publish 机制实现。节点 onoff 等状态信息实时监测的方式可以通过 publish 机制实现。

当同时实现在线离线监测以及状态信息实时监测就需要 publish 机制了。但是 publish 机制有以下几个限制：

publish message 一般需要设置 relay，所以空中的包会比较多。

publish 周期不能设置太短(一般需要几十秒或者更久)，否则空中的包会太多，影响正常的控制。

有时候需要几条 publish status message，才能把包含所有需要上报的 status，这个时候空中的包会更多。

所以我们增加了 online status 机制，目的是:实现快速有效的在线离线检测，以及上报节点重要状态信息，同时还能有效的降低网络中的数据包。

17.2 配置方式

把灯节点端的 app_config.h 的 ONLINE_STATUS_EN 由 0 改为 1.

目前 GATT master dongle 模式和 APP 默认支持处理 online status message。 gateway 暂不支持此功能。

17.3 数据包格式

Online status 数据包采用 ADV NON CONN IND 的 adv 进行发送，payload 的 type 采用自定义的 MESH_ADV_TYPE_ONLINE_ST (0x62)，如下图。

Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
				Type	TxAdd	RxAdd	PDU-Length					
+319991 =1599997	0x25	0x8E89BED6	ADV_NON_CONN_IND	2	0	0	37	0xFFFF82580002	1E 62 0F 07 31 56 F2 03 47 DA 76 D7 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 91 9F AB 4A	0x000080	-38	OK
+360010 =1960007	0x25	0x8E89BED6	ADV_NON_CONN_IND	2	0	0	37	0xFFFF82580002	1E 62 0F 07 31 57 F2 03 47 DA 76 D5 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 7E 1D 3E 25	0x000076	-38	OK
+319994 =2280001	0x25	0x8E89BED6	ADV_NON_CONN_IND	2	0	0	37	0xFFFF82580002	1E 62 0F 07 31 54 F2 03 47 DA 76 D4 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 0C C1 5E 66	0x000084	-38	OK

Payload 有效字节的长度是 24byte，默认每个节点需要占用 6 个 byte，详细请参考

```
typedef struct{
    u16 dev_adr :15;      // don't change include type
    u16 rsv     :1;
    u8 sn;           // don't change include type
    u8 par[MESH_NODE_ST_PAR_LEN];   //lumen-rsv,
}mesh_node_st_val_t;
```

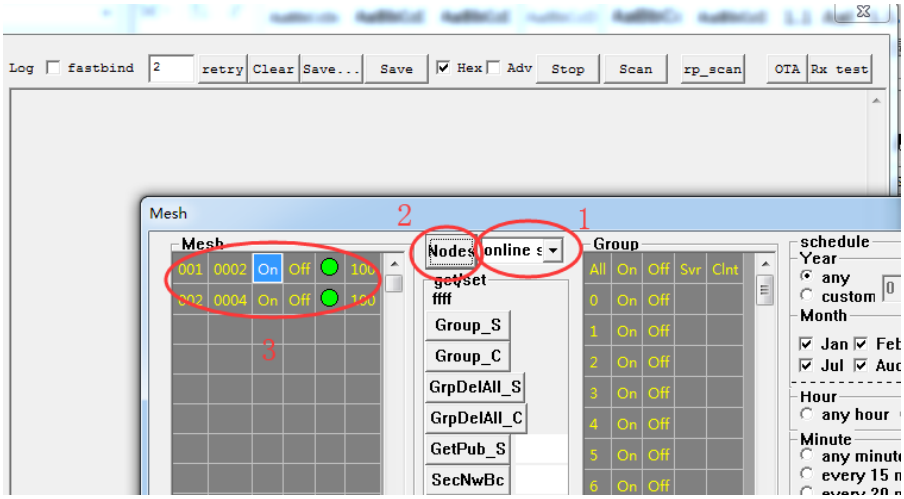
每个节点的有效长度是 MESH_NODE_ST_PAR_LEN(3)，具体填充的 data 根据每个产品可以自定义。默认填充 BYTE 0 是亮度，BYTE 1 是色温值，BYTE 2 保留。针对客户需求修改 device_status_update()即可。

```
void device_status_update()
{
    // packet
    u8 st_val_par[MESH_NODE_ST_PAR_LEN] = {0};
    memset(st_val_par, 0xFF, sizeof(st_val_par));
    // led_lum should not be 0, because app will take it to be light off
    st_val_par[0] = light_lum_get(0, 1);
    #if (LIGHT_TYPE_CT_EN)
    st_val_par[1] = light_ct_lum_get(0, 1);
    #else
    st_val_par[1] = 0xff;    // rsv
    #endif
    // end

    ll_device_status_update(st_val_par, sizeof(st_val_par));
}
```

注意，MESH_NODE_ST_PAR_LEN(3)，可以修改，但是长度越大，传输速度会相应变慢。并且这个长度没有 length 字段来描述，所以，需要在定义网络的时候，就要先确定好 MESH_NODE_ST_PAR_LEN 的值。如果网络中有节点配置的 MESH_NODE_ST_PAR_LEN 值不一致，就会有兼容性问题，导致数据格式解析错误。

17.4 GATT master dongle 上位机演示



Step 1　　如图选择 online status 模式

Step 2　　如图点击“Node”按钮，点击后，从 log 窗口可以看出，并没有发送 lightness get 等类似命令，而是通过自定义的 UUID 直接获取直连节点里面的 online status 数据。

Step 3　点击 Node 按钮后，左边的节点显示窗口可以看到当前网络的节点信息。

18. OTA 简要测试说明

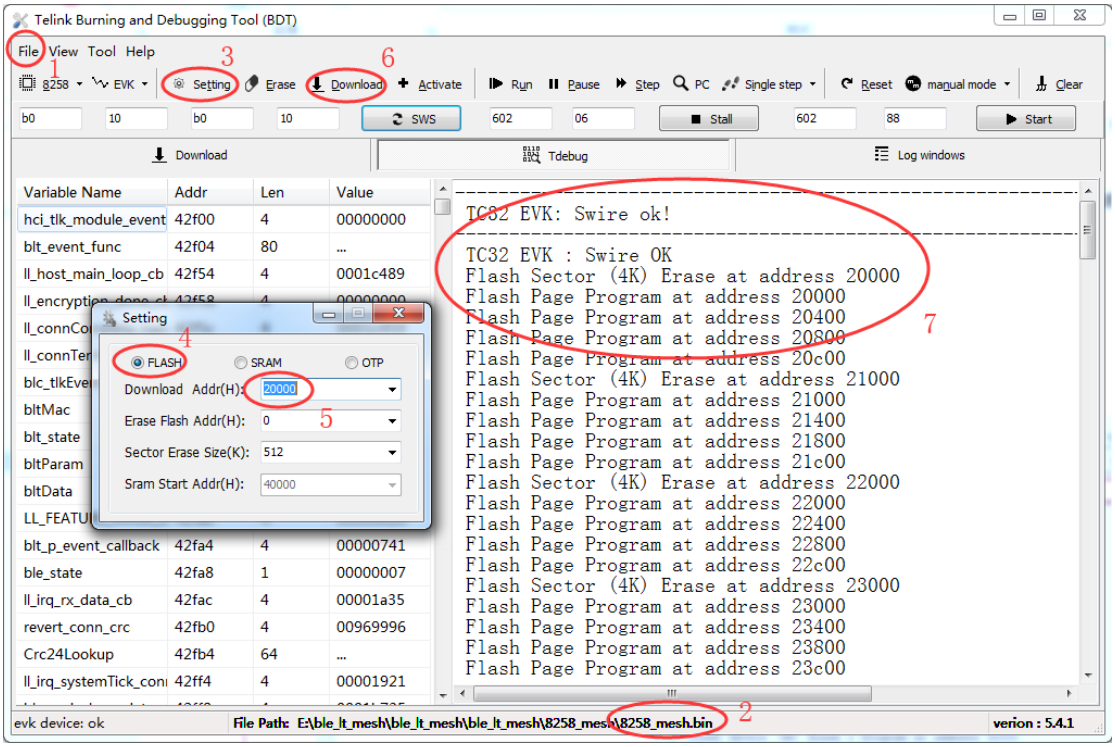
18.1 GATT master dongle 对 BLE 直连节点进行 firmware 更新的 OTA

此模式是点对点的 BLE 直连 OTA。

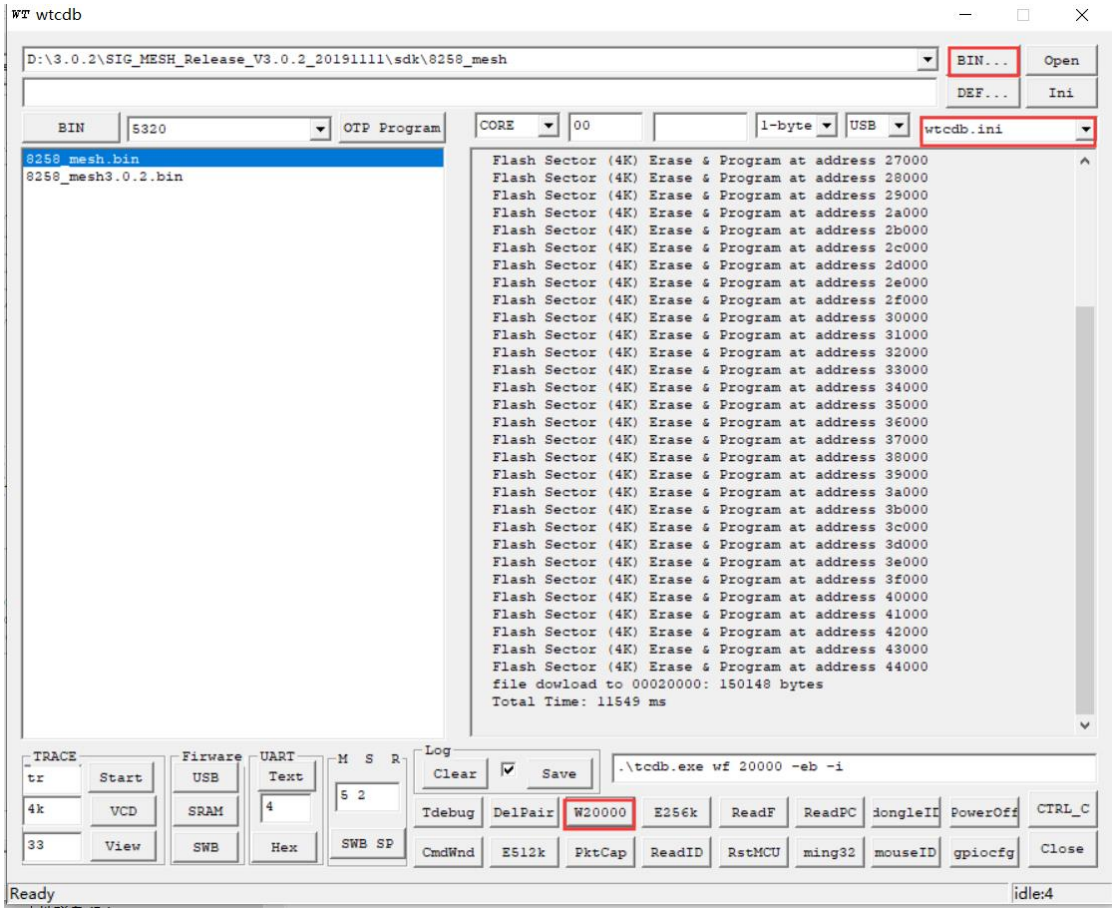
1) 将需要 OTA 的 BIN 文件（New FW）按照 7.1 的说明下载到 8269 master dongle 的 flash 地址 0x20000 开始的位置，0x0000 的位置烧录 8269_mesh_master_dongle.bin。

注意:烧录 New FW 的差异:

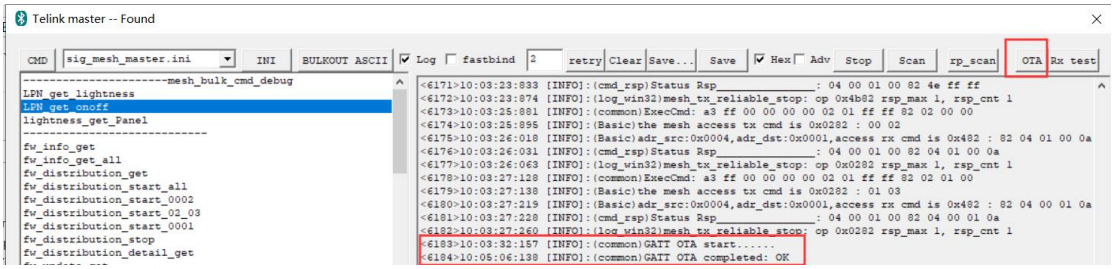
BDT 工具，请参考以下图示步骤:



如果是 wtcdb 工具，应点击“WF20000”按钮开始烧录。请参考以下图示步骤:



2) 参考第 7.2 小节"GATT master dongle 模式的 BLE 连接和加灯"的的(1)~(5)的步骤，建立起目标节点和工具之间的 BLE 连接。
3) 点击 OTA 按钮，启动 OTA 过程。如果正常完成 OTA，该节点会连续闪烁 8 下。

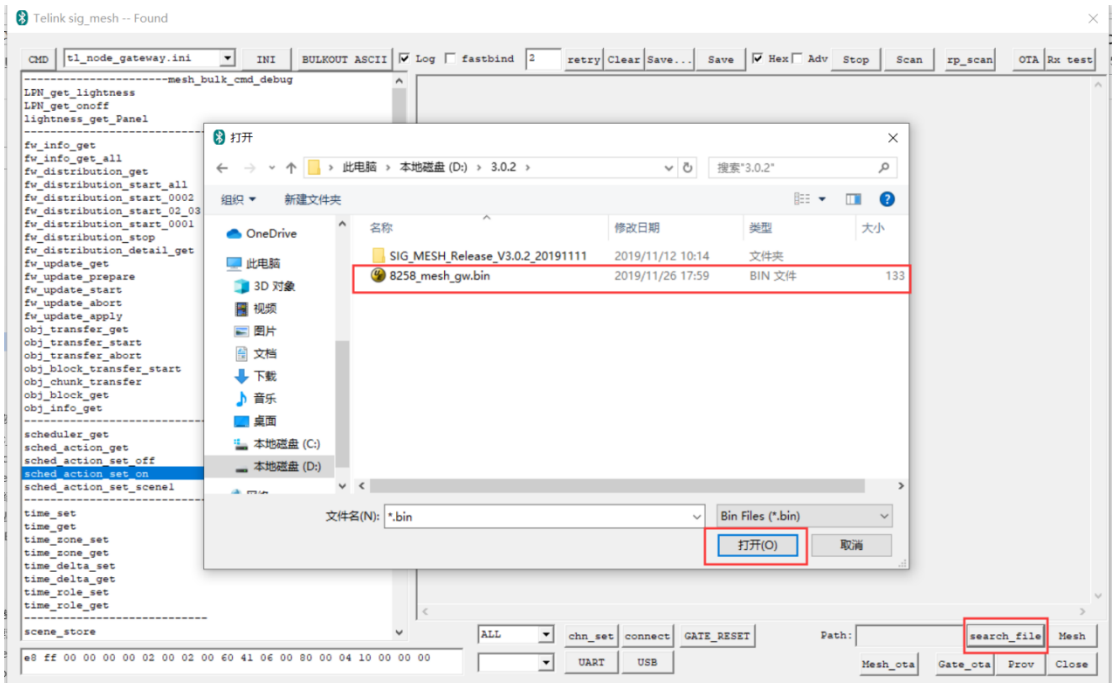


4) OTA 部分的命令以及协议部分详细部分，可参考《AN_17092701_Telink 826x BLE SDK Developer Handbook》的 6.4 章节。里面对命令和协议格式有详细的说明。

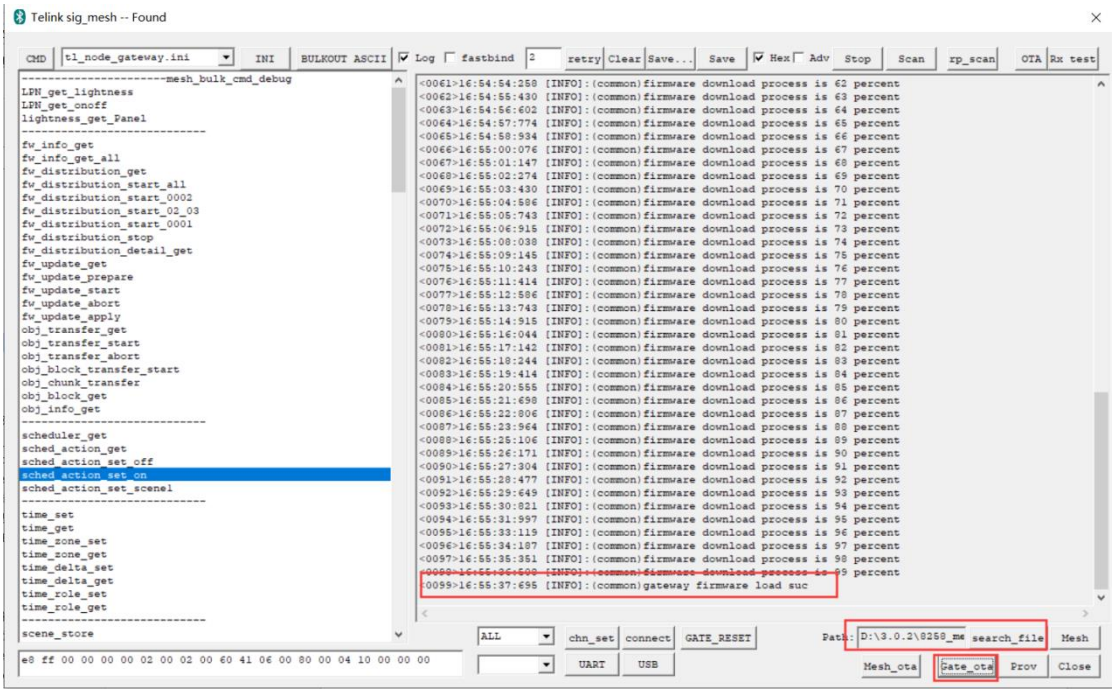
18.2 GATE WAY 节点对自己进行 firmware 更新的 OTA

Getway Gate OTA 目的是为 getway 本身固件做升级。

1) 打开 BDT 工具，把 8258_mesh_gw.bin 下载到 8258dongle。
2) 工具左上角的"Found"表示 8258 Dongle 和 PC 工具正常连接，并且可以正常通信。
3) 点击右下角 search_file 按钮，选择不同版本的 8258_mesh_gw.bin 文件；



4) 点击 Gate_ota 按钮，开始升级。LOG 提示 gateway firmware load suc 表示升级成功，升级成功后，gateway 会自动重启，并启用新的 firmware。



19. 网络分享

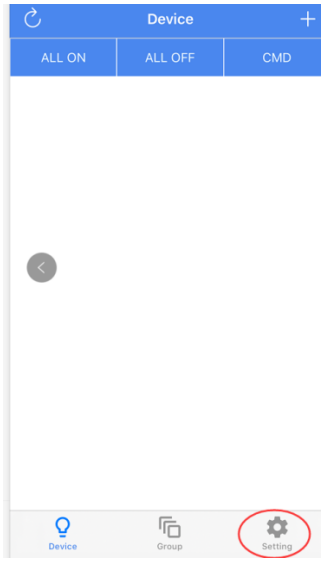
19.1 Gateway 或者 GATT master dongle 模式组网后，分享给 APP

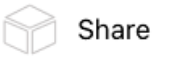
Step 1　用 VC master 或 getway 组网，确定 8258 dongle 节点能正常组网和控制，
Step 2　找到"sig_mesh_tool.exe"工具文件夹里生成的 文件。
Step 3　把 mesh.json 文件导入 TelinkSigmesh APP;

IOS APP：步骤如下：

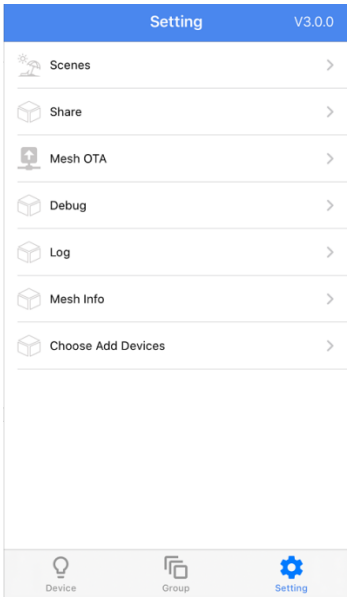
Step 1　将手机连接到安装了 iTunes 的电脑上。
Step 2　点击 iTunes 左上角的手机图标进入 iTunes 设备详情界面。
Step 3　选择 iTunes 左侧的"文件共享"，然后在应用中找到并点击 demo APP "TelinkSigMesh"，等待 iTunes 加载文件。
Step 4　文件加载完成后，将电脑上的 json 文件拖入右侧的"TelinkSigMesh"的文稿中。
Step 5　APP 点击 IMPORT 按钮选择刚刚的 JSON 文件进行加载。此步骤详见以下图片所示：

第一步：打开 APP TelinkSig mesh,点击 Setting 按钮



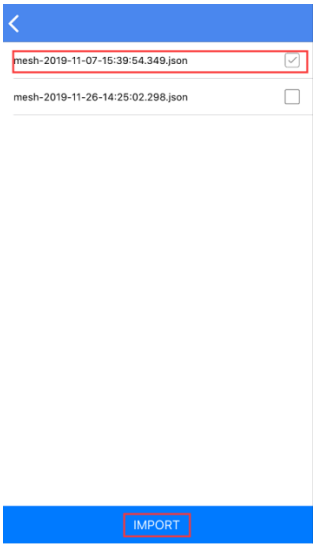


第二步点击 ——————— 按钮



第三步，点击 IMPORT 按钮

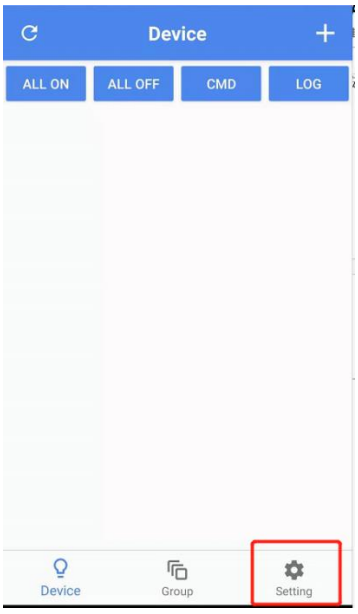
第四步，选择 mesh.json 文件，点击 IMPRT



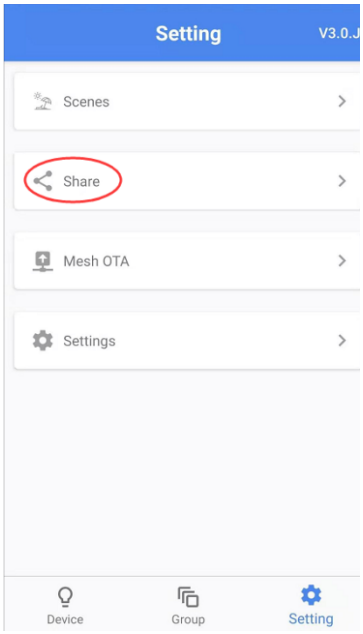
Android APP 导入文件步骤如下:

Step 1　将手机连接到电脑上，把 mesh.json 文件导入手机任意文件夹，记住文件夹的路径。

第一步:打开 APP TelinkSig mesh,点击 Setting 按钮

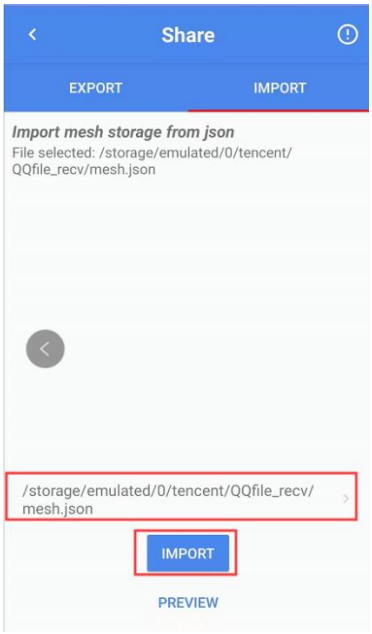


第二步点击  按钮



第三步，点击 IMPORT 按钮

第四步 点击 Select File 选择从电脑导入手机的 mesh.json 文件。



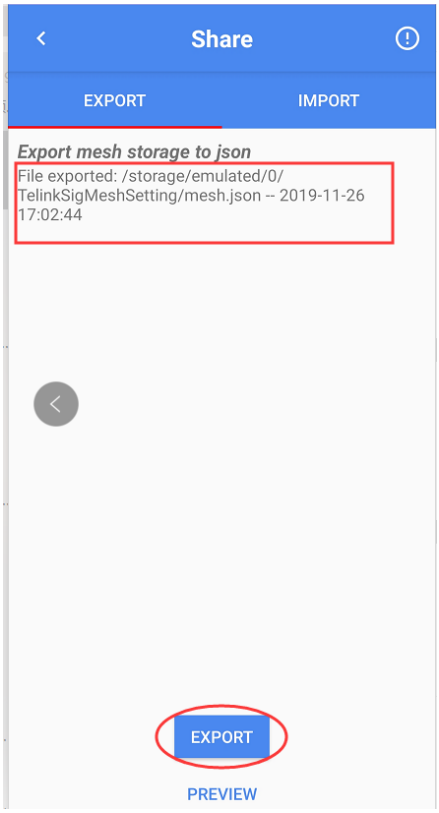
第五步 点击 

按钮。

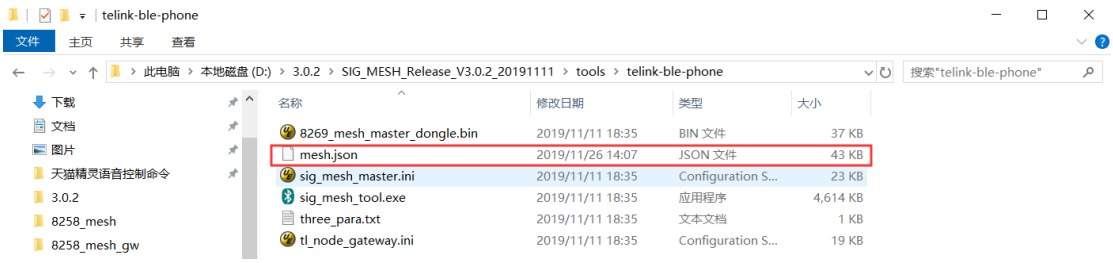
Step 2 导入成功后，返回主页面:APP Device 界面，此时会显示分享过来的节点，这些节点是 VC 工具组网的节点，APP 可以受控，并且 VC 和 APP 都能控制节点。

19.2 APP 模式组网后，分享给 Gateway 或者 GATT master dongle

1) 用 ios 或者 android 版本的 TelinkSIGmesh APP 进行组网，能正常控制功能。
2) 点击 Setting 按钮，再点击 Share 按钮，进入 share 界面，点击 EXPORT 按钮,生成 JSON 文件。界面提示 JSON 文件路径所在的文件夹。



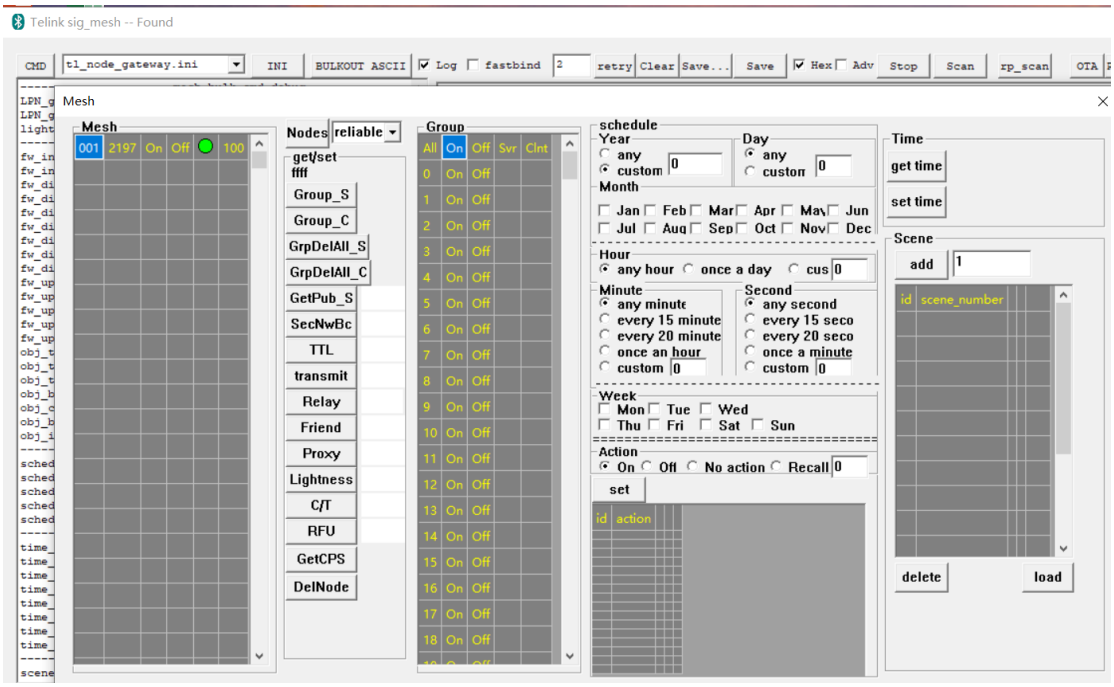
3) 将 APP 生成的 mesh.json 文件拷贝到电脑的"SIG_MESH_TOOL"工具的文件夹里，如果文件夹里已经存在，则直接替换。



4) 此时确保已经把 master 8269 Dongle 或者 gateway dongle 接到 PC 上。
5) 打开"SIG_MESH_TOOL"工具，此时会显示 master 8269 Dongle 和 PC 工具正常连接或者 getway 8258 Dongle 和 PC 工具正常连接。

Getway dongle 一般来说是还没有被配置过网络的，如果已经被配置过，里面的信息会被删除，并且使用导入的数据。

6) 点击"mesh"按钮，此时"mesh"窗口会出现导入的节点，并且可以受控。
7) 至此分享完成，gateway/master dongle 和 APP 都能控制节点。


Telink sig_mesh -- Found
CMD
tl_node_gateway.ini
INI
BULKOUT ASCII
Log
fastbind
retry
Clear
Save...
Save
Hex
Adv
Stop
Scan
rp_scan
OTA
Mesh
Nodes
reliable
get/set
ffff
Group_S
Group_C
GrpDelAll_S
GrpDelAll_C
GetPub_S
SecNwBc
TTL
transmit
Relay
Friend
Proxy
Lightness
C/T
RFU
GetCPS
DelNode
Group
All
On
Off
Svr
Clnt
schedule
Year
any
custom
Day
Month
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Hour
any hour
once a day
Minute
any minute
every 15 minute
every 20 minute
once an hour
Second
any second
every 15 seco
every 20 seco
once a minute
Week
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Action
On
Off
No action
Recall
set
id
action
Time
get time
set time
Scene
add
id
scene_number
delete
load

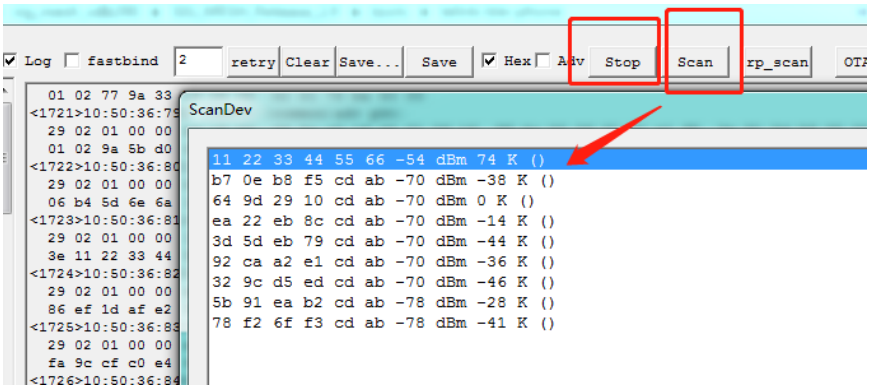
20. 通过 INI 控制节点实例讲解

20.1 设备入网

设备的入网流程在 PB-GATT 和 PB-ADV 会有一些不同，但是在界面操作上流程执行相同，都将执行如下步骤：

Step 1 扫描 UNprovision_beacon 广播的设备
Step 2 根据扫描到 UNprovision 广播的设备的 MAC 进行连接
Step 3 Provision device
Step 4 Bind model

此例程采用的是 mast dongle 进行测试。点击 stop，后点击 scan 进入扫描模式，根据扫描到的设备进行连接，如下连接 mac:112233445566 的设备。



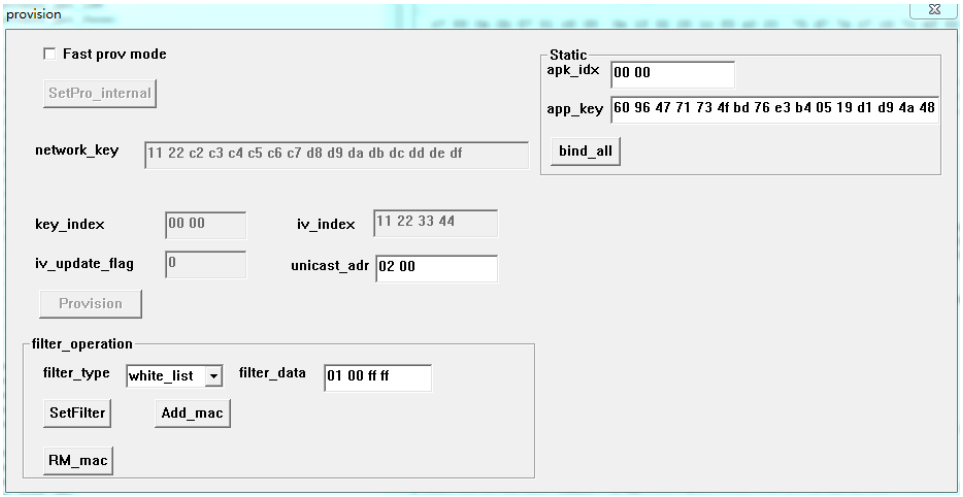
设备连接成功后 log 信息：

```
  29 02 01 00 00 e4 4a d8  a7 6c 7f 1f 1e ff 06 00  01 09 20 02 23 73 90 d6  fc c3 94 59 c5 fa 87 08
  86 ef 1d af e2 85 c0 41  6b 83 d8 ca 14 00
<11788>10:53:00:253 [INFO]:(common)adv pkt: 29 02 01 00 00 26 29 c0 89 d2 7b 0e 02 01 1a 0a ff 4c 00 1
<11789>10:53:00:266 <11790>10:53:00:396 [INFO]:(gatt_provision)CScanDlg::OnConnect:the device uuid is
: 91 2f 68 1d 5c 48 fb 3d b6 3e 11 22 33 44 55 66
[INFO]:(common)adv pkt:
  29 02 01 00 00 11 22 33  44 55 66 1d 02 01 06 03  03 27 18 15 16 27 18 91  2f 68 1d 5c 48 fb 3d b6
  3e 11 22 33 44 55 66 00  00 ca 4a 00
<11791>10:53:01:118 [INFO]:(common)Mesh Provisioning Service:
<11792>10:53:01:132 [INFO]:(common)uuid:dc 2a
<11793>10:53:01:144 [INFO]:(common)the handle:13
<11794>10:53:01:160 [INFO]:(common)Mesh Proxy Service:
<11795>10:53:01:177 [INFO]:(common)uuid:de 2a
<11796>10:53:01:191 [INFO]:(common)the handle:1c
<11797>10:53:01:201 [INFO]:(Basic)filter send cmd is 0: 00
<11798>10:53:01:222 [INFO]:(Basic)filter send cmd is 1: 4b 7e
<11799>10:53:01:242 [INFO]:(Basic)filter send cmd is 1: ff ff
<11800>10:53:01:264 [INFO]:(iv_update)app tx beacon with GATT,IV index step0: : 12 34 56 78 12 34 56 7
<11801>10:53:01:279 [INFO]:(iv_update)secure NW beacon:: 17 2b 01 00 44 cf 7a d5 44 8c f1 6e 12 34 56
<11802>10:53:01:359 [INFO]:(Basic)the filter rsp is 0: 00 00 46 6f
<11803>10:53:01:373 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11804>10:53:01:388 [INFO]:(log_win32) white list
<11805>10:53:01:405 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 0
<11806>10:53:01:422 [INFO]:(Basic)the filter rsp is 0: 00 01 74 0e
<11807>10:53:01:432 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11808>10:53:01:444 [INFO]:(log_win32) white list
<11809>10:53:01:456 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 1
<11810>10:53:01:467 [INFO]:(Basic)the filter rsp is 0: 00 02 85 01
<11811>10:53:01:479 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11812>10:53:01:494 [INFO]:(log_win32) white list
<11813>10:53:01:507 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 2
```

Provision 参数设置与设备入网

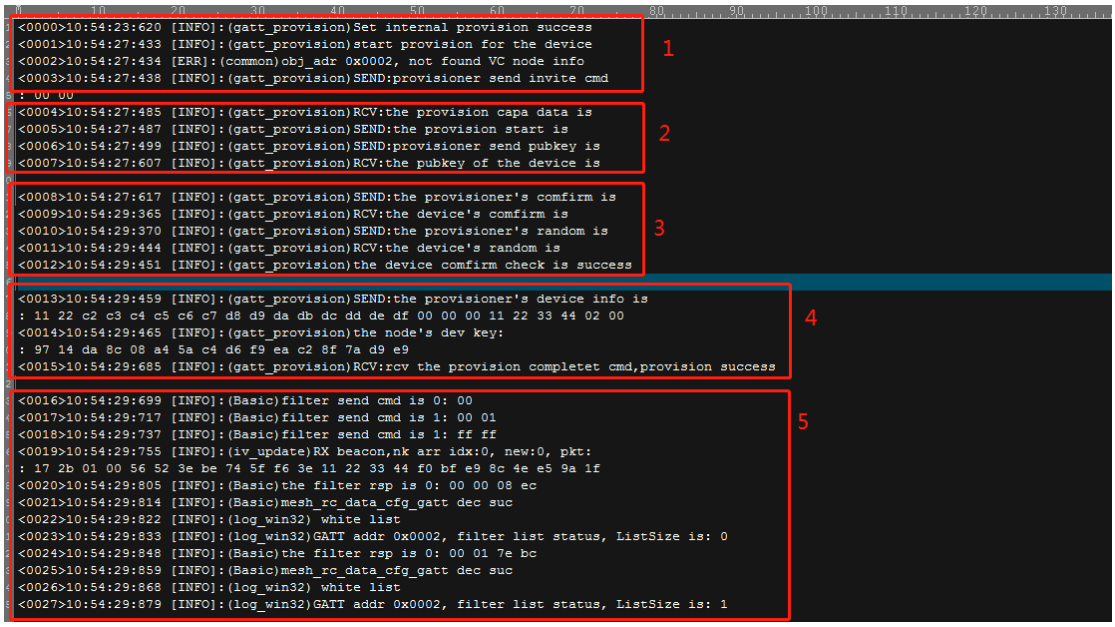
1) 点击 prov 按钮，进入 provision 界面，首先点击 SetPro_internal 将 net key 下发给 dongle。

2) 点击 provision 按钮，对已连接的 MAC:112233445566 设备进行 provision 动作，provision 过程中将上位机生成 netkeyindex，IVindex，unicast_addr 下发给设备。
3) 点击 bind_all 按钮对设备进行 APPkey 的 bind（bind 过程会根据设备 composition data 上报的 model 进行 bind）。



如下图截取的 Provision 流程交互数据进行说明如下说明：

1) 开始 provision，provisioner 发送
2) public key 交互
3) check confirm
4) send provision data（net_key/nkey_index/IV_updata_flag/IV_index/unicast_addr）
5) 对 proxy 添加 白名单



Bind 流程说明：

Bind 是在 provision 后根据读取的设备的 composition data 的信息，将 APP key 与 device 的所有 model 进行 bind 的过程。这个过程会根据 model 的数量决定 bind 的时间（20s 只有）。因此在天猫精灵等平台都将 bind 进行了优化处理，接下来的 fastbind 将会进行介绍。

Bind 前首先需要获取 device 的 composition data。Composition data 的详细解析格式请参考<Mesh_v1.0>的<4.2.1 Composition Data >

```
<0322>19:20:53:228 [INFO]:(KEYBIND)start key bind and the appkey is
: 60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48
<0323>19:20:53:239 [INFO]:(KEYBIND)SEND: get composition data
<0324>19:20:53:252 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 02 00 80 08 ff
<0325>19:20:53:262 [INFO]:(Basic)the mesh access tx cmd is 0x0880 : ff
<0326>19:20:53:431 [INFO]:(Basic)rx segment all packetes received
<0327>19:20:53:442 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x2 :
  02 00 11 02 01 00 32 39  69 00 07 00 00 00 11 01  00 00 02 00 03 00 04 00  00 fe 01 fe 00 ff 01 ff
  00 10 02 10 04 10 06 10  07 10 00 13 01 13 03 13  04 13 11 02 00 00 00 00  02 00 02 10 06 13
<0328>19:20:53:454 [INFO]:(cmd_rsp)Status Rsp_____________:
  02 00 01 00 02 00 11 02  01 00 32 39 69 00 07 00  00 00 11 01 00 00 02 00  03 00 04 00 00 fe 01 fe
  00 ff 01 ff 00 10 02 10  04 10 06 10 07 10 00 13  01 13 03 13 04 13 11 02  00 00 00 00 02 00 02 10
  06 13
```

根据获取到的 element 和 model 的对应信息，后下发 bind 命令根据 model 逐个 bind。

```
<0330>19:20:53:490 [INFO]:(KEYBIND)SEND: appkey add ,0x0 is the appkey index
: 60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48
<0331>19:20:53:502 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 00 00 00 00 60 96 47 71 73 4f
<0332>19:20:53:514 [INFO]:(Basic)the mesh access tx cmd is 0x0000 : 00 00 00 60 96 47 71 73 4f bd 76 e
<0333>19:20:53:872 [INFO]:(Basic)rc data layer upper:segment tx success, map in ACK is: 03 00 00 00
<0334>19:20:53:883 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x380 : 80 03 00 00 00
<0335>19:20:53:893 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 03 00 00 00 00
<0336>19:20:53:907 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x0000 rsp_max 1, rsp_cnt 1
<0337>19:20:53:919 [INFO]:(KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0002
<0338>19:20:53:932 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 02 00
<0339>19:20:53:945 [INFO]:(Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 02 00
<0340>19:20:54:032 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0341>19:20:54:044 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 3e 00 02 00 00 00 02 00
<0342>19:20:54:063 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0343>19:20:54:073 [INFO]:(KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0003
<0344>19:20:54:085 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 03 00
```

```
<0448>19:20:57:152 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 03 0
<0449>19:20:57:166 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 3e 00 03 00 00 00 06 13
<0450>19:20:57:187 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0451>19:20:57:218 [INFO]:(KEYBIND)SEND: mesh keybind event success
```

20.2 配置操作

20.2.1 Key add /bind 操作

APPKey add 命令格式解析:

CMD-cfg_appkey_add_001= a3 ff 00 00 00 00 02 00 07 00 00 00 00 00 60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48

1		2	3	4	5	7	8	9	10	11	12		
Flag		nk_idx	ak_idx	reliable retry count	eliable rsp_max	dst	op	para0	para1	para2	para3	para4	para5
								AppKeyIndex			app key		
a3	ff	0000	0000	02	00	0700	00	000000			60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48		

命令解析:

Flag:为 telink 私有化定义，目的为 USB or UART 通信的头包识别，UART 为 E8FF, USB 为 A3FF。

NK_idx: network key index

Ak_idx: APP key index

Reliable retry cnt: 应用层 retry(上位机发出命令后如果收不到回复则重发次数)

Reliable resp_max:设置回复的节点的个数

Dst: 目的地址填充

Op: sig mesh 规范定义的标准命令码，可以参考< Mesh_v1.0>，命令码不是固定长度

文档里的< 4.3.4 Messages summary >

[9:12]: 传参部分，

填充数据为参考文档< Mesh_v1.0>的< 4.3.2.37 Config AppKey Add >

Field	Size (octets)	Notes
NetKeyIndexAndAppKeyIndex	3	Index of the NetKey and index of the AppKey
AppKey	16	AppKey value

Key bind 命令格式解析:

CMD-cfg_appkey_bind_001 = a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 00 10

命令解析:

Flag:为 telink 私有化定义，目的为 USB or UART 通信的头包识别，UART 为 E8FF, USB 为 A3FF。

NK_idx: network key index

Ak_idx: APP key index

Reliable retry cnt: 应用层 retry(上位机发出命令后如果收不到回复则重发次数)

Reliable resp_max:设置回复的节点的个数

Dst: 目的地址填充

Op: sig mesh 规范定义的标准命令码，可以参考< Mesh_v1.0>，命令码不是固定长度

文档里的< 4.3.4 Messages summary >

[9:12]: 传参部分，light HSL 控制命令填充数据为参考文档< Mesh_v1.0>

的< 4.3.2.46 Config Model App Bind >

传参解析格式参考:

Field	Size (octets)	Notes
ElementAddress	2	Address of the element
AppKeyIndex	2	Index of the AppKey
ModelIdentifier	2 or 4	SIG Model ID or Vendor Model ID

20.2.2 订阅设置

CMD-cfg_sub_add =a3 ff 00 00 00 00 00 01 02 00 80 1b 02 00 01 c0 00 10

或者参考《4.4.2 分组控制(即 subscription 的功能演示)》的组号控制。

20.2.3 publish 设置

CMD-cfg_pub_set_sig_2s =a3 ff 00 00 00 00 00 00 02 00 03 02 00 01 00 00 00 ff 14 15 00 10

或者参考《4.4.3 通过 UI 配置某一节点的详细参数》的"GetPub_S"按钮控制。

20.2.4 Relay/Friend 功能设置

Relay:a3 ff 00 00 00 00 02 01 07 00 80 27 01

```
<0000>11:44:52:572 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 27 01
<0001>11:44:52:573 [INFO]:(Basic)the mesh access tx cmd is 0x2780 : 01
<0002>11:44:52:679 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x2880 : 80 28 01 a0
<0003>11:44:52:681 [INFO]:(cmd_rsp)Status Rsp_____________: 0a 00 01 00 80 28 01 a0
<0004>11:44:52:685 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x2780 rsp_max 1, rsp_cnt 1
<0005>11:44:54:880 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 27 00
<0006>11:44:54:882 [INFO]:(Basic)the mesh access tx cmd is 0x2780 : 00
<0007>11:44:54:999 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x2880 : 80 28 00 c4
<0008>11:44:55:001 [INFO]:(cmd_rsp)Status Rsp_____________: 0a 00 01 00 80 28 00 c4
<0009>11:44:55:008 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x2780 rsp_max 1, rsp_cnt 1
```

Friend:a3 ff 00 00 00 00 02 01 07 00 80 10 01

```
<0000>11:44:15:977 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 10 01
<0001>11:44:15:977 [INFO]:(Basic)the mesh access tx cmd is 0x1080 : 01
<0002>11:44:16:080 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1180 : 80 11 01
<0003>11:44:16:081 [INFO]:(cmd_rsp)Status Rsp_____________: 0a 00 01 00 80 11 01
<0004>11:44:16:086 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1080 rsp_max 1, rsp_cnt 1
<0005>11:44:24:487 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 10 00
<0006>11:44:24:488 [INFO]:(Basic)the mesh access tx cmd is 0x1080 : 00
<0007>11:44:24:599 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1180 : 80 11 00
<0008>11:44:24:602 [INFO]:(cmd_rsp)Status Rsp_____________: 0a 00 01 00 80 11 00
<0009>11:44:24:607 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1080 rsp_max 1, rsp_cnt 1
```

Proxy:a3 ff 00 00 00 00 02 01 07 00 80 13 01

```
<0000>11:45:14:247 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 13 01
<0001>11:45:14:248 [INFO]:(Basic)the mesh access tx cmd is 0x1380 : 01
<0002>11:45:14:358 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1480 : 80 14 01
<0003>11:45:14:362 [INFO]:(cmd_rsp)Status Rsp_____________: 0a 00 01 00 80 14 01
<0004>11:45:14:367 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1380 rsp_max 1, rsp_cnt 1
```

或者参考《4.4.3 通过 UI 配置某一节点的详细参数》的"Relay"，"Friend"，"Proxy"按钮控制。

20.2.5 Heatbeat 设置

CMD-cfg_hb_pub_set_sig =a3 ff 00 00 00 00 00 00 02 00 80 39 01 00 ff 02 01 07 00 00 00

20.3 控制操作

20.3.1 控制 Generic model 实例

测试准备

- ✧ Provisionner dongle（烧录 8258_mesh_gw.bin ）
- ✧ 上位机工具（sig_mesh_tool.exe），选择 tl_node_gateway.ini
- ✧ Dongle_2#（烧录 mesh.bin）
- ✧ SDK 无需修改

测试说明

测试主要目的是实现对 Generic model 的控制实现，主要以实现 G_ONOFF_SET 命令测试。

并对 CMD send 与 ACK 的响应时间进行测试计算。

测试步骤

Step 1 将 dongle_1#烧录 gateway bin 后，插入电脑，同时打开上位机软件 sig_mesh_tool.exe。

Step 2 将 dongle_2#烧录 mesh 节点 bin。

Step 3 上位机将 mesh 节点添加入网

Step 4 按照如下命令格式发或者自己在上位机的 ini CMD 栏直接双击该条命令。

```
CMD-g_on_03                    =e8 ff 00 00 00 00 00 00 03 00 82 02 01 00
```

Step 5 上位机显示发送成功后，如下图所示。

```
.. g_on_03
<0011>15:53:49:075 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 01 00
<0012>15:53:49:199 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 00 01 0a
<0013>15:53:49:211 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a
.. g_off_03
<0014>15:54:54:035 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 00 00
<0015>15:54:54:094 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 01 00 0a
<0016>15:54:54:107 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 01 00 0a
.. g_on_03
<0017>15:54:57:898 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 01 00
<0018>15:54:57:955 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 00 01 0a
<0019>15:54:57:969 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a
.. g_off_03
<0020>15:55:01:740 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 00 00
<0021>15:55:01:938 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 01 00 0a
<0022>15:55:01:952 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 01 00 0a
```

Step 6 如上图所示<0011> 所示，CMD 发送时间与 rsp 时间间隔为:199 – 075 = 124ms。Gateway 与 node 采用 adv 控制的方式，受网络影响命令 ack 回复的时间会有差异。

Step 7 如果控制的目的地址是组播或者广播地址，则 node 在收到命令后回复 ACK 前会加随机延时，以便多设备回复消息尽可能的避让。如下图所示广播地址的时候，CMD 与 rsp 的时间间隔为:

<0023> 12:390 – 11:752 = 538 ms

```
.. g_off
<0023>16:14:11:752 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 02 00 00
<0024>16:14:12:390 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 00
<0025>16:14:12:405 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00
.. g_on
<0026>16:14:14:328 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 02 01 00
<0027>16:14:15:096 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 04 00 01 0a
<0028>16:14:15:129 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a
```

20.3.2 CTL model

测试准备

- ✧ Provisionner dongle（烧录 8269_mesh_gw.bin or 8258_mesh_gw.bin ）
- ✧ 上位机工具（sig_mesh_tool.exe），选择 tl_node_gateway.ini
- ✧ Dongle_2#（烧录 mesh.bin）
- ✧ SDK 需要修改 #define LIGHT_TYPE_SEL LIGHT_TYPE_CT

测试说明

测试主要目的是实现对 CTL model 的控制实现，主要以实现 LIGHT_CTL_SET 命令测试。

测试步骤

Step 1 将 dongle_1#烧录 gateway bin 后，插入电脑，同时打开上位机软件 sig_mesh_tool.exe。

Step 2 将 dongle_2#烧录 mesh 节点 bin。

Step 3 上位机将 mesh 节点添加入网

Step 4 按照如下命令格式发或者自己在上位机的 ini CMD 栏直接双击该条命令。

```
CMD-light_ctl_set          =e8 ff 00 00 00 00 00 00 ff ff 82 5e 01 00 20 4e 00 00 00
```

上位机显示发送成功后，如下图所示。

```
.. light_ctl_set
<0000>16:29:46:909 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 5e 01 00 20 4e 00 00 00
<0001>16:29:47:513 [INFO]:(cmd_rsp)Status Rsp_____________: 03 00 02 00 82 60 01 00 20 4e
<0002>16:29:47:521 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 60 01 00 20 4e
```

20.3.3 HSL model

Hsl model 在配网后会分配 3 个 element 地址，主 element 用于 lightness, generic model 控制以及 configuration model 配置等，element2 是 hue 控制，element3 是 saturation 控制。

测试准备

- ✧ Provisionner dongle（烧录 8269_mesh_gw.bin or 8258_mesh_gw.bin ）
- ✧ 上位机工具（sig_mesh_tool.exe），选择 tl_node_gateway.ini
- ✧ Dongle_2#（烧录 mesh.bin）
- ✧ SDK 需要修改 #define LIGHT_TYPE_SEL LIGHT_TYPE_HSL

测试说明

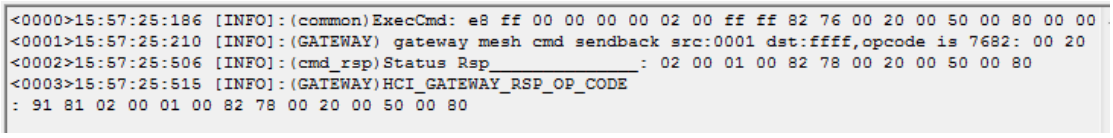
测试主要目的是实现对 HSL model 的控制实现，主要以实现 LIGHT_HSL_SET 命令测试。

测试步骤

Step 1　将 dongle_1#烧录 gateway bin 后，插入电脑，同时打开上位机软件 sig_mesh_tool.exe。
Step 2　将 dongle_2#烧录 mesh 节点 bin。
Step 3　上位机将 mesh 节点添加入网
Step 4　按照如下命令格式发或者自己在上位机的 ini CMD 栏直接双击该条命令。

```
CMD-light_hsl_set          =a3 ff 00 00 00 00 00 00 ff ff 82 76 01 00 00 50 00 80 00
```

上位机显示发送成功后，如下图所示。



20.3.4 Vendor model

自定义 OP 操作

测试准备

- ✧ Provisionner dongle（烧录 8269_mesh_master_dongle.bin ）
- ✧ 上位机工具（sig_mesh_tool.exe）
- ✧ Dongle_2#（烧录 8258_mesh.bin）
- ✧ Sdk 需要做出修改，按照<测试说明进行配置>

SDK 修改说明

1) 在 vendor_model.h 里配置

```
#define VD_USER_ONOFF_GET                    0xE1
#define VD_USER__ONOFF_SET                   0xE2
#define VD_USER__ONOFF_SET_NOACK      0xE3
#define VD_USER__ONOFF_STATUS                0xE4
```

2) 在 vendor_model.c 进行声明

```
    {VD_USER_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
cb_vd_light_onoff_set, VD_USER_ONOFF_STATUS},
    {VD_USER_ONOFF_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
cb_vd_light_onoff_get, VD_USER_ONOFF_STATUS},
    {VD_USER_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
cb_vd_light_onoff_set, STATUS_NONE},
   {VD_USER_ONOFF_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C,
cb_vd_light_onoff_status, STATUS_NONE},
```

3) main_loop 里添加处理

测试说明

Telink 的 sdk 根据客户的使用情况做出了如下的限定值:

1) 一个 friend node 可以最多挂靠 16 个 LPN，默认为 2 个，可以修改 MAX_LPN_NUM 为 16 个
2) Friend node 缓存的 1 个 LPN 节点的数据可以支持长包，长包最大字节为 41byte。（APP 在发送 data 的时候最大传参是 41byte）。

20.3.5 Gateway 发长包到 LPN

测试准备

- ✧ Provisionner dongle（烧录 8258_mesh_gw.bin ）
- ✧ 上位机工具（sig_mesh_tool.exe），选择 tl_node_gateway.ini
- ✧ Dongle_2#（烧录 LPN.bin）
- ✧ Sdk 需要做出修改，按照<测试说明进行配置>

测试说明

Telink 的 sdk 根据客户的使用情况做出了如下的限定值:

1) 一个 friend node 可以最多挂靠 16 个 LPN，默认为 2 个，可以修改 MAX_LPN_NUM 为 16 个
2) Friend node 缓存的 1 个 LPN 节点的数据可以支持长包，长包最大字节为 41byte。（APP 在发送 data 的时候最大传参是 41byte）。

为了调试方便 sdk 做出如下修改:

1) 调试方便需要 DEBUG_SUSPEND =1，（不进入低功耗模式）
2) 需要在 cb_vd_light_onoff_set()回调里添加如下的信息。

```
u8 debug_fn_reciver_data[64];
u8 debug_fn_cnt;
    memset(debug_fn_reciver_data,0,sizeof(debug_fn_reciver_data));
    memcpy(debug_fn_reciver_data,par,par_len);
```

```
u8 debug_fn_reciver_data[64];
u8 debug_fn_cnt;
int cb_vd_light_onoff_set(u8 *par, int par_len, mesh_cb_fun_par_t *cb_par)
{
    debug_fn_cnt++;
    int err = -1;
    int pub_flag = 0;
    //model_g_light_s_t *p_model = (model_g_light_s_t *)cb_par->model;
    vd_light_onoff_set_t *p_set = (vd_light_onoff_set_t *)par;

    memset(debug_fn_reciver_data,0,sizeof(debug_fn_reciver_data));
    memcpy(debug_fn_reciver_data,par,par_len);
```

测试步骤

Step 1 将 dongle_1#烧录 gateway bin 后，插入电脑，同时打开上位机软件 sig_mesh_tool.exe。

Step 2 将 dongle_2#烧录 LPN 节点 bin。

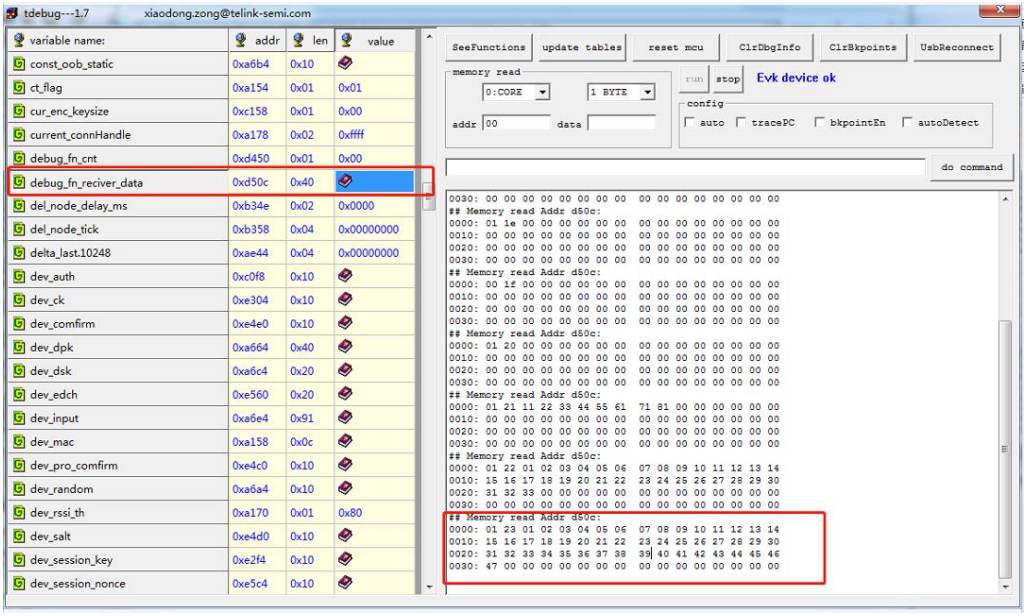
Step 3 上位机将 LPN 添加入网

Step 4 在现有的 vendor_on 命令和 vendor_off 命令控制 LPN 节点，LPN 可以被正常控制

Step 5 将 vendor_on 的命令的传参加长共计 41byte 的传参，然后发送

```
CMD-vendor_on      =a3 ff 00 00 00 00 00 00 ff ff c2 11 02 c4 02 01 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
```

Step 6 上位机显示发送成功后，使用 tdebug 工具查看 debug_fn_reciver_data[64]的内如是否正确。


tdebug---1.7
xiaodong.zong@telink-semi.com
variable name:
addr
len
value
const_oob_static 0xa6b4 0x10
ct_flag 0xa154 0x01 0x01
cur_enc_keysize 0xc158 0x01 0x00
current_connHandle 0xa178 0x02 0xffff
debug_fn_cnt 0xd450 0x01 0x00
debug_fn_reciver_data 0xd50c 0x40
del_node_delay_ms 0xb34e 0x02 0x0000
del_node_tick 0xb358 0x04 0x00000000
delta_last.10248 0xae44 0x04 0x00000000
dev_auth 0xc0f8 0x10
dev_ck 0xe304 0x10
dev_comfirm 0xe4e0 0x10
dev_dpk 0xa664 0x40
dev_dsk 0xa6c4 0x20
dev_edch 0xe560 0x20
dev_input 0xa6e4 0x91
dev_mac 0xa158 0x0c
dev_pro_comfirm 0xe4c0 0x10
dev_random 0xa6a4 0x10
dev_rssi_th 0xa170 0x01 0x80
dev_salt 0xe4d0 0x10
dev_session_key 0xe2f4 0x10
dev_session_nonce 0xe5c4 0x10
SeeFunctions
update tables
reset mcu
ClrDbgInfo
ClrBkpoints
UsbReconnect
memory read
0:CORE
1 BYTE
addr 00
data
run
stop
Evk device ok
config
auto
tracePC
bkpointEn
autoDetect
do command
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 01 1e 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 00 1f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 01 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 01 21 11 22 33 44 55 61 71 81 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 01 22 01 02 03 04 05 06 07 08 09 10 11 12 13 14
0010: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
0020: 31 32 33 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory read Addr d50c:
0000: 01 23 01 02 03 04 05 06 07 08 09 10 11 12 13 14
0010: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
0020: 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
0030: 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00