应用文档:

泰凌 SIG Mesh SDK 开发手册

AN-17120401-C2

Ver 1.1.0

2018/7/30

简介:

本文档为泰凌 SIG Mesh SDK 的开发手册。

TELINK SEMICONDUCTOR

Published by
Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,
Zhangjiang Hi-Tech Park, Shanghai, China

Legal Disclaimer

Telink Semiconductor reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein or in any other disclosure relating to any product.

Telink Semiconductor does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others

The products shown herein are not designed for use in medical, life-saving, or life-sustaining applications. Customers using or selling Telink Semiconductor products not expressly indicated for use in such applications do so entirely at their own risk and agree to fully indemnify Telink Semiconductor for any damages arising or resulting from such use or sale.

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

版本历史

版本	主要改动	日期	作者
1.0.0	初始版本	2017/12	SQF, XHW, Cynthia
1.1.0	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, 6 SWITCH 工程介绍	2018/7	YWX, Cynthia

目录

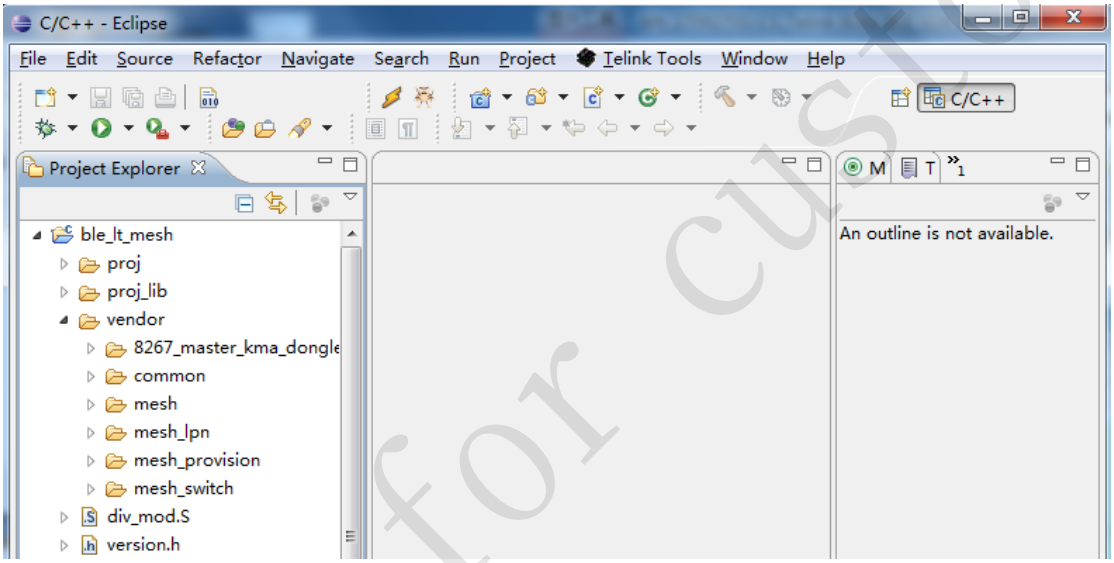
1. SDK 概述

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 8269。负责上位机通信的 IC 为 8267(或 8269)。

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，显示的文件架构如下图所示。有 3 个主要的顶层文件夹：proj，proj_lib，vendor。



文件架构图

- **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，user 无法看到源文件，如 liblt_8269_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。
- **vendor**：用于存放用户应用层代码。目前 vendor 目录下有：

  8267_master_kma_dongle：上位机测试使用固件，配合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 中共用的模块，例如 led 部分，出厂初始化 ，测试命令等模块。

  mesh/mesh_lpn/mesh_provision/mesh_switch 这四个文件夹的结构一样，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。具体的函数接口部分详见第三章中关于各个文件的介绍。

1.2 入口函数

```
int main (void) {
  FLASH_ADDRESS_CONFIG;
  cpu_wakeup_init();//mcu硬件初始化，用户不用关心
  clock_init();//时钟初始化，用户可以在user_config.h里面修改配置
  set_tick_per_us(CLOCK_SYS_CLOCK_HZ/1000000);//时钟初始化
  gpio_init();//gpio初始化，用户可以在user_config.h里面修改配置
  rf_drv_init(CRYSTAL_TYPE);//rf初始化，用户不用关心
  user_init ();//ble初始化&系统初始化&用户自定义初始化
   irq_enable();//使能全局中断
  while (1) {
      #if (MODULE_WATCHDOG_ENABLE)
      wd_clear(); //clear watch dog
      #endif
      main_loop ();//ble收发处理&低功耗处理&用户自定义处理
  }
}
```

1.3 BLE 协议栈收发包处理

在 lib 中实现，用户不用关心。如果有需要，用户可以参考 my_Attributes_provision[]/my_Attributes_proxy[]/my_Attributes[]定义，自己订制 BLE service。

1.4 Mesh 应用收发处理

1.4.1 发包

(1) 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。

(2) 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。

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。

(3) 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

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：

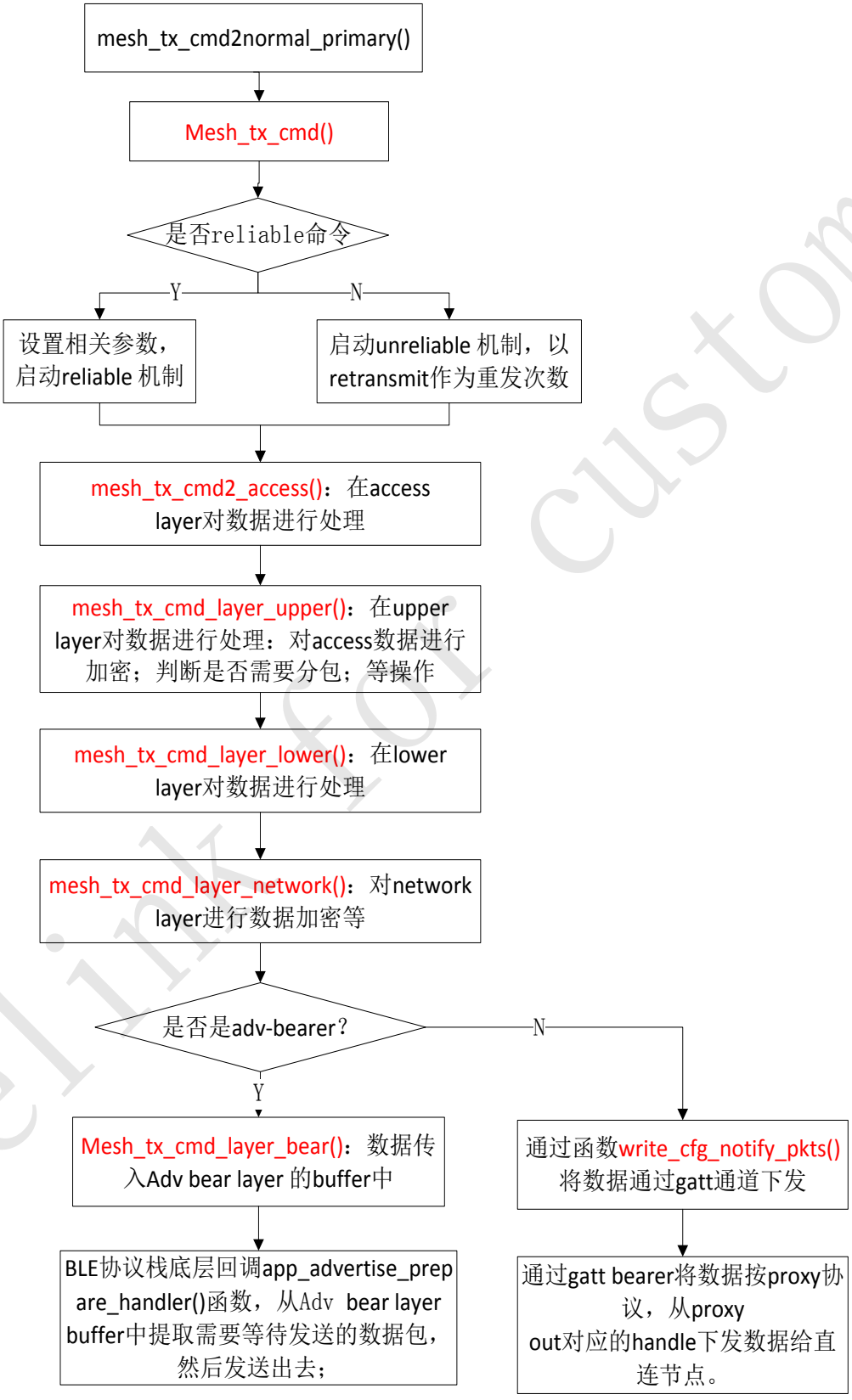
- ✧ UART: blc_register_hci_handler (blc_rx_from_uart, blc_hci_tx_to_uart);
- ✧ USB: blc_register_hci_handler (app_hci_cmd_from_usb, blc_hci_tx_to_usb);

1.4.2 收包

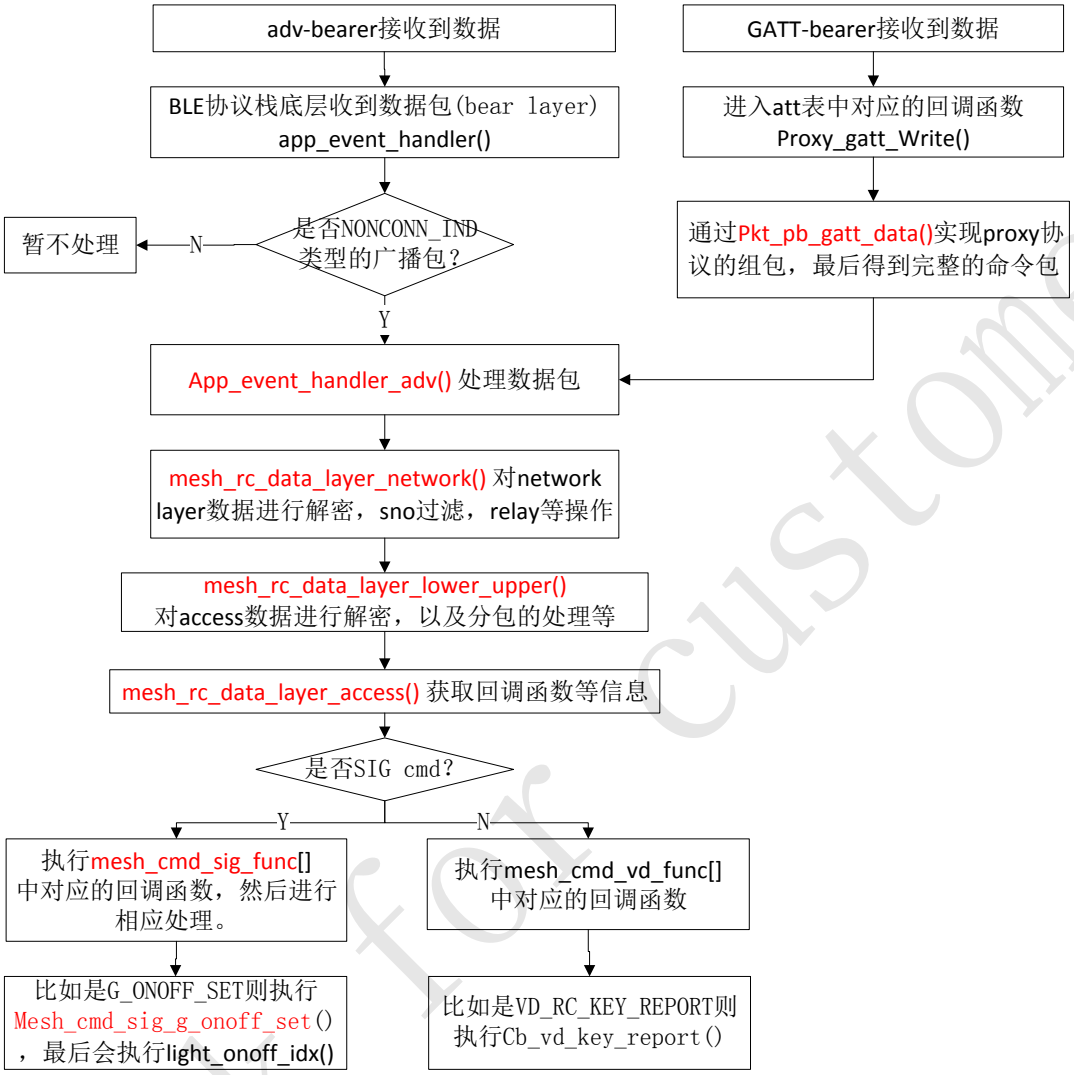
收包处理见 1.6 收包流程。

1.5 发包流程简介

注：红色字体为 library 函数。



1.6 收包流程简介



(1) 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[]。比如开关灯，收到这个命令后，按 SIG mesh spec 流程，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)。

(2) 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()。

1.7 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；

另外一种是通过 gatt-bearer，通过建立 BLE 连接实现通信。

2. 全局配置文件说明

2.1 mesh_config.h

(1) 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

(2) LIGHT_TYPE_SEL

选择灯的类型，目前几种类型的灯是互斥的关系

LIGHT_TYPE_NONE
LIGHT_TYPE_CT: Light CTL
LIGHT_TYPE_HSL: Light HSL
LIGHT_TYPE_XYL : Light xyL
LIGHT_TYPE_POWER : Generic Power Level

(3) MD_LIGHT_CONTROL_EN

Lighting Control model 开关，默认关闭

(4) MD_MESH_OTA_EN

Mesh_OTA_Model 接口，mesh 默认开启

(5) MD_ONOFF_EN

Generic OnOff model，默认开启

(6) MD_LEVEL_EN

Generic Level model，默认开启

(7) MD_DEF_TRANSIT_TIME_EN

Generic Default Transition Time model，默认支持

(8) MD_POWER_ONOFF_EN

Generic Power OnOff Model，默认关闭

(9) MD_TIME_EN

Time Model，默认关闭

(10) MD_SCENE_EN

Scene Model，默认关闭

(11) MD_SCHEDULE_EN

Schedule Model，默认关闭

(12) MD_CLIENT_EN

client model 的开关。对于节点一般是不需要 client model，所以为了节省 Ram，灯端默认关闭所有的 client model

(13) 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。

(14) TRANSITION_TIME_DEFAULT_VAL

默认的渐变参数。在控制灯的命令中没带渐变参数时，采用默认的渐变参数。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。

(15) FEATURE_FRIEND_EN

设置是否支持 Friend feature。

(16) FEATURE_LOWPOWER_EN

设置是否支持 Low Power feature。

(17) FEATURE_PROV_EN

Provision 开关，需打开

(18) FEATURE_RELAY_EN

设置是否支持 Relay feature

(19) FEATURE_PROXY_EN

设置是否支持 Proxy feature

(20) MAX_LPN_NUM

设置一个 friend 节点支持几个 low power 节点，目前为 2

(21) SEND_STATUS_WHEN_POWER_ON

设置上电时是否发亮度状态包

2.2 app_mesh.h

为方便调试，设备支持通过串口打印 log，log 分级信息如下：

TL_LOG_LEVEL_DISABLE:禁止

TL_LOG_LEVEL_ERROR:错误

TL_LOG_LEVEL_WARNING:警告

TL_LOG_LEVEL_INFO :普通 log

TL_LOG_LEVEL_DEBUG:调试 log

设备端 HCI_ACCESS 选为 HCI_USE_UART 后，即可通过串口打印 LOG。如果不需要 LOG，LogMsgModuleDlg_and_buf 函数前面直接 return 1 即可。

下图是设备端收到开关灯的串口打印 log.

```
(sdk) rcv network cmd 0x0282  01 09 cc

(sdk) rcv network cmd 0x0282  00 0a cc
```

2.3 mesh_common.c 文件介绍

2.3.1 int mesh_get_proxy_hci_type()

暂不需关心。

2.3.2 void mesh_tid_save(int ele_idx)

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。

2.3.3 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,
};
```

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。

2.3.4 void entry_ota_mode(void)

当收到 OTA start 命令后，会回调此函数。

2.3.5 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.

2.3.6 u8 proc_telink_mesh_to_sig_mesh(void)

检测 ota 前的 firmware 是否是 telink mesh，如果是，则执行参数初始化。

2.3.7 void mesh_ota_reboot_proc()

mesh ota 完成后，延时 1 秒再重启。

2.3.8 void updata_para_change_MTU()

在适当的时间申请更改 BLE 连接参数请求.

2.3.9 int app_advertise_prepare_handler (rf_packet_adv_t * p)

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。

(1) u8* 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) int mesh_adv_cmd_set(u8 *p_adv, u8 *p_bear)

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。

2.3.10 int app_l2cap_packet_receive (u16 handle, u8 * raw_pkt)

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。

2.3.11 int chn_conn_update_dispatch(u8 *p)

暂不关心。

2.3.12 void sim_tx_cmd_node2node()

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

2.3.13 void 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。

2.3.14 void ble_mac_init()

当 MAC 的参数位置，为非法值时，会随机生成一个 MAC 并保存起来。

2.3.15 void 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。

2.3.16 void mesh_scan_rsp_update_adr_primary(u16 adr)

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。

2.3.17 void uart_drv_init()/usb_bulk_drv_init()

串口和 USB 初始化，通过宏 HCI_ACCCESS 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。

2.3.18 void 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。

2.3.19 void 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。

2.3.20 void set_unprov_beacon_para(u8 *p_uuid ,u8 *p_info,u8 *p_hash,u8 *uri_para,u8 uri_len)

- 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。

2.3.21 void set_provision_adv_data(u8 *p_uuid,u8 *oob_info)

- 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。

2.3.22 void set_proxy_adv_data(u8 *p_hash,u8 *p_random,u8 node_identity)

- 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。

2.4 cmd_interface.h 文件介绍

2.4.1 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。

2.4.2 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。

2.4.3 int access_cmd_get_level(u16 adr,u32 rsp_max)

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。

2.4.4 int access_cmd_set_level(u16 adr, u8 rsp_max, s16 level, int ack)

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。

2.4.5 int access_set_lum(u16 adr, u8 rsp_max, u8 lum, int ack)

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。

2.4.6 int access_cmd_onoff(u16 adr_dst, u8 rsp_max, u8 onoff, int ack)

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。

2.5 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。

2.5.1 vendor 命令码的注册

(1) 在 mesh_cmd_vd_func[]数组中按 mesh_cmd_sig_func_t 成员要求直接添加新的命令码及相关信息。

```
typedef struct{
    u16 op;
    u16 status_cmd; // u16 for align
    u32 model_id_tx;
    u32 model_id_rx;
    cb_cmd_sig2_t cb;
```

```
    u32 op_rsp;         // -1 for no rsp and ensure 4BYTE align
}mesh_cmd_sig_func_t;
```

以现有的 VD_RC_KEY_REPORT 命令码为例：

op：VD_RC_KEY_REPORT。

status_cmd：该命令码是否是 status 命令？比如 G_ONOFF_STATUS、G_LEVEL_STATUS 等，如果是，则写 1，否则写 0。

model_id_tx：支持发送这个命令的 model id。

model_id_rx：支持接收这个命令的 model id。

cb：收到这个 opcode 执行的回调函数。

op_rsp：如果是 reliable 命令，则填写对应的 status opcode，比如 G_ONOFF_SET 对应的 op_rsp 是 G_ONOFF_STATUS。如果是 unreliable 命令，则该填写 STATUS_NONE。

(2) 增加相关 cb 中指向的回调函数。

(3) 如果是带 tid 命令，则还需要在 is_cmd_with_tid_vendor()函数中添加相关信息。详见后续对该函数的介绍。

2.5.2 int mesh_search_model_id_by_op_vendor(mesh_op_resource_t *op_res, u16 op, u8 tx_flag)

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。

2.5.3 int vd_cmd_key_report(u16 adr_dst, u8 rsp_max, u8 key_code)

上报按键事件。在 8269_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。

2.5.4 int is_cmd_with_tid_vendor(u8 *tid_pos_out, u16 op)

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。

2.6 mesh_test_cmd.c 文件介绍

该文件用于存放测试命令的实现。

2.7 配置默认 feature

(1) Mesh 节点，目前默认是支持以下 feature：friend、relay、proxy。详见 mesh_global_var_init()里面的相关配置。

```
model_sig_cfg_s_cps.page0.feature.relay = FEATURE_RELAY_EN;
model_sig_cfg_s_cps.page0.feature.proxy = FEATURE_PROV_EN;
model_sig_cfg_s_cps.page0.feature.frid = FEATURE_FRIEND_EN;
```

(2) friend、relay、proxy feature 默认是开启的，详见 mesh_global_var_init()里面的相关配置：model_sig_cfg_s.frid 、model_sig_cfg_s.relay 、model_sig_cfg_s.gatt_proxy。

当对应的 feature 是支持的情况下，可以设置上述几个值为默认开启或者关闭状态。目前默认是开启的状态。在 firmware 运行的过程中，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. 8269 MESH 工程介绍

3.1 app_config.h

(1) 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。

(2) UART_GPIO_SEL
设置 uart 的 IO

(3) ADC_ENABLE
设置是否使能 ADC

(4) PWM_R/ PWM_B/ PWM_B/ PWM_W
设置 PWM 对应的 IO

(5) SW2_GPIO/ SW1_GPIO
8269 Dongle 板的 2 个按键，调试使用。

3.2 app.c 文件介绍

3.2.1 广播包以及广播响应包的定制

(1) 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。

(2) 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[21 - 3];
}mesh_scan_rsp_t;
```

3.2.2 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。

3.2.3 int app_event_handler (u32 h, u8 *p, int n)

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。

(1) 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。

(2) 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。

(3) 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。

3.2.4 main_loop ()

- ✧ 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。

3.2.5 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 (RF_POWER_8dBm);设置发送功率为 8dbm。
- ✧ mesh_init_all()：sig mesh 相关的初始化

3.2.6 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。

3.3 app_att.c 文件介绍

3.3.1 pb_gatt_provision_out_ccc_cb(void *p)

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。

3.3.2 pb_gatt_Write (void *p)

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。

3.3.3 proxy_gatt_Write(void *p)

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。

3.3.4 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。

3.4 light.c 文件介绍

3.4.1 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
```

3.4.2 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] = {}
```

3.4.3 void light_dim_set_hw(int idx, int idx2, u16 val)

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

3.4.4 void rf_link_light_event_callback (u8 status)

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。

3.4.5 void proc_led()

led 指示灯事件轮询处理函数。

3.4.6 void show_ota_result(int result)

OTA 结束后的闪灯指示处理函数。

3.4.7 void show_factory_reset()

执行恢复出厂设置后的闪灯指示处理函数。

4. Provisioner（Gateway）工程介绍

4.1 provisioner 的功能介绍

4.1.1 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。

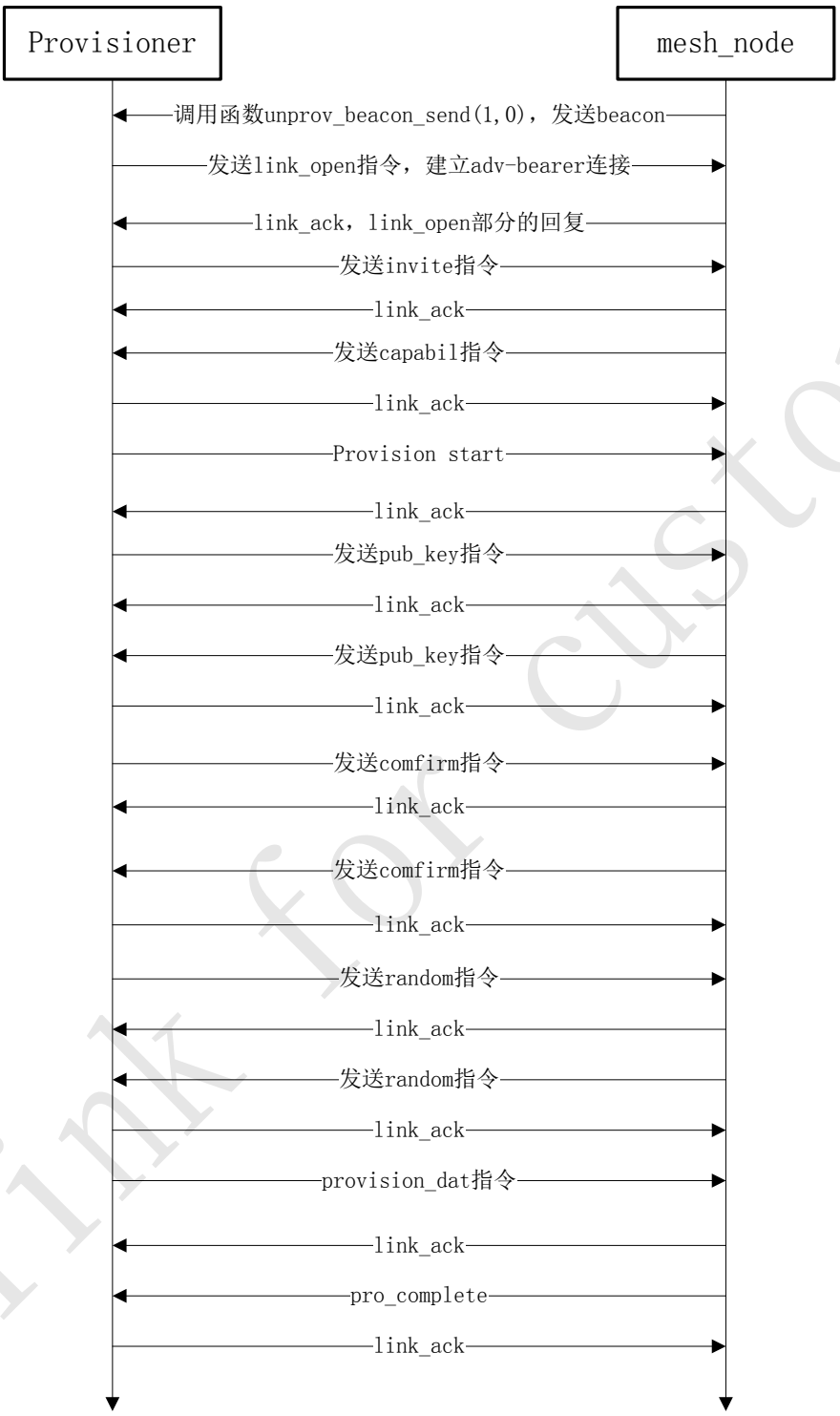
provision 的链接信道分为两种：一种是通过广播包实现 adv-bearer 信道上的通信，本章节主要介绍 adv-bearer 部分的内容。另外一种是采用 BLE 连接实现，通过 gatt-bearer 实现，gatt-bearer 的实现方式详见 7.2 章节中的 BLE 连接和加灯，以及 SIG_mesh 对应 Android 和 iOS 的 app 可实现对应的功能。

4.2 provisioner 的原理

4.2.1 provisioner 的命令交互

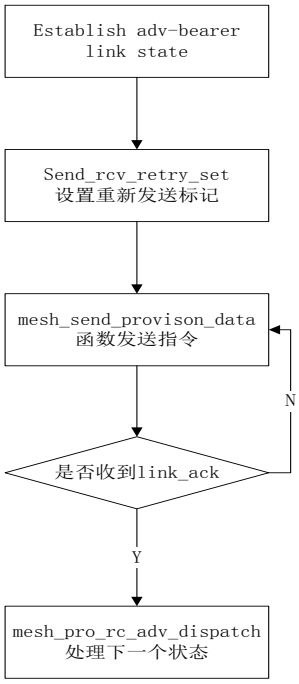
provisioner 通过 adv-bearer 将 unprov node（未配对节点）加入到网络中，采用蓝牙的 37,38,39 信道和 unprov node 进行通信，通过相互交互 random，key 之类的参数（详见 Mesh_v1.0.pdf 中 5.3）来交换网络参数和地址的分配最终达到将 unprov node 加入到网络中。

4.2.2 adv provisioner 的时序图

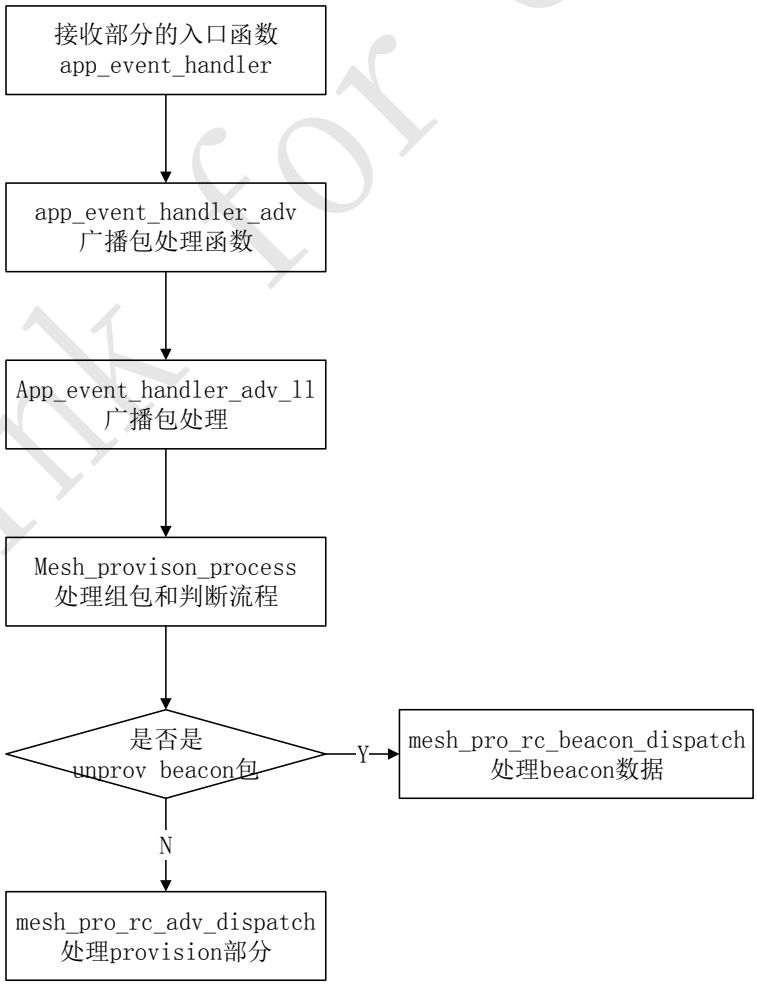


在 provision_dat 指令中包含 network key、IV index、unicast adr 这 3 个网络参数，实现组网的功能。

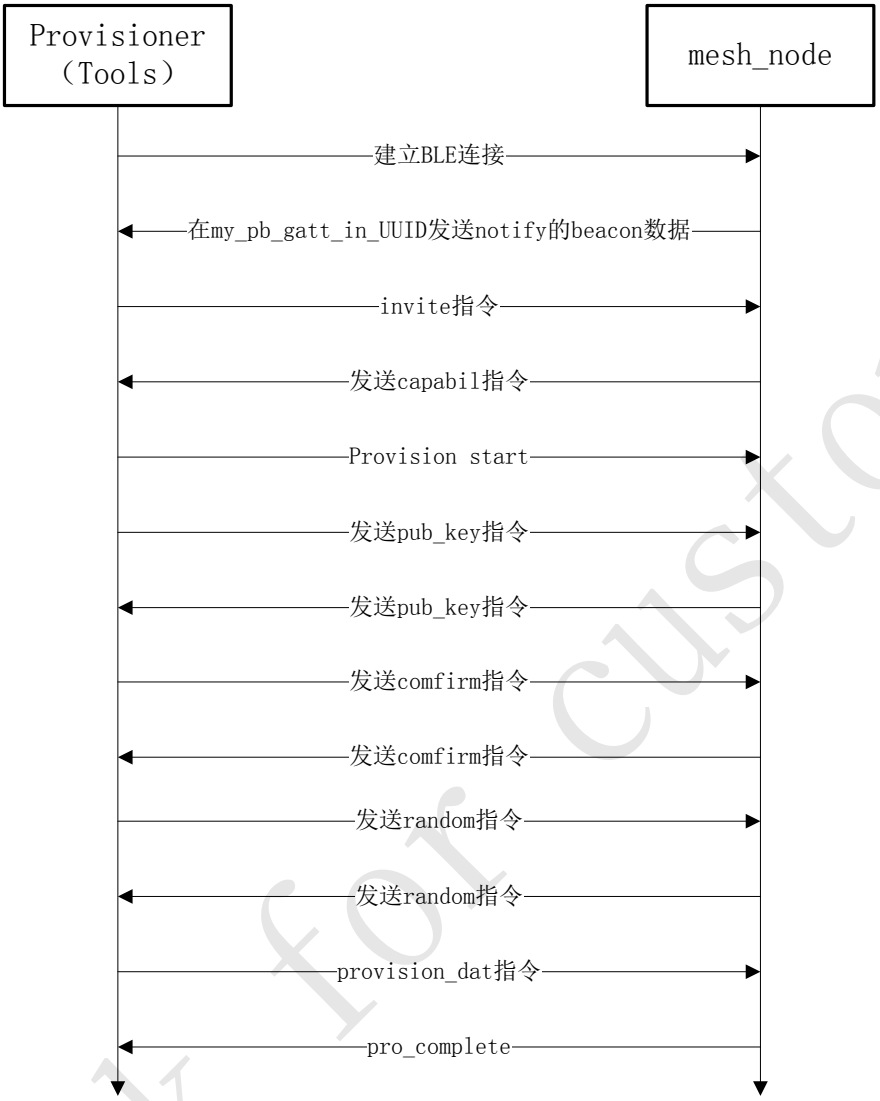
adv-provision 的函数发包部分的函数关系调用图：



adv-provision 的函数收包部分的函数关系调用图：

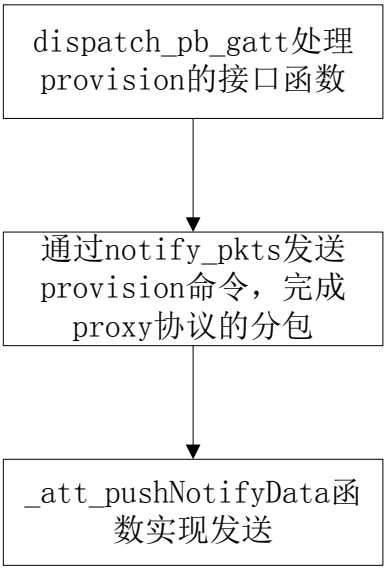


4.2.3 gatt provisioner 的时序图

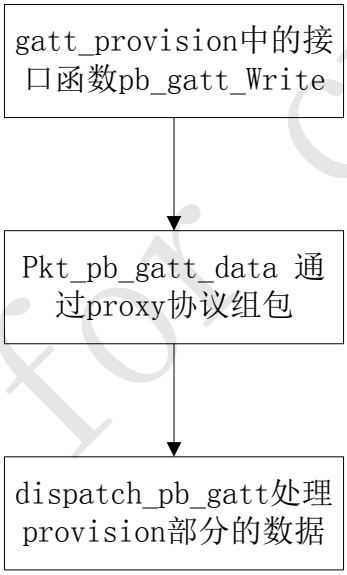


通过 gatt-provision 的方式能够更加快速的实现 provision 的功能，其中 int pb_gatt_Write (void *p)为 gatt_provision 部分的入口函数。

gatt_provision 的发包函数入口：



gatt_provision 的收包函数入口：



4.3 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。

4.3.1 广播包以及广播响应包的定制

参考《8269 MESH 工程介绍》。

4.3.2 fifo 部分配置

参考《8269 MESH 工程介绍》。

4.3.3 int app_event_handler (u32 h, u8 *p, int n)

参考《8269 MESH 工程介绍》。

4.3.4 main_loop ()

参考《8269 MESH 工程介绍》。

4.3.5 user_init()

参考《8269 MESH 工程介绍》。

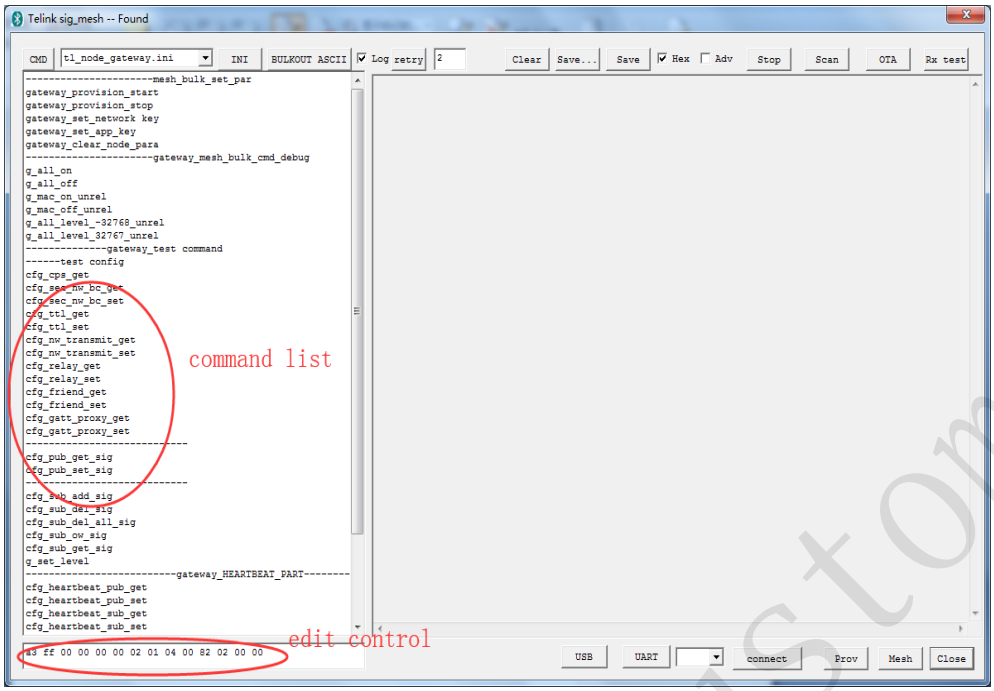
4.3.6 proc_ui()

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。

4.4 provision 如何实际操作

4.4.1 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。



tl_node_gateway.ini 中命令列表的格式如下，用户可以根据需求编辑或添加：

前 2 个字节是标识符，后面参数结构如下：

```
typedef struct{
    u16 nk_idx;     //netkey index
    u16 ak_idx;     //app_key index
    u8 retry_cnt;    // 应用层需要retry的次数，适用于需要回复的命令
    u8 rsp_max;      // 表示需要回复的节点的个数
    u16 adr_dst; // 目标地址
    u16 op;       //操作码
    u8 par[MESH_CMD_ACCESS_LEN_MAX]; //parameter after opcode
}mesh_bulk_cmd_par_t;
```

以全开 g_all_on 为例：

CMD-g_all_on = e8 ff 00 00 00 00 00 00 ff ff 82 02 01 00

e8 ff：标识符

00 00：network_key index

00 00：App_key index

00：应用层不做 retry

00：不设置回复的节点的个数

ff ff：目标地址为广播地址

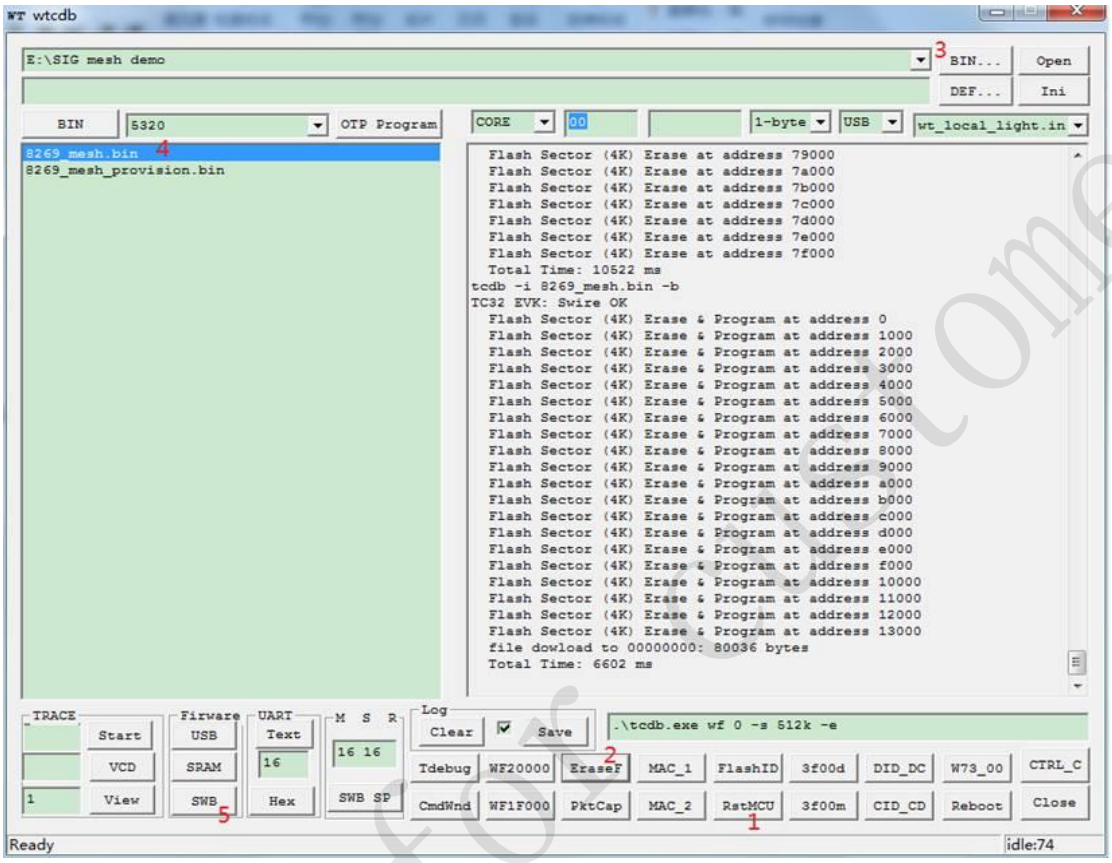
82 02： generic onoff 命令

01：表示开

00：tid

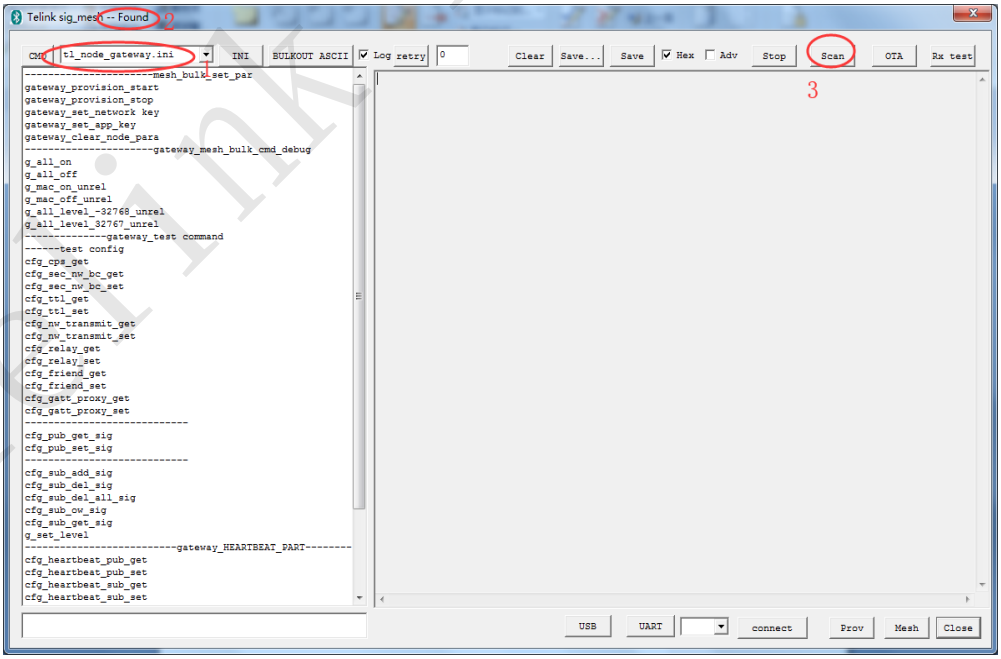
4.4.2 节点的烧录

烧录 2 个 8269 dongle：1 个 8269 provisioner 节点、1 个 dongle 节点，按照以下图示步骤进行操作。

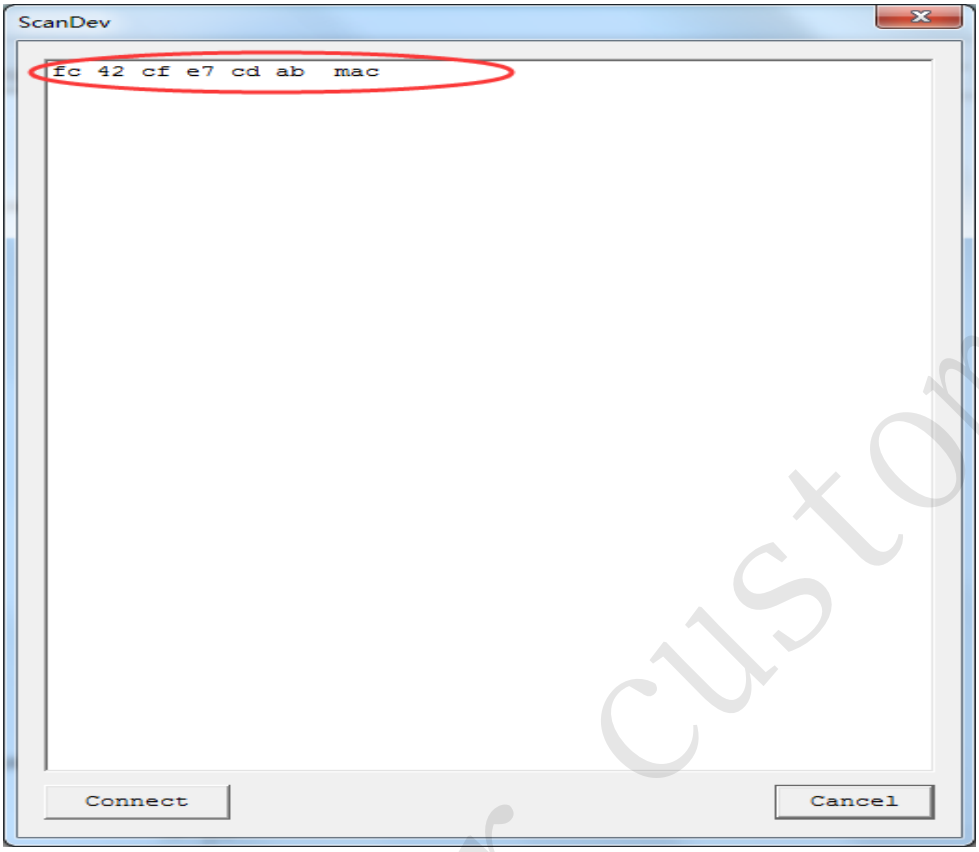


4.4.3 provisioner 加灯

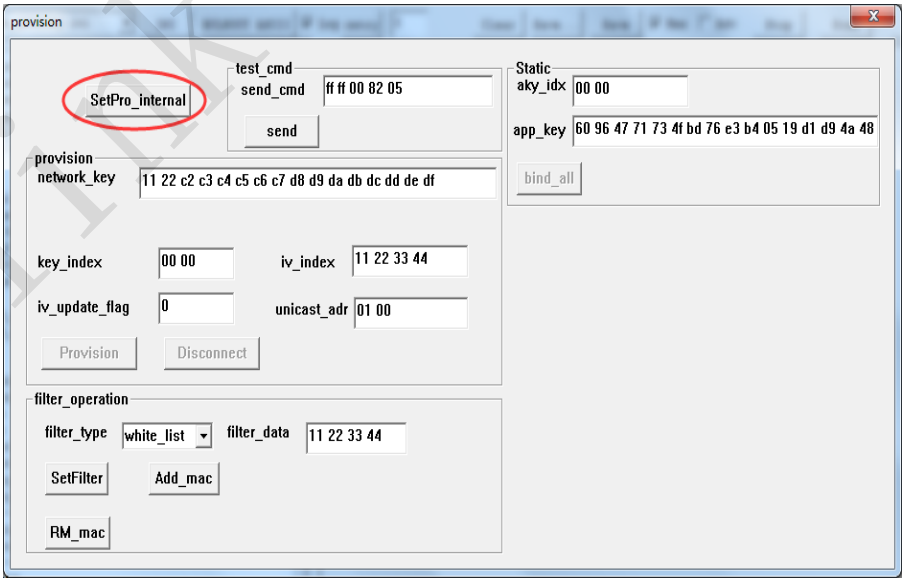
(1) provisioner dongle 插到 USB 口。打开“SIG_MESH_TOOL”选择 tl_node_gateway.ini，标题栏显示 Found 说明找到 gateway 设备（provisioner）。8269 dongle 上电，然后点击 scan 搜索设备。



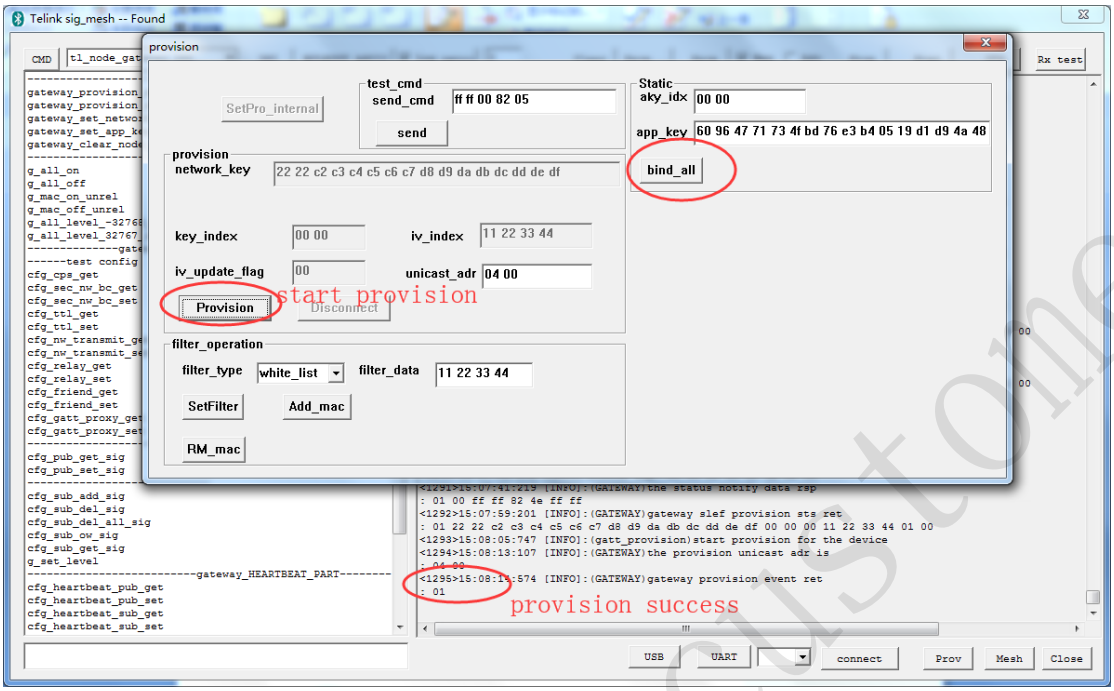
(2) 搜索到的设备会在设备列表显示出来，双击选择需要 provision 的设备。



(3) 点击 Prov 进入 provision 界面，若 SetPro Internal 按钮是使能的，说明 gateway 本身还未配置网络参数，设置好 network_key 等参数后点击“SetPro Internal”设置 gateway 本身参数。若 SetPro Internal 按钮禁止，表示 gateway 自身已配置参数，跳过此步骤即可。

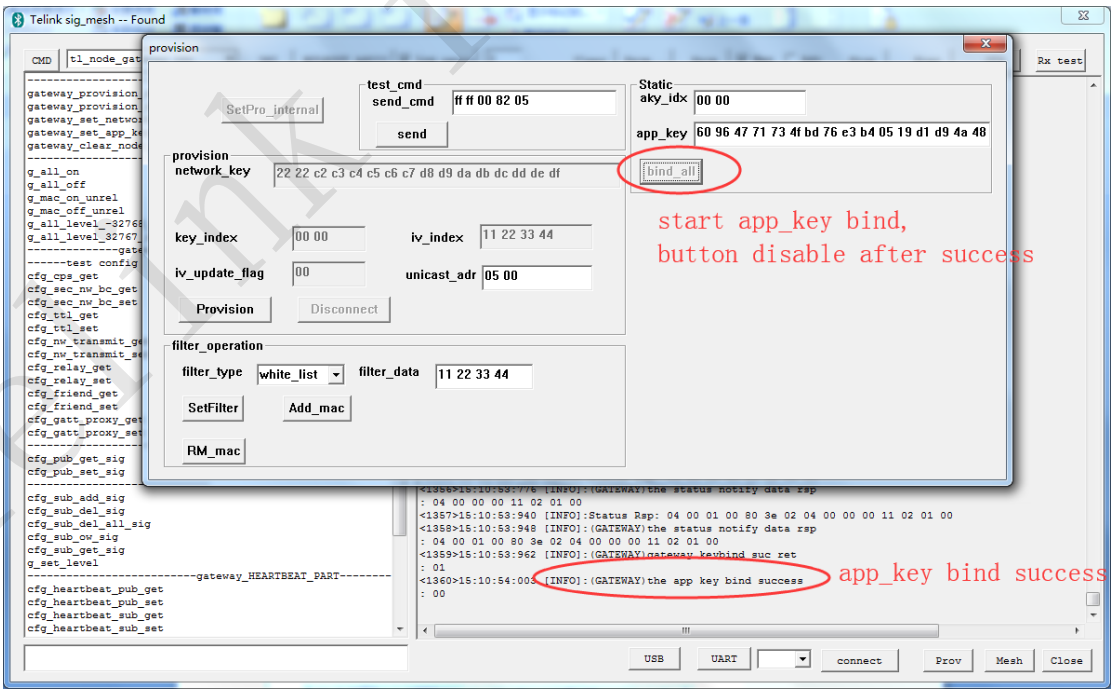


(4) 点击 Provision 按钮进行 provision。provision 过程中主界面的会打印相关 log，provision 成功后 bind_all 按钮使能。



4.4.4 绑定 app_key

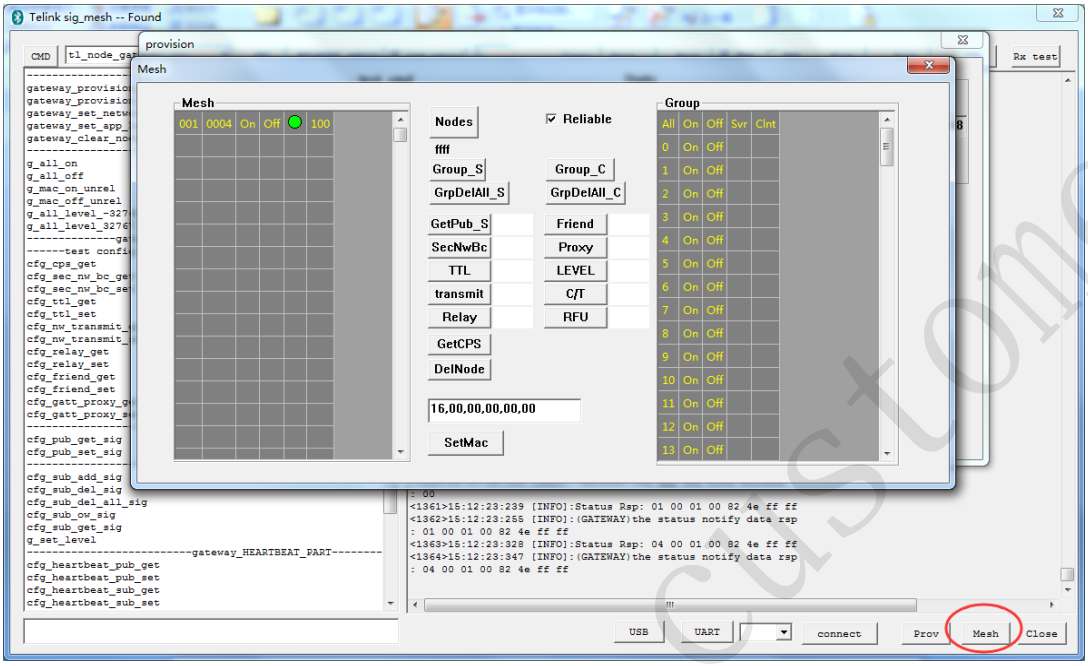
Provision 完成后，还需要为 model 绑定 app_key。点击 bind_all 为 model 绑定 app_key。



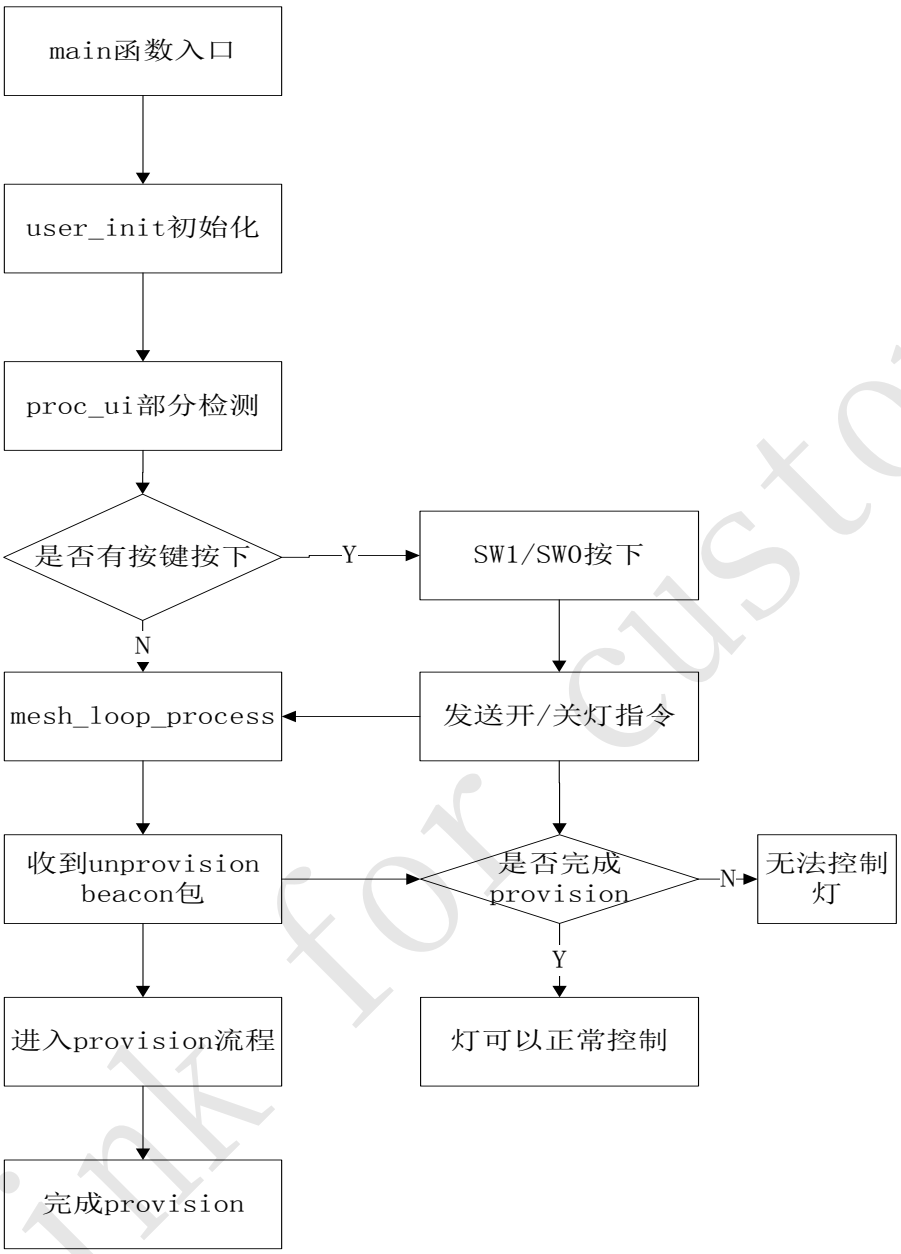
目前为了节省时间，对绑定的 app_key 做了过滤，默认绑定的 app_key 详见 Mesh_common.c 文件中 master_filter_list[]，用户可根据需要自行修改。

4.4.5 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。



4.4.6 provisioner 的控制流程图



5. Mesh LPN 工程介绍

5.1 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。

5.1.1 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。

5.1.2 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。

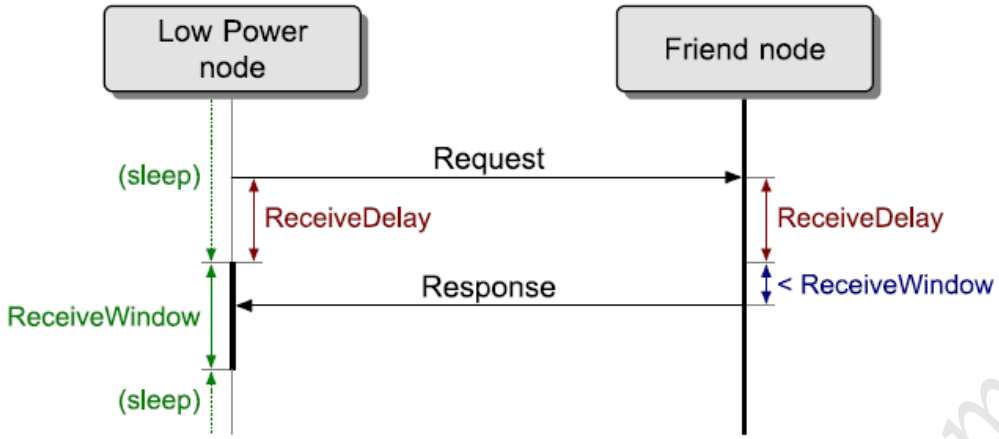


Figure 3.17: Friendship timing parameters

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。

5.1.3 “友谊”建立

蓝牙 mesh 网络中“友谊”的建立，需要经过以下步骤。

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。
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。
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。
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。
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。

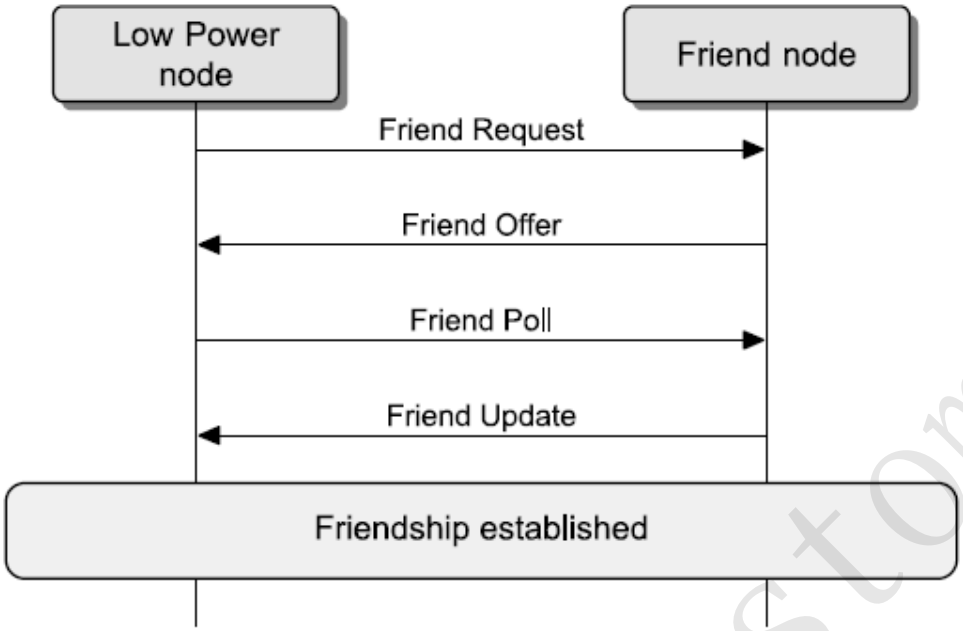
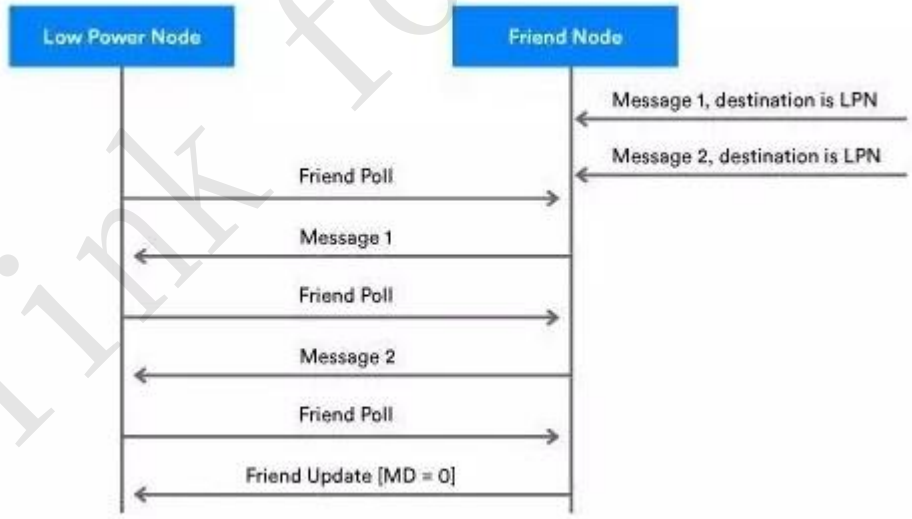


Figure 3.19: Establishment of a friendship

5.1.4 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。

5.1.5 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。

5.1.6 “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。

5.2 LPN 使用演示

5.2.1 硬件准备

两个 8269 dongle，一个烧录 8269_mesh.bin(默认支持 friend)，另一个烧录 8269_mesh_lpn.bin。

5.2.2 测试步骤

(1) 用 APP 或者 SIG_MESH_TOOL 工具对 mesh friend 节点进行配置。

(2) 用 APP 或者 SIG_MESH_TOOL 工具对 mesh LPN 节点进行配置。LPN 节点在未配置的时候，目前暂时是一直发送 adv，不进入睡眠，等待进行 provision。

(3) 当 provision 完成后，默认也没有进入睡眠，等待别的 friend 节点发送包含 iv update 信息的 Secure Network beacon 包，因为要确认 iv index 是和当前 mesh network 有相同的 iv index 以及有其他节点存在，此时 LPN 才能主动发起 friend request。此时可连接广播包已经停止发送。

(4) 当收到包含 iv update 信息的 Secure Network beacon 包后，LPN 主动发出 friend request，然后按 flow 建立友谊（friendship），如果建立成功，则红灯灭，并进入 suspend 模式。

(5) 进入 suspend mode 后，目前为了测试方便，暂时没有主动按周期发送 poll 命令，而是按下按键 SW1 (GPIO_PD2) 发送 poll，当处于 suspend 模式的时候，每按一次会发送一次 POLL。

(6) 发送 POLL 后，如果 friend 节点有 LPN 的消息，则会获取到该消息。

至此，整个 friend ship 建立完成。

(7) 另外，当 LPN 处于 suspend 模式的时候，按下按键 SW2 (GPIO_PC5)唤醒 LPN，并用主安全资料加密执行 test_cmd_wakeup_lpn()函数发送 ALL ON/OFF 命令。

5.3 app.c 文件介绍

5.3.1 广播包以及广播响应包的定制

参考《8269 MESH 工程介绍》。

5.3.2 fifo 部分配置

参考《8269 MESH 工程介绍》。

5.3.3 int app_event_handler (u32 h, u8 *p, int n)

参考《8269 MESH 工程介绍》。

5.3.4 main_loop ()

参考《8269 MESH 工程介绍》。

5.3.5 user_init()

参考《8269 MESH 工程介绍》。

5.3.6 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。

5.3.7 void 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。

6. SWITCH 工程介绍

6.1 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。

6.2 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。

6.3 app.c 文件介绍

6.3.1 广播包以及广播响应包的定制

参考《8269 MESH 工程介绍》。

6.3.2 fifo 部分配置

参考《8269 MESH 工程介绍》。

6.3.3 int app_event_handler (u32 h, u8 *p, int n)

参考《8269 MESH 工程介绍》。

6.3.4 main_loop ()

参考《8269 MESH 工程介绍》。

6.3.5 user_init()

参考《8269 MESH 工程介绍》。

6.3.6 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。

6.3.7 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。

6.3.8 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。

6.3.9 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 6.7 节中的休眠部分的处理流程图。

6.3.10 kb_scan_key (int numlock_status, int read_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。

6.4 switch 部分的配置

6.4.1 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。

6.4.2 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

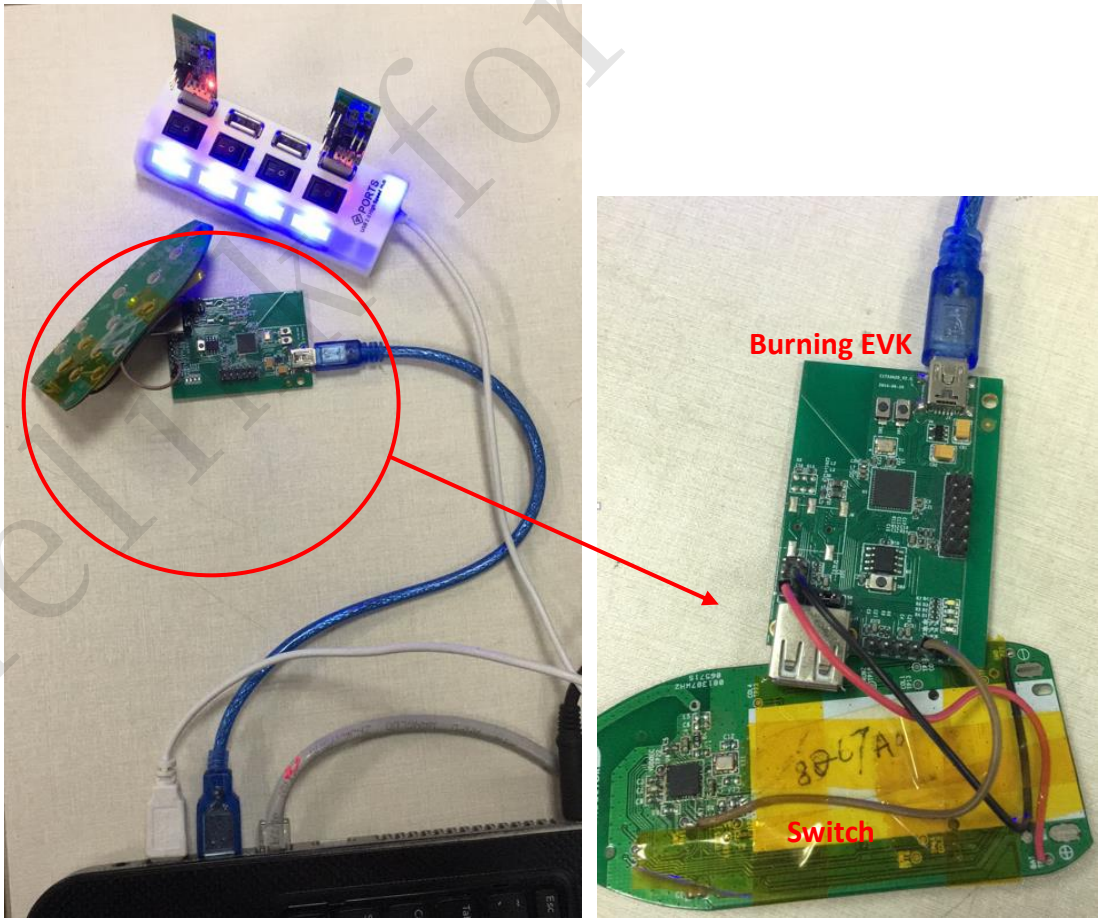
6.4.3 switch 开关灯

根据不同的键值发送不同的命令以及做出相应的处理。

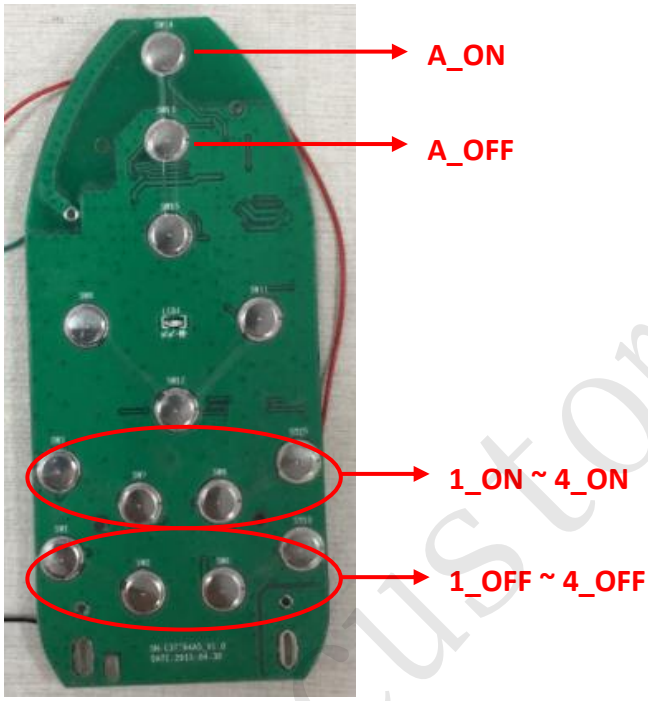
参考具体的按键处理程序 mesh_proc_keyboard（）。switch 在加入网络前，无法控制网络中灯的开关。可以通过同时按键 RC_KEY_A_ON 和 RC_KEY_1_OFF 超过 2s，之后触发配对模式，通过 provisioner 将 switch 加入网络，之后就可以通过 RC_KEY_A_ON 和 RC_KEY_A_OFF 来全开/全关网络中的灯。详见 6.5 中 switch 部分的操作。

6.5 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：



switch 按键的示意图为：

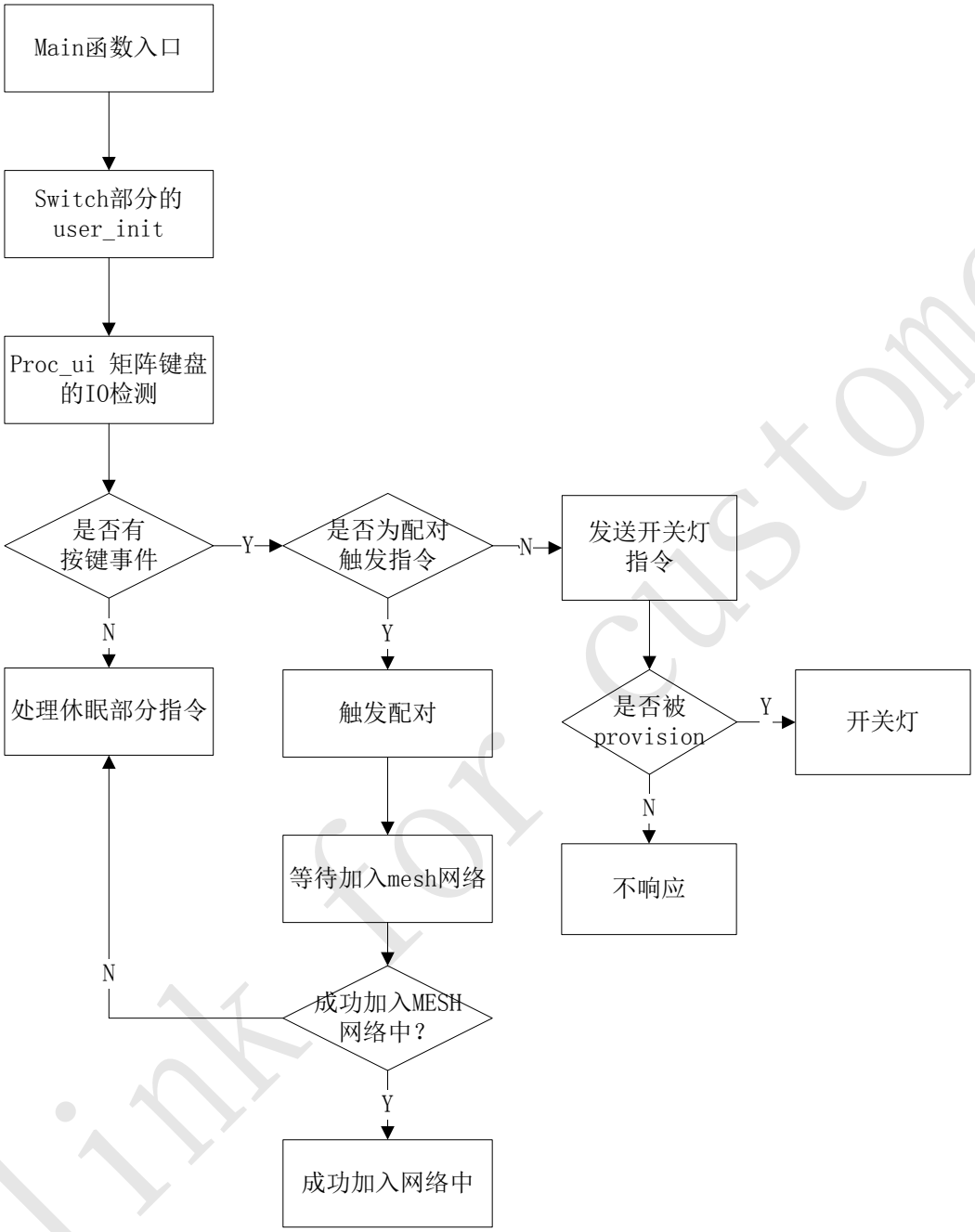


将 switch 节点的设备上电，上电之后由于 switch 属于低功耗节点，所以必须先触发配对模式，使得 switch 通过 provisioner 加入到 mesh 网络中。触发配对模式的方式为同时按压 RC_KEY_A_ON 和 RC_KEY_1_OFF 超过 2s。

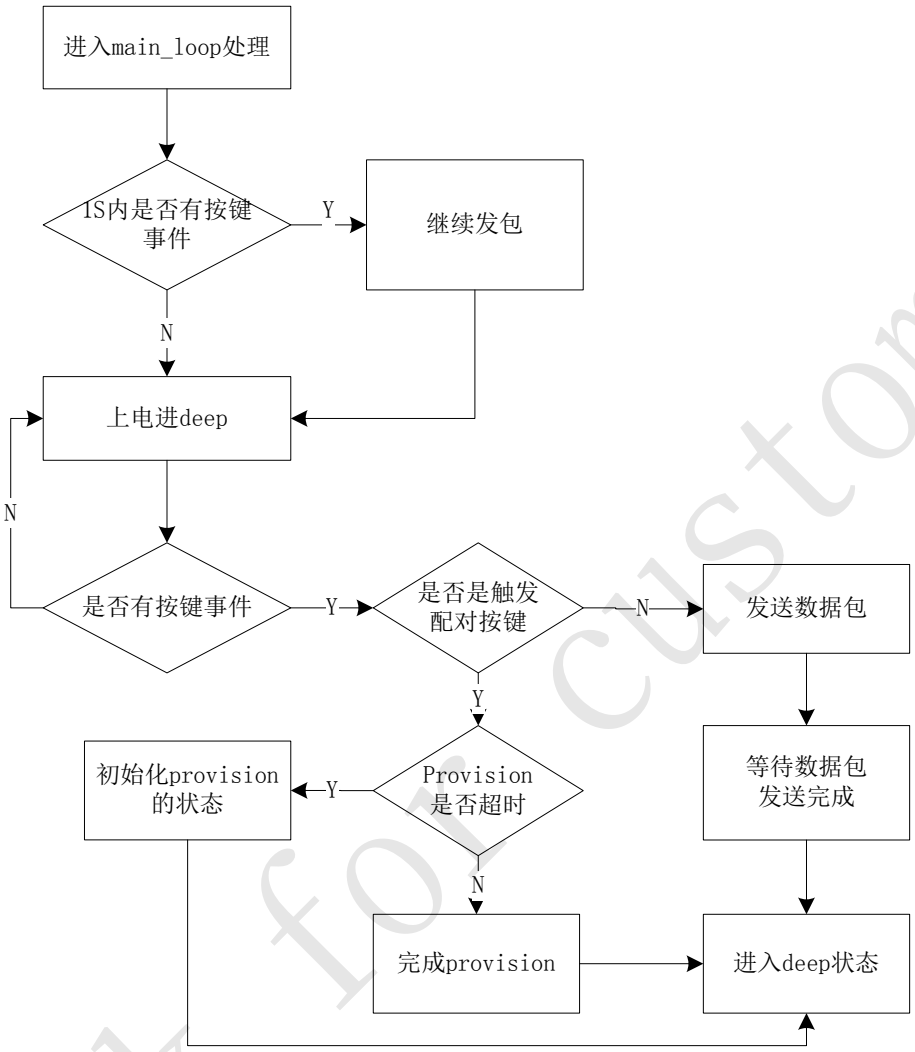
switch 上的 led 灯连续闪烁 4 次，表示进入配对模式，provisioner 节点上电，等待 15s 左右，switch 上的 led 连续闪烁 4 次，表示 switch 已经加入到网络。

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。

6.6 遥控器的运行流程图



6.7 休眠处理的流程图



7. 工具操作说明

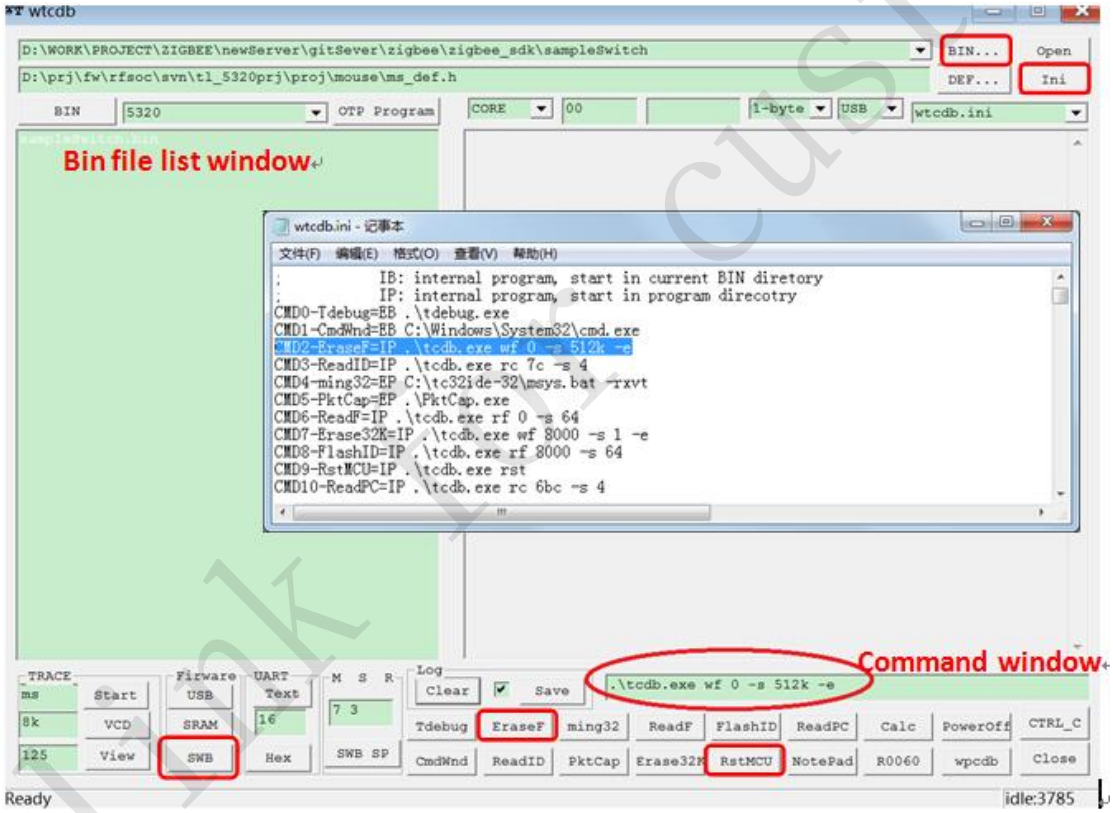
7.1 下载固件

在操作之前，需要先下载“8269_mesh.bin”到每一个节点中 (8269 Dongle)，然后把“8267_kma_dongle.bin”下载到 8267 的 Master Dongle (8267 Dongle)。

例如，用户可以按照以下步骤下载灯的固件。

(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。8269 Dongle 通过 USB 接口来连接 EVK 板的 USB 接口。

(2) 通过 Telink Wtcdb 工具将“8269_mesh.bin”下载到 8269 Dongle 的 flash。



Wtcdb 界面

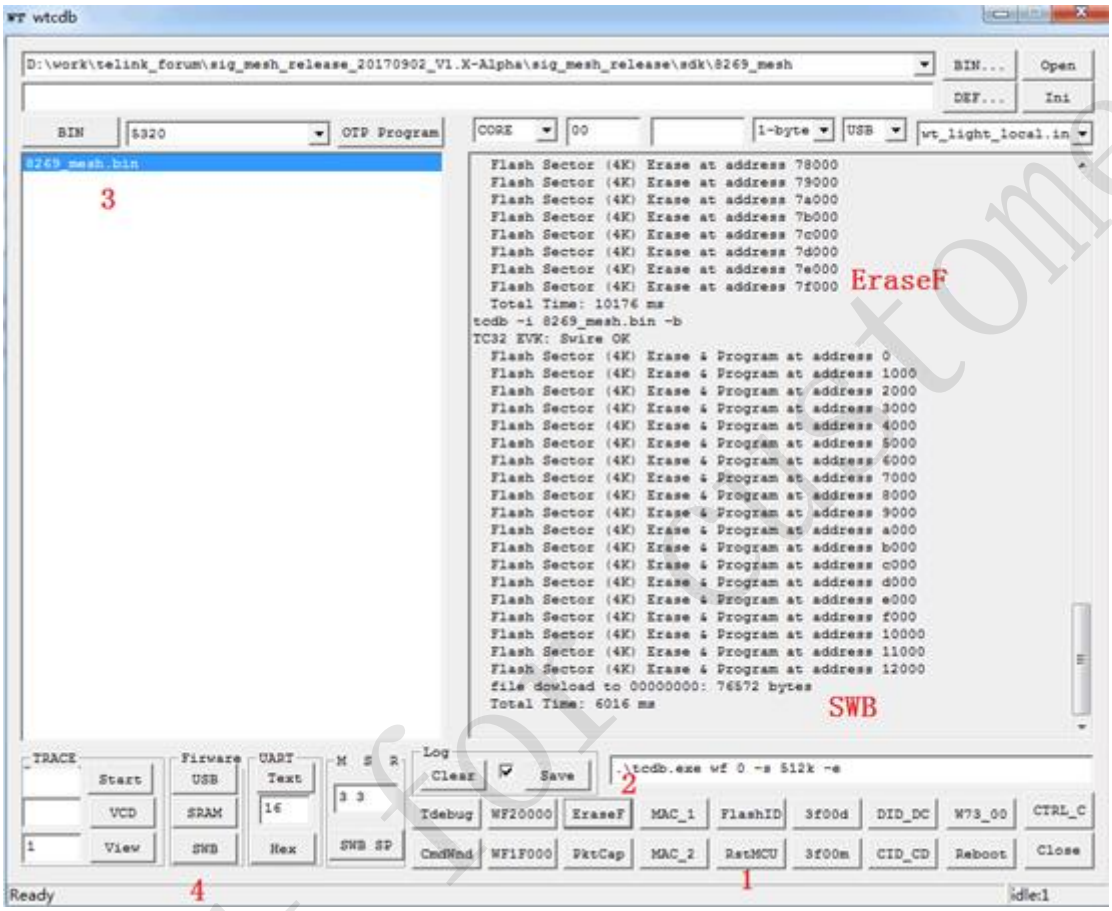
Step 1: 打开 Wtcdb，点击“**RstMCU**”按钮复位 MCU 并且检查硬件连接部分。

Step 2: 通过以下两种方法擦除 8269 Dongle 板的整个 flash 部分。

- ✧ 在 command window 输入指令 “ .\tcdb.exe wf 0 -s 512k -e”并按下回车键来擦除整个 flash 部分的内容。
- ✧ 用户通过点击“Ini”按钮来打开“wtcdb.ini”文件，将配置命令“CMD2-EraseF”改成“.\tcdb.exe wf 0 -s 512k -e”，保存文件，然后用户可以通过“EraseF”按钮来实现整个 flash 部分的擦除。

Step 3：点击“**BIN**”按钮来选择对应的“8269_mesh.bin”文件来下载，然后在左边框中选中对应的文件。

Step 4：点击“**SWB**”按钮，将选中的“8269_mesh.bin”烧写到 flash 从 0 开始的地址。



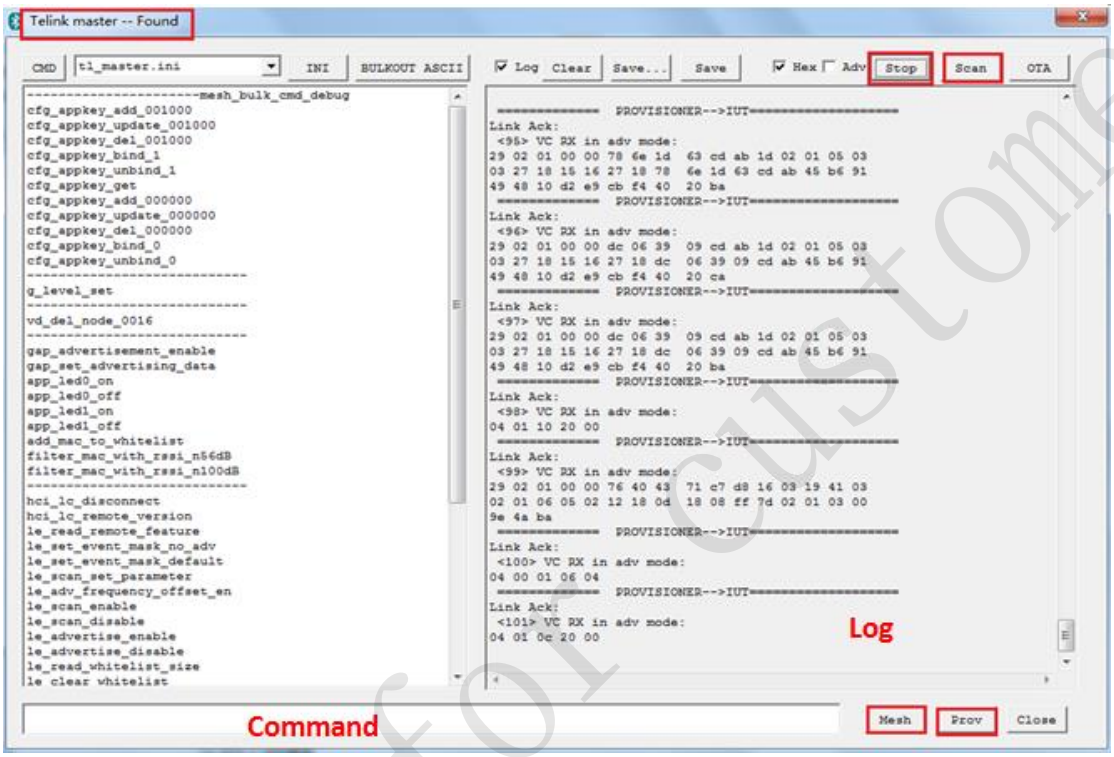
固件烧写步骤

Step 5: 重新上电后，8269 Dongle 板能够当成一个普通的灯的节点。用户也可以参考泰凌微电子的文档--“***AN_FBD-EVK-UG_Firmware burning and debugging User Guide***”来获取更多的细节。

7.2 BLE 连接和加灯

(1) 打开“SIG_MESH_TOOL”工具，将 8267 Dongle 插到 PC 的 USB 口中。

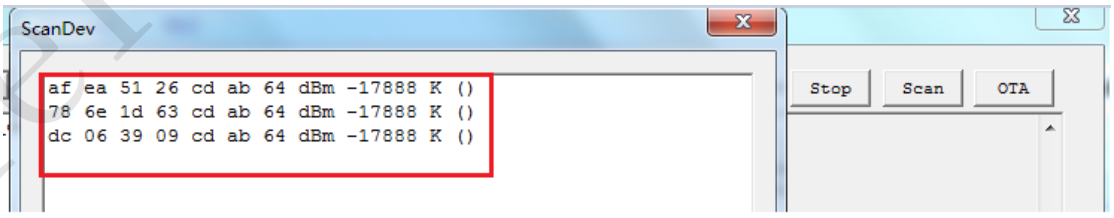
(2) 如下图所示，工具左上角的“Found”表示 8267 Dongle 和 PC 工具正常连接，并且可以正常通信。



“SIG_MESH_TOOL”工具界面

(3) 将 8269 mesh 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 地址和频率偏差列表。



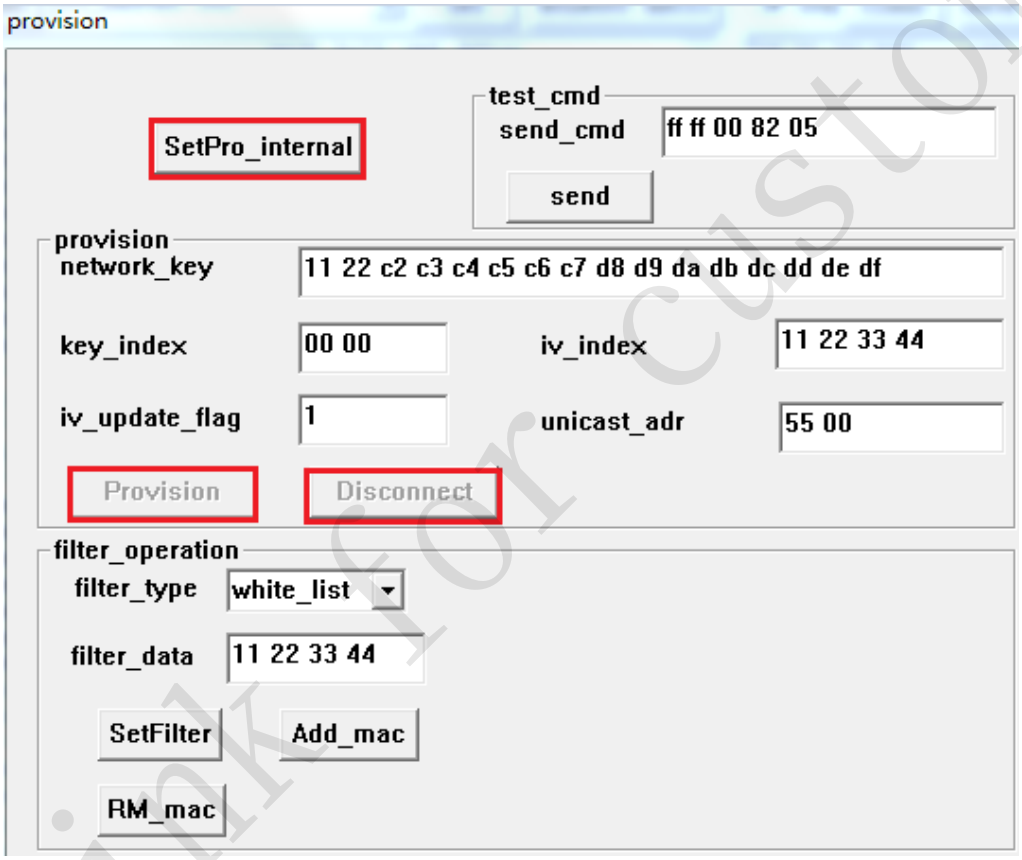
“ScanDev”窗口

(5) 8267 Master dongle 暂时只支持单个节点的 BLE 连接。双击“ScanDev”窗口中对应的条目来建立 BLE 连接，如果 8267 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，8267 Dongle 上的白灯亮起，表示 BLE 连接结束。

(6) 点击右下角“Prov”按钮来打开“provision”窗口。

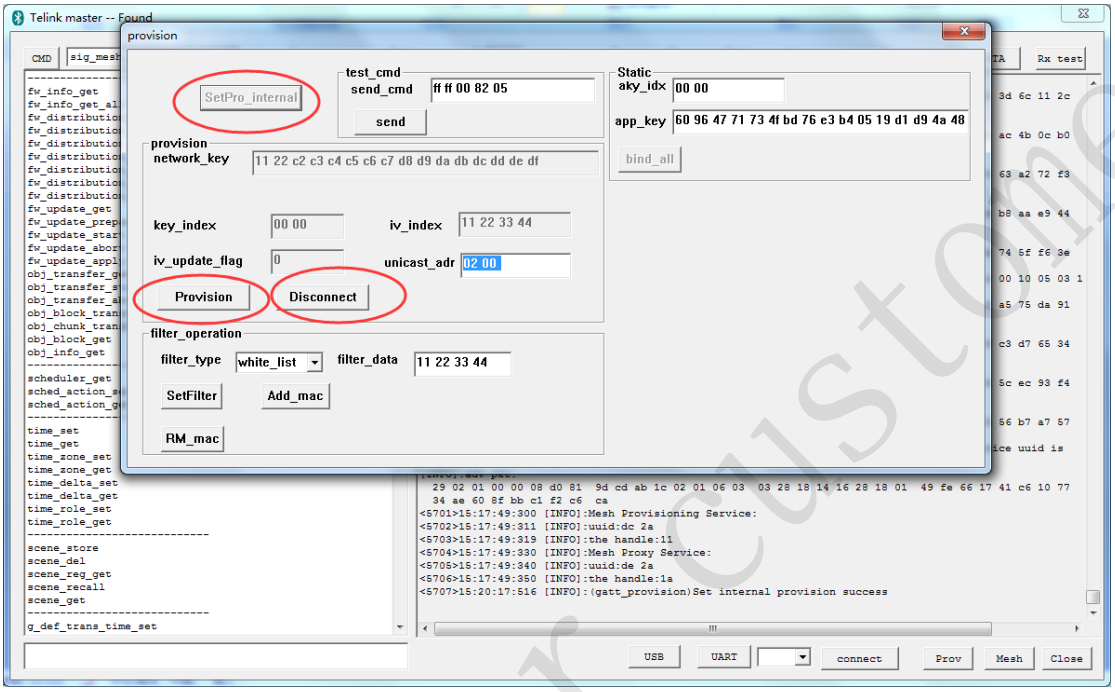
注意：在初始状态下，“Provision”按钮和“Disconnect”按钮是被禁止的，用户不可同时操作这两个按钮。



Provision 窗口

(7) 第一次点击“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。

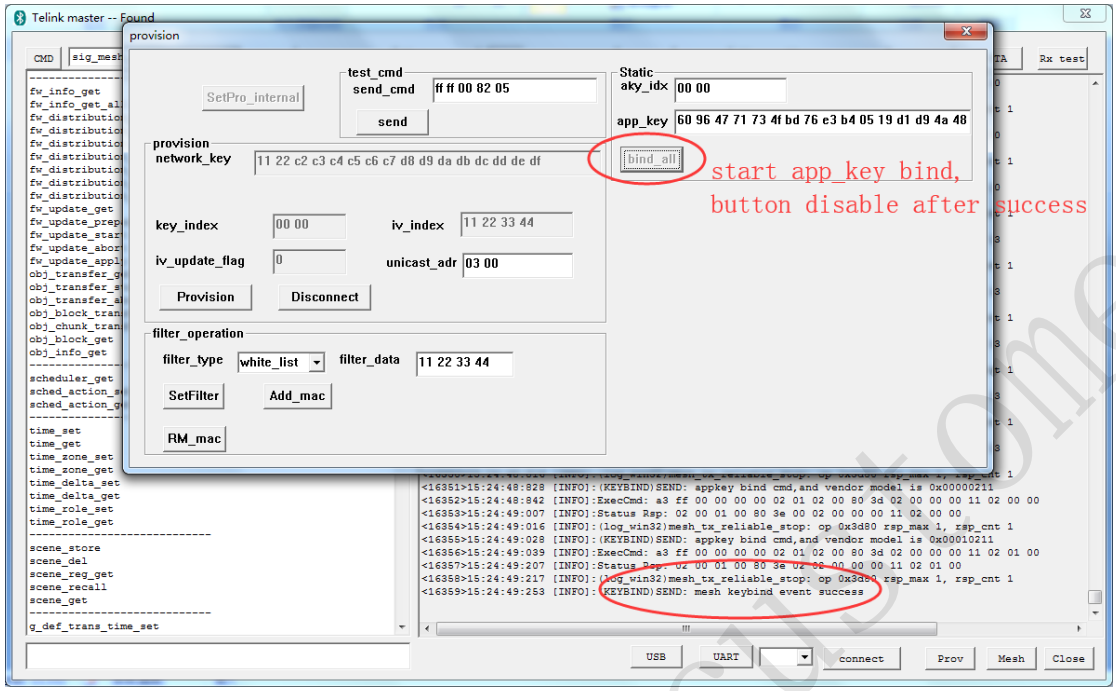
一旦点击“SetPro_internal”，在工具内部会以文件的方式存储信息，下次重新开启工具，会自动读取之前设置的参数。“SetPro_internal”禁止，“Provision”和“Disconnect”两个按钮使能。若需要修改 network_key，需删除工具目录下的 test.bin 文件，然后重启工具重复（6）步骤。



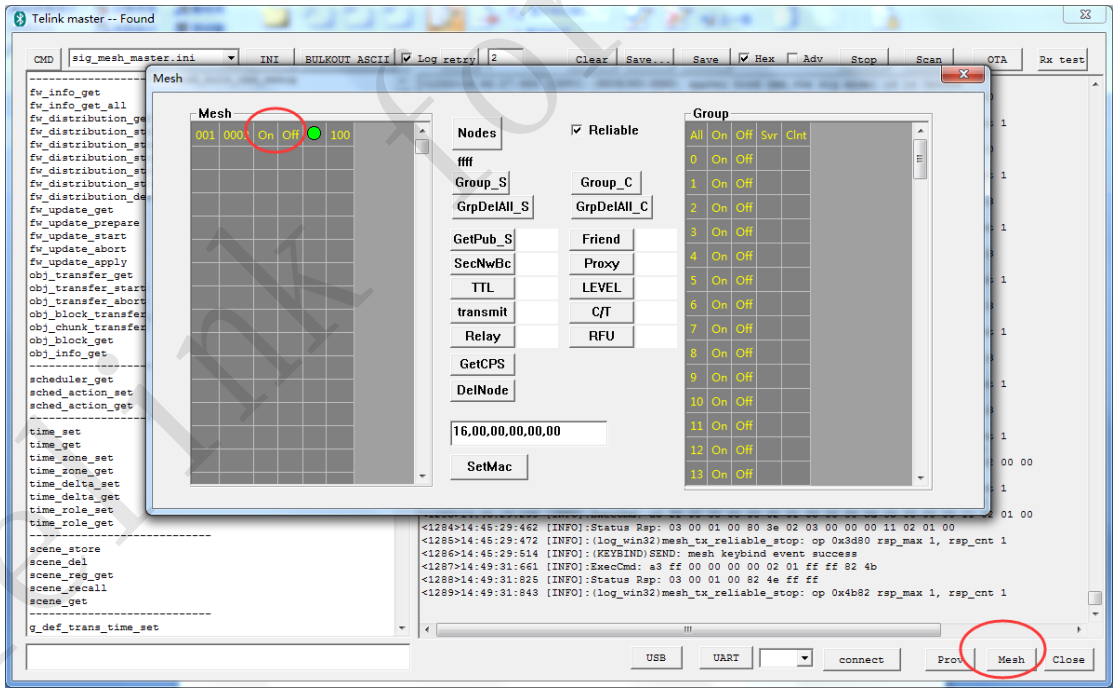
(8) 点击“Provision”按钮来设置对应的参数，即能把对应的节点加入到网络中，连接的节点会闪烁 4 下表示成功。log 窗口会打印 “provision success”信息。

```
provision success
```

(9) 设置好 app_key 后点击 bind_all 为 model 绑定 app_key。



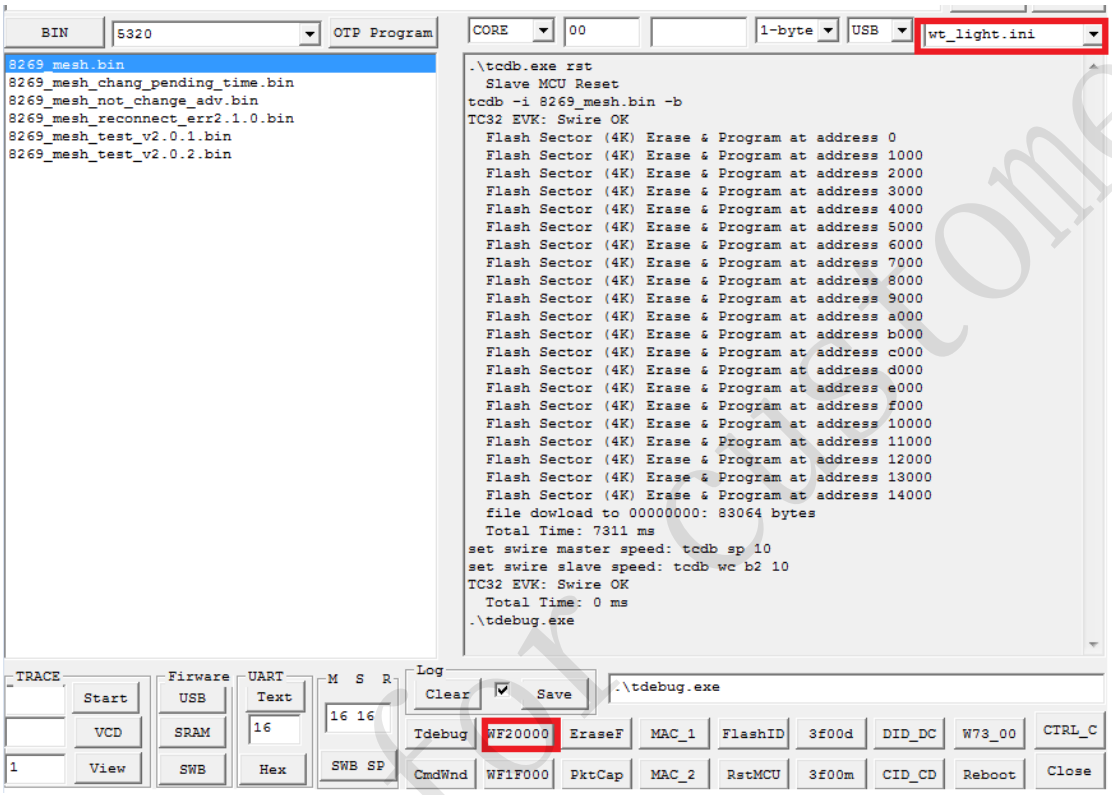
(10) 绑定 App_key 后点击主界面 Mesh 按钮进去 mesh 界面可进行开关灯等操作。



7.3 OTA 部分的说明

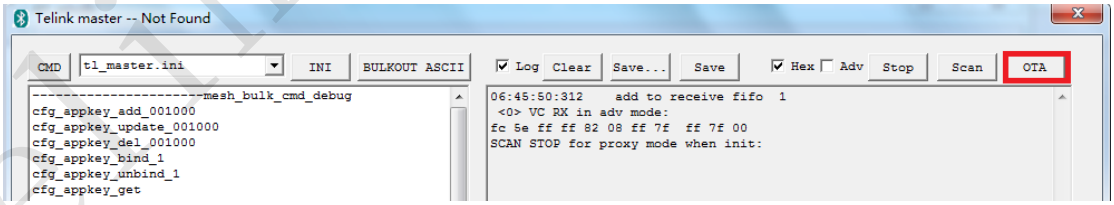
(1) 将需要 OTA 的 BIN 文件（New FW）按照 7.1 的说明下载到 8267 dongle 的 flash 地址 0x20000 开始的位置，0x0000 的位置烧录 kma_dongle.bin。

注意：烧录 New FW 和 kma_dongle.bin 时，应分别点击“**WF20000**”/“**SWB**”按钮开始烧录（对应 7.1 中的 step 4）。



(2) 执行完 7.2 中的(1)~(5)的步骤，建立起目标节点和工具之间的 BLE 连接。

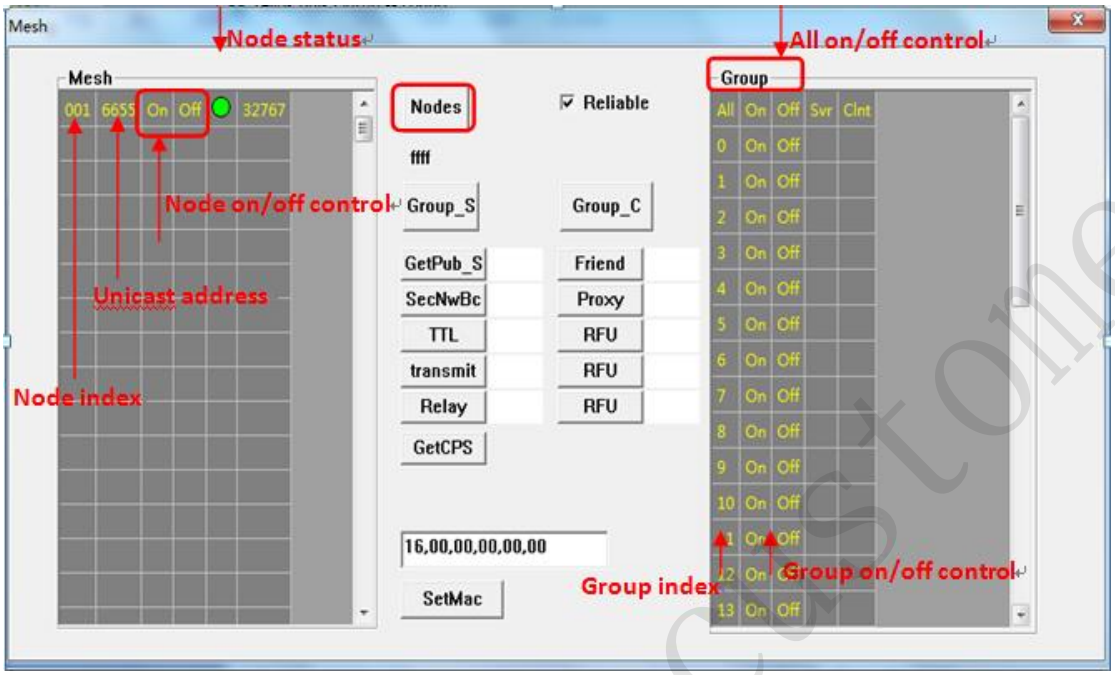
(3) 点击 OTA 按钮，启动 OTA 过程。如果正常完成 OTA，该节点会连续闪烁 8 下。



(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。

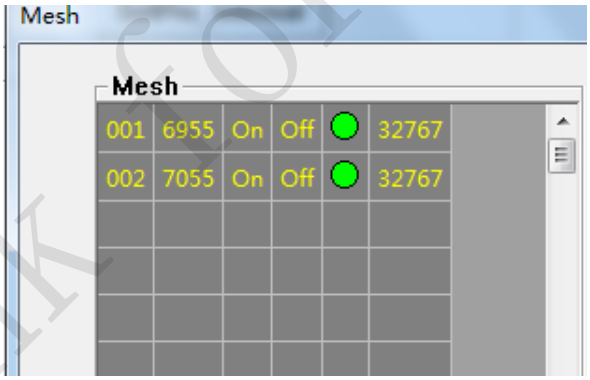
7.4 控制对应的节点

(1) 点击主窗口右下角的“Mesh”按钮。会弹出一个“Mesh”窗口。



Mesh 窗口

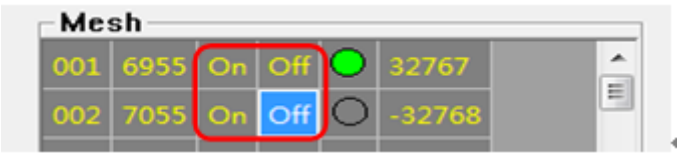
(2) 用户能够点击“Mesh”窗口中的“Nodes”来刷新所有的灯的状态。



节点状态

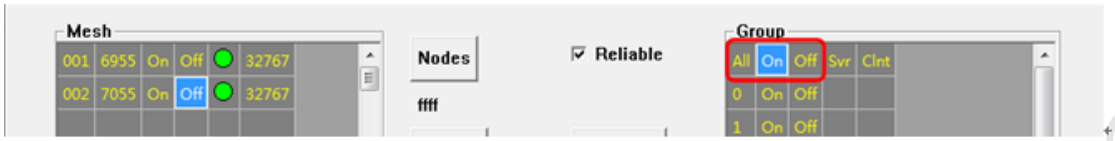
(3) 单个节点操作：点击“On”/“Off” 按钮 (如图红框所示)，对应的灯会控制开关的状态。对应的节点的状态会上报给工具来刷新对应的状态。

表示状态“on”， 表示状态“off”。



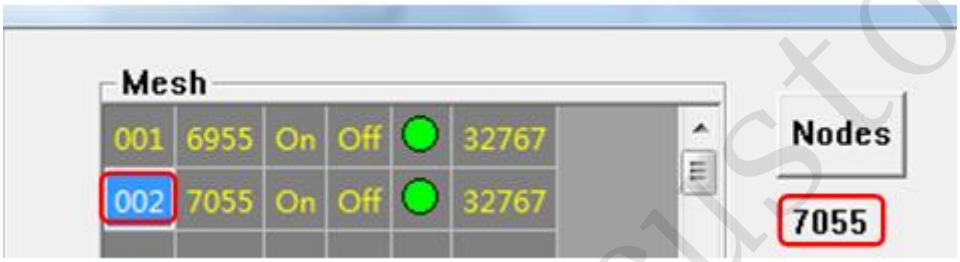
单个节点的控制

(4) 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。



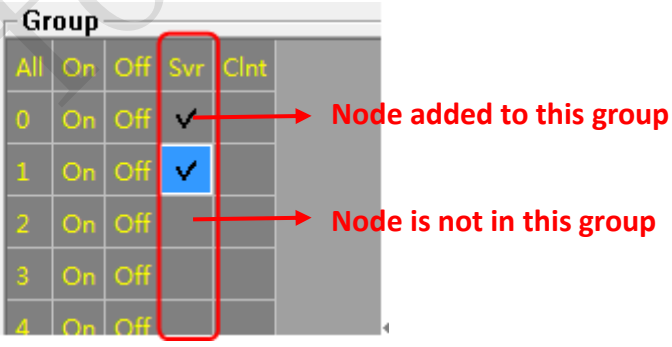
所有的节点的控制

(5) 分组控制：点击对应灯节点的索引，如 002，获取关联灯的地址。



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。



分配一个灯到不同的组

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。

组的控制

8. 增加 model 举例说明

暂不支持增加 model，用户可以直接使用已经添加的 vendor model：VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S。

9. 增加 vendor 命令的举例说明

9.1 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 命令去掉 vendor id 部分。
- ✧ 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。
- ✧ model_id_tx：发送该命令对应的 model ID。
- ✧ model_id_rx：接收该命令对应的 model ID，如果该 node 没有对应的 model id，则不处理这个 opcode。
- ✧ cb：收到该命令时，调用的回调处理函数。
- ✧ op_rsp：如果该 opcode 是 acknowledge request command，则该处写对应的 ack command，否则写 STATUS_NONE。

9.2 增加 acknowledge command

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。

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。

(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。

(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。

9.3 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。

10. 恢复出厂配置

10.1 默认触发动作

SIG_mesh 模组(非低功耗节点)可以用下面的流程来恢复出厂配置：

1) 上电 SIG_mesh 模组，等待 30s 以上。

   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)。

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)。

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)。

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,};
```

10.2 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：

(1) 左边的值比右边的小。

(2) 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,};
```