Telink SIG Mesh

SDK Developer Handbook

AN-17120400-E7

Ver1.6.0

2024.04.23

Keyword

SIG Mesh

Brief

This document is Telink SIG Mesh SDK Developer Handbook.

Published by
Telink Semiconductor

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

Legal Disclaimer

This document is provided as-is. Telink Semiconductor reserves the right to make improvements without further notice to this document or any products herein. This document may contain technical inaccuracies or typographical errors. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein.

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:

telinksales@telink-semi.com

telinksupport@telink-semi.com

Revision History

Version	Changes
V1.0.0	Initial release.
V1.1.0	This is the second release, compare with last version, the following parts have been updated：1. SDK Overview; 2. Global Configuration Introduction; 3. 8268 Mesh Project Introduction; 4. Provisioner(Gateway) Project Introduction; 5. SWITCH Project Introduction
V1.2.0	This is the third release, compare with last version, the following parts have be updated：1 SDK Overview; 4 Debugging Tool Instruction; 11 Mesh LPN Project Introduction; 8 Global Configuration File Introduction. The following parts are added：2 MCU Basic Modules; 5 Factory Test Mode; 6 Important SDK Modules; 7 Vendor Model Introduction; 9 8258 MESH Project Introduction; 13 Connect with a Platform; 14 Factory Reset; 15 SIG Remote provision Demo; 16 Fast bind Mode(PROVISION_FLOW_SIMPLE_EN Mode); 17 Private Fast provision Function Demo; 18 Private online status Function Demo; 19 OTA Test Brief; 20 Network Sharing; 21 Control Nodes via INI Demo
V1.3.0	This is the fourth release, compare with last version, the following parts have be updated：Delete draft feature
V1.4.0	This is the fifth release, compare with last version, corrected some terminology.
V1.5.0	This is the sixth release, compare with last version, new chapters 21~34 chapters have been added, and the following sections have been updated：2. MCU basic modules; 3. commonly used modules in SDK; 7. use of Vendor model; 10. Gateway; 11. LPN; 12. Switch.
V1.6.0	This is the seventh release, compare with last version, new chapter Android & iOS APP User Guide is added.

Contents

List of Figures

List of Tables

1 SDK Overview

Telink SIG Mesh SDK supplies demonstration code for SIG_mesh protocol application development, based on which users can develop their own application program.

SDK download method:

Visit Telink Wiki to download the latest Bluetooth Mesh SDK, or click this link to download directly: sig_mesh_sdk.zip.

Current projects of the SDK apply to Telink IC 825x/ 8278/ b91m/ 8269, Telink provides PC tool (sig_mesh_tool.exe) connecting master dongle through USB to realize provisioner function, please note, master dongle supports only 8269 SoC series.

For a quick overview and demonstration of basic functionality, please refer to:

BLE SIG Mesh Quick Start

1.1 SDK File Architecture

The file architecture for Telink SIG Mesh SDK includes APP (application) layer and BLE&SIG_mesh protocol layer. After the SDK project is imported in Telink IDE (please refer to AN_IDEUG-E1_Telink IDE User Guide.pdf for project importing, AN_16063000-E1_Guide for Adding New Project on Existing SDK.pdf for adding new project), the file structure is shown in figure below, containing the following top-layer folders. There are several main top-level folders: boot, common, drivers, homekit_src, proj_lib, stack, and vendor.

Note:

> In the B91m project, enable this macro __TLSR_RISCV_EN__ in the compiler's pre-compile macro.

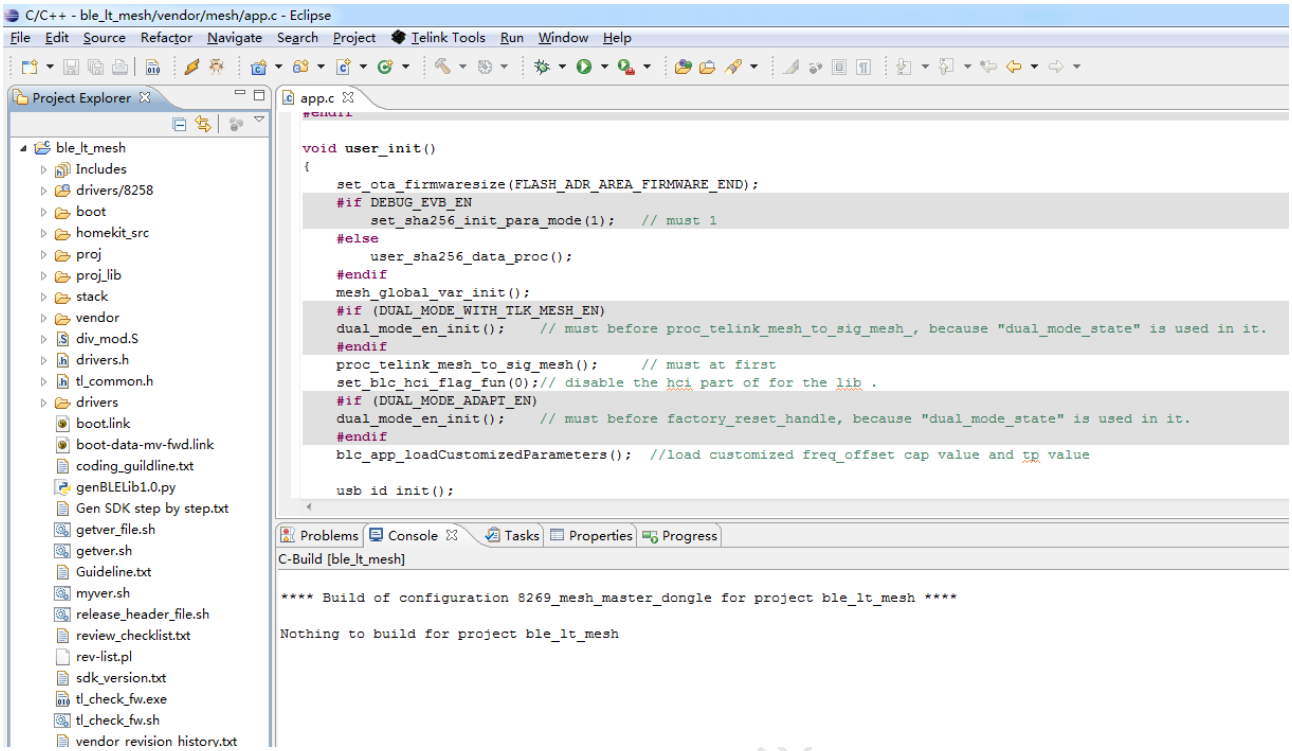


Figure 1.1: File Architecture

- boot: This folder contains bootloader of the SoC, i.e, the compiling process after MCU boot or awake from deep sleep, this is the environment base for later C programs.
- drivers: This folder contains configuration files and driver programs for hardware peripherals related with MCU, like clock, flash, i2c, usb, gpio, uart and etc.
- proj: This folder contains MCU related peripheral driver, such as flash, I2C, USB, GPIO, UART driver, and etc.
- proj_lib: This folder contains library files necessary for MCU running, for example, BLE stack, RF driver, PM driver. Since this folder is supplied in the form of library files, the source files are not open to users, for example, BT stack library file "liblt_8269_mesh.a", library file for SIG_mesh common node "libsig_mesh.a", library file for SIG_mesh low power node "libsig_mesh_LPN.a", library file for SIG_mesh provision node "libsig_mesh_prov.a".
- stack: This folder contains BLE protocol related header files. The sorce files are compiled into library files, and are not open to users.
- vendor: This folder contains user APP-layer code, including:
  - 8267_master_kma_dongle:Firmware used for host test. In combination with host tool(sig_mesh_tool.exe) of GATT mode, it can act as a provisioner and it's used for demonstration and debugging.
  - common:It mainly contains common modules in mesh/ mesh_lpn/ mesh_provision/mesh_switch, for example, SIG mesh model processing, LED module, factory initialization module, test command module.
  - mesh/ mesh_gw_node_homekit/ mesh_lpn/ mesh_provision/ mesh_switch/ spirit_lpn have the same structure, which all include "app.c", "app.h", "app_att.c", "app_config.h" and "main.c".

"app.c/app.h" contains initialization and bottom-layer callback function; "app_att.c" is description of BT ATT table and interface functions; "app_config.h" defines corresponding macros and declarations in projects; main.c is main function and interrupt function entry.

1.1.1 main.c

This includes the main function entry, system initialization related functions, and the infinite loop while(1), it is not recommended to verify this file, please follow the fixed codes.

```
int main (void) {
    FLASH_ADDRESS_CONFIG;
#if PINGPONG_OTA_DISABLE
    ota_fw_check_over_write();  // Copying firmware for non-Pinkpong OTAs
#endif
    blc_pm_select_internal_32k_crystal(); //Select internal 32k rc as 32k counter clock source
    cpu_wakeup_init();//The most basic hardware initialization of the MCU
    int deepRetWakeUp = pm_is_MCU_deepRetentionWakeup();  //Determine whether to wake up from
     ↪ deep retention.
    rf_drv_init(RF_MODE_BLE_1M); //RF Initialization
    gpio_init(!deepRetWakeUp);   //gpio initialization, user configure relevant parameters in
 ↪ app_config.h
    clock_init(SYS_CLK_16M_Crystal);
    if( deepRetWakeUp ){
        user_init_deepRetn ();//Quick initialization of deep retention waking up
    }else{
        user_init_normal ();//ble initialization, whole system initialization, user to set up
    }
    irq_enable();        //globalize interruptions
    while (1) {
#if (MODULE_WATCHDOG_ENABLE)
        wd_clear(); //clear watch dog
#endif
        main_loop ();  //Includes tasks for ble transceiver processing, low power management,
 ↪ mesh and user
    }
}
```

1.1.2 app_config.h

Users configure file to configure all system related parameters, including BLE parameters, GPIO parameters, PM low power management configure parameters and etc.

The definition of each parameter in app_config.h in later parts of this document when each module is introduced.

1.1.3 BLE stack entry

There are 2 entry functions of BLE stack code in Telink BLE SDK.

a) BLE related interrupt handler entry of irq_handler in main.c file irq_blt_sdk_handler.

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

b) BLE logic and data processing function entry in application file main_loop blt_sdk_main_loop.

```
void main_loop (void)
{
    mesh_loop_proc_prior();//process with high priority, leap over the 10ms interval mailoop
    8269
////////////////////// BLE entry /////////////////////////////
    blt_sdk_main_loop();

////////////////////// UI entry /////////////////////////////
    factory_reset_cnt_check();//5 times of boot factory reset
    mesh_loop_process();//mesh related loop function
        ......

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

1.2 Demo Project

Telink SIG Mesh SDK provides multiple BLE demonstrations.

Users can observe the intuitive effect by running the hardware and software demos. Users can also modify the demo code to complete their own application development.

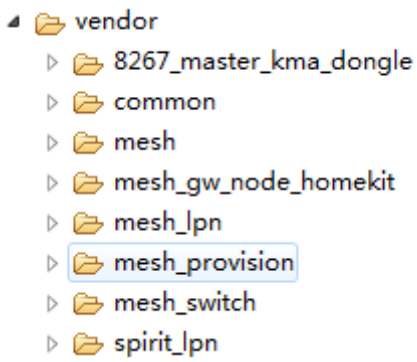


Figure 1.2: Mesh SDK demo code

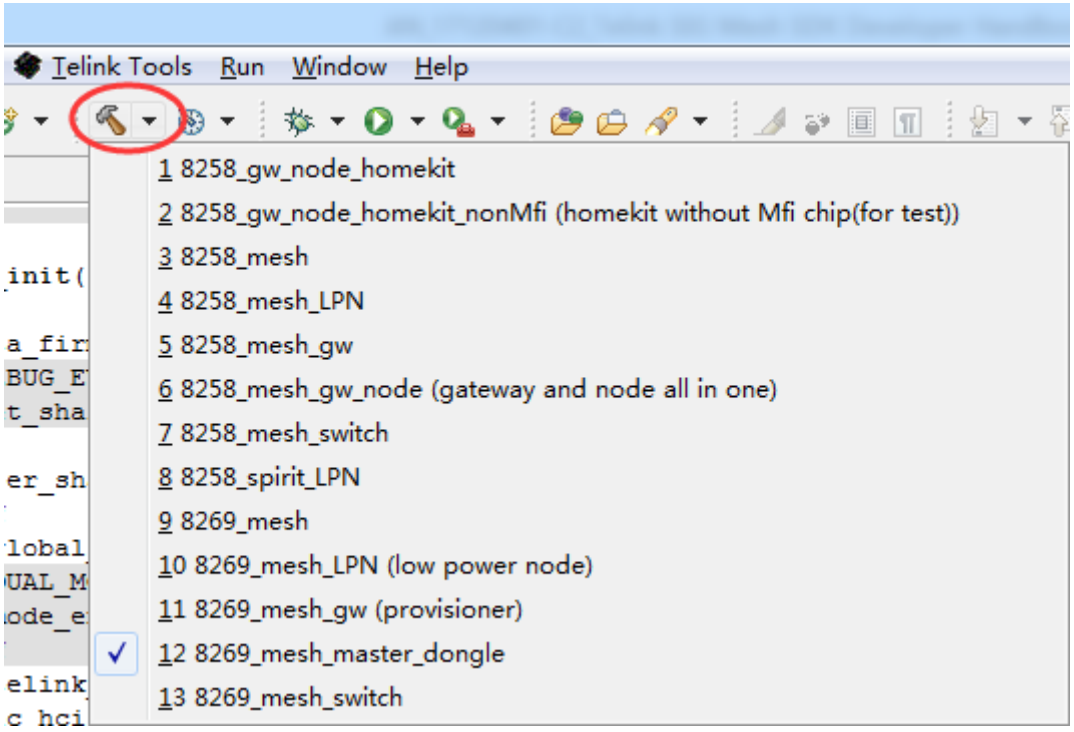


Figure 1.3: Mesh SDK compiling options

The differences of each mesh demo are listed in the following table.

Table 1.1: Mesh SDK Project Example

Demo	Vendor folder	Application	Mesh Feature
8258_mesh/ 8269_mesh	.\mesh	CT/HSL light and etc	Relay, friend, proxy
8258_mesh_LPN\ 8269_mesh_LPN	.\mesh_lpn	LPN	LPN, proxy

Demo	Vendor folder	Application	Mesh Feature
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	Remote control applications	proxy
8258_spirit_LPN	.\spirit_lpn	Tmall Genie Customized LPN Mode	proxy
8269_mesh_ master_dongle	.\8267_master _kma_dongle	Tool	GATT provisioner

- 8269_mesh_master_dongle compiling option: supports GATT provisioner function, no rely nor friend function. Firmware uses this for host test, together with GATT mode, the firmware can act as a provisioner to demonstrate and debug.
- 8258_mesh, 8269_mesh compiling option: compiling project of normal SIG Mesh nodes, can be configured by provisioner, supports relay, friend, proxy function, no provision function.
- 8258_mesh_LPN, 8269_mesh_LPN compiling option: compiling project of LPN MESH nodes, receive message via friendship, does not support relay，friend，provision functions. It supports proxy function, but only communicates between the GATT master and LPN, does not forward commands sent by the app to other nodes.
- 8258_mesh_gw, 8269_mesh_gw compiling option: compiling project of gateway provisioner nodes, supports adv provisioner, can configure other nodes, supports relay，friend functions.
- 8258_mesh_switch, 8269_mesh_switch compiling option: compiling project of (switch) MESH nodes. To lower power consumption, after the switch is provisioned, it send message with other receiving. Does not support relay，friend function.
- 8258_gw_node compiling option: supports functions of both gateway adv provisioner and mesh node. Like a gateway, it can establish its own network and add other unprovision nodes. It can also be provisioned by other provisioners.
- 8258_spirit_LPN compiling option: self-defined LPN mode of TMALL Genies.

1.3 LIGHT_TYPE_SEL Introduction

This macro is used to choose the pre-configured light types.

```
#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 ON/OFF  model
#define LIGHT_TYPE_LPN_ON_OFF_LEVEL   8   // only ON/OFF, LEVEL model
#define TYPE_TOOTH_BRUSH              9
#define LIGHT_TYPE_NLC_CTRL_CLIENT    10
//#define LIGHT_TYPE_NLC_BLC        // set NLCP_BLC_EN to 1 to enable Basic Lightness Controller
 ↪  Mesh Profile.
#define LIGHT_TYPE_NLC_SENSOR         11
```

LIGHT_TYPE_CT:

CT is the abbreviation of color temperature light, and the corresponding product is color temperature light, contains color temperature related model, for example, Light CTL Server，Light CTL Setup Server，Light CTL Temperature Server, and corresponding extend model, such as Generic On/off Server，Generic Level Server，Light Lightness Server and etc.

LIGHT_TYPE_HSL:

The corresponding product is RGB light, contains Light HSL Server，Light HSL Hue Server，Light HSL Saturation Server，Light HSL Setup Server and corresponding extend model, for example, Generic On/off Server，Generic Level Server，Light Lightness Server and etc.

LIGHT_TYPE_XYL:

The corresponding product is XYL light, contains Light xyL Server，Light xyL Setup Server and corresponding extend model, for example, Generic On/off Server，Generic Level Server，Light Lightness Server and etc.

LIGHT_TYPE_POWER:

The corresponding product is power adapter, contains generic Power Level Server，Generic Power Level Setup Server and corresponding extend model, for example, Generic On/off Server，Generic Level Server and etc.

LIGHT_TYPE_CT_HSL:

The corresponding is CT light + HSL Light, contains CT and HSL related model, and Generic On/off Server，Generic Level Server，Light Lightness Server. CT light use the same lightness and on/off parameter with HSL light. Only one light is lighted at one time.

LIGHT_TYPE_DIM:

The corresponding light is dimming light, contains Light Lightness Server，Light Lightness Setup Server and the corresponding extend model, for example, Generic On/off Server，Generic Level Server.

LIGHT_TYPE_PANEL:

The corresponding product is switch panel, which is the server, controlled by instruments like app and executing on/off switch. The default switch number is 3(defined by LIGHT_CNT).

LIGHT_TYPE_LPN_ONOFF_LEVEL:

The corresponding product is LPN equipment, contains Generic On/off Server model by default, and mesh OTA model is disabled. This is mainly for demo LPN function.

Note:

- When use LPN equipment, 825x retention RAM size should not exceed 32K.
- All models in node will appear in composition data (global variable model_sig_cfg_s_cps).

TYPE_TOOTH_BRUSH:

The corresponding product type is a customized product.

LIGHT_TYPE_NLC_CTRL_CLIENT:

The corresponding product type is the NLC feature for Mesh V1.1, as detailed in section DICNLCP.

LIGHT_TYPE_NLC_SENSOR:

The corresponding product type is the NLC feature for Mesh V1.1, as detailed in section OCSSNLCP, ALSNLCP and ENMNLCP.

1.4 Version ID(VID) and Product ID(PID) Configuration

Configure file: vendor -> common -> version.h, this file will be also used in compiling codes.

For example:

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

1) PID and VID in Composition data are defined by MESH_PID_SEL，MESH_VID
2) The 3rd to 6th bits in firmware file is the PID and VID here.

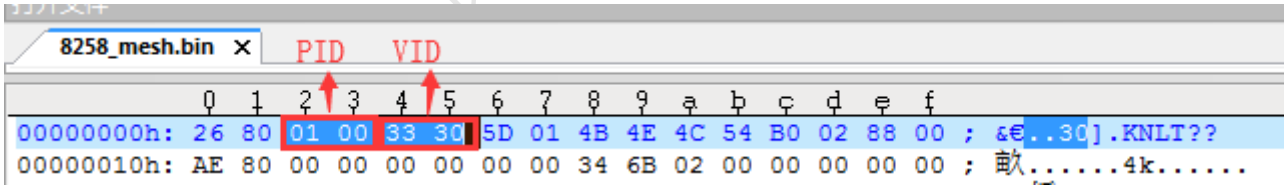


Figure 1.4: PID and VID

3) It showed in ATT UI of general APP in the following way:

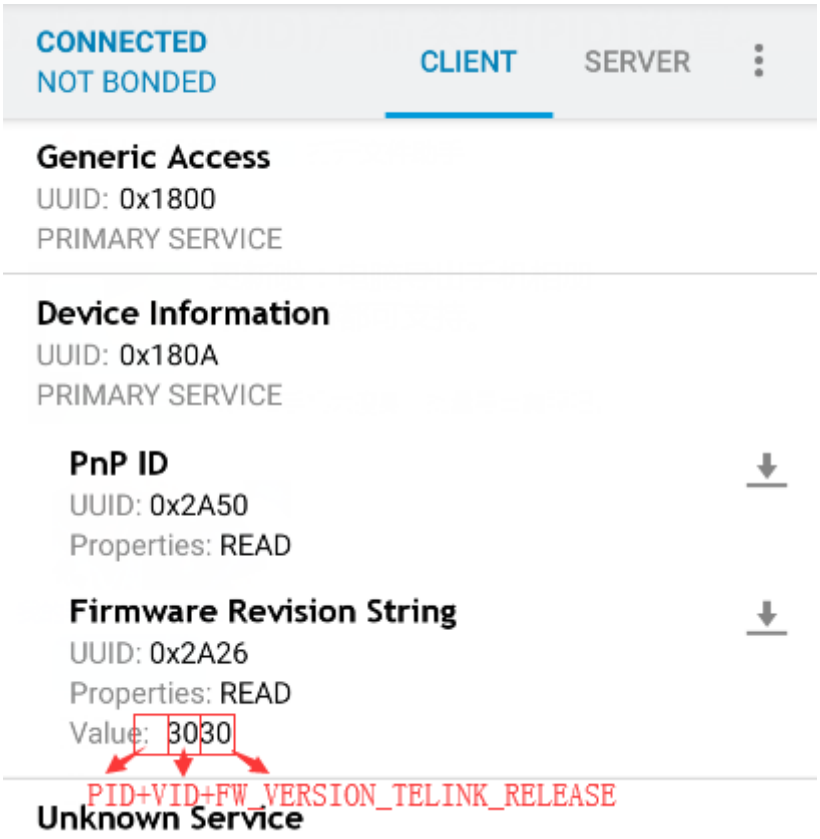


Figure 1.5: ATT user interface

MESH_PID_SEL(PID): general APP will show information in ASCII, so "0x00 0x01" is not visible. Users need to verify it to their own PID.

MESH_VID(VID): "0x33, 0x30" is shown as "30" in ASCII. Users need to verify it to their own PID.

FW_VERSION_TELINK_RELEASE: "0x33, 0x30" is shown as "30" in ASCII. This is the version ID when Telink release the SDK, users should not verify this.

4) Users can verify MESH_PID_SEL and MESH_VID according to their own requirement.

1.5 Mobile App Introduction

1.5.1 App Installation

1.5.1.1 Android App

Get the installation package in the SDK development kit:

```
\telink_sig_mesh_sdk\app\android\TelinkBleMesh\TelinkBleMeshDemo-V4.1.0.0-20231113.apk
```

Or download it through the app "Telink Apps".

1.5.1.2 iOS App

Get it in the App Store by searching telinksigmesh. Or download it through the app "Telink Apps".

1.5.1.3 App Operating Instructions

Refer to the chapter Android and iOS APP User Guide.

1.6 Mesh Application Packet Tx/Rx Processing

1.6.1 Packet Transmission Function

Packet Transmission between Nodes

Data transmitted by invoking the function "mesh_tx_cmd2normal_primary()" follows SIG mesh protocol. Commands such as "access_cmd_on/off ()" are derived by assembling the "mesh_tx_cmd2normal_primary()".

Developers can enable the function "sim_tx_cmd_node2node()" (it's masked by default) to demonstrate command transmission. The effect is: After power on, "ON" command and "OFF" command are automatically and alternately sent with the interval of three seconds. This function will be introduced in detail in subsequent section.

Be careful that whether the sending is successful:

(1) After executing the packet sending function, you need to judge the return value of the function, if it is 0, it means success, otherwise it fails, the corresponding error code, see tx_errno_e in the SDK for details:

```
enum tx_errno_e{
    TX_ERRNO_SUCCESS                    = 0,
    TX_ERRNO_DEV_OR_APP_KEY_NOT_FOUND   = 1,/* device key or app key not found */
    TX_ERRNO_GET_UT_TX_BUF_FAIL         = 2,/* get the upper layer tx buffer fail */
    TX_ERRNO_ADDRESS_INVALID            = 3,/* source address or destination address invalid */
    TX_ERRNO_PAR_LEN_OVER_FLOW          = 4,/* parameters length > 378 */
    TX_ERRNO_TX_BUSY                    = 5,/* segment busy, reliable busy,... */
    TX_ERRNO_TX_FIFO_FULL               = 6,/* tx fifo full: mesh_adv_cmd_fifo_(normal message)
↪   or mesh_adv_fifo_fn2lpn_(message from friend to LPN)*/
    TX_ERRNO_PAR_LEN_LPN_CTL            = 7,/* All transport control messages originated by a
↪   Low Power node shall be sent as Unsegmented */
    TX_ERRNO_IV_INVALID                 = 8,/* have not get iv index after import JSON */
    TX_ERRNO_ALL_OTHER_ERR              = -1,/* default error */
};
```

(2) In the case of sending segment packet, if it fails in the middle of sending the packet, the callback for sending failure is "mesh_seg_block_ack_cb()", see "st_block_ack_t" for the error type. For example, "ST_BLOCK_ACK_BUSY" indicates that the receiver is receiving a segment packet from another node and has not finished. At this time, the client can delay for a period of time according to the situation, and then the application layer will do the retransmission processing.

```
typedef enum{
    ST_BLOCK_ACK_RX_ALL     = 0,    // RX node has received all segments.
    ST_BLOCK_ACK_MISSING    = 1,    // RX node only received some segments.
    ST_BLOCK_ACK_BUSY       = 2,    // RX node is receiving another segment flow, so current tx
↪   segment flow should be stopped and retry later.
    ST_BLOCK_ACK_TIMEOUT    = 3,    // tx segment flow timeout.
    ST_BLOCK_ACK_UNKNOW     = 4,    //
}st_block_ack_t;
```

(3) Before sending a packet, you can determine if it is currently in the tx busy state. Use is_busy_mesh_tx_cmd() to determine this. You can also call the packet sending interface directly first, and then look at the return value.

Master Packet Transmission from Directly-Connected Node to Master

Call bls_att_pushNotifyData(), please refer to <AN_17092701_Telink 826x BLE SDK Developer Handbook> section 3.4.3.10 for more details, this method is used to send the data in any customised format by the customer. However, there is no mesh function, so it is not recommended to use.

Note:

- New UUID should be introduced when employ this method, otherwise it may conflict with current UUID protocol. Users can define new UUID in my_Attributes_provision[]/my_Attributes_proxy[]/my_Attributes[] to define BLE service.

1.6.2 Packet Transmission Flow

Note: red font shows library functions.

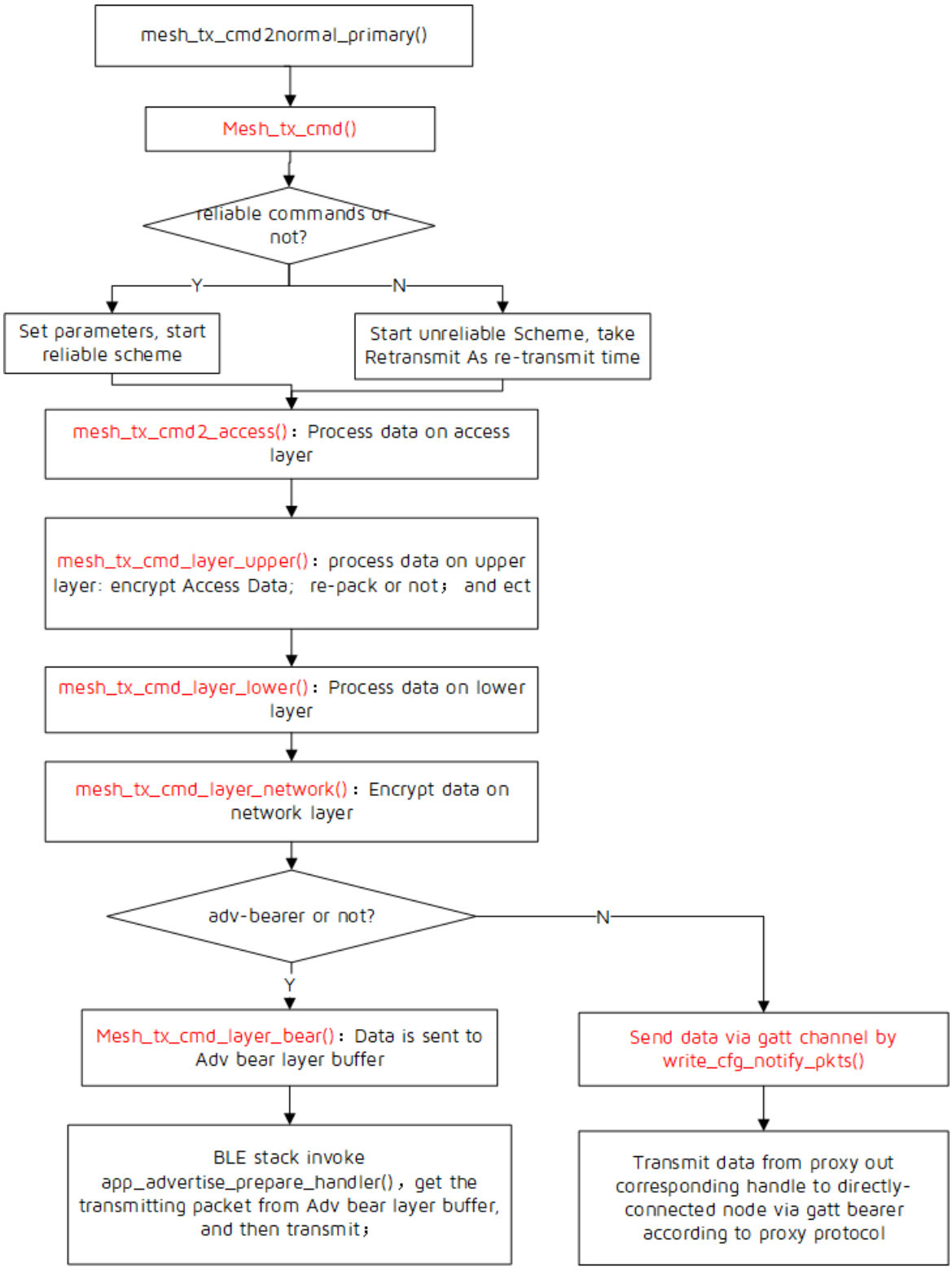


Figure 1.6: Packet Transmission Flow

1.6.3 Packet Reception Flow

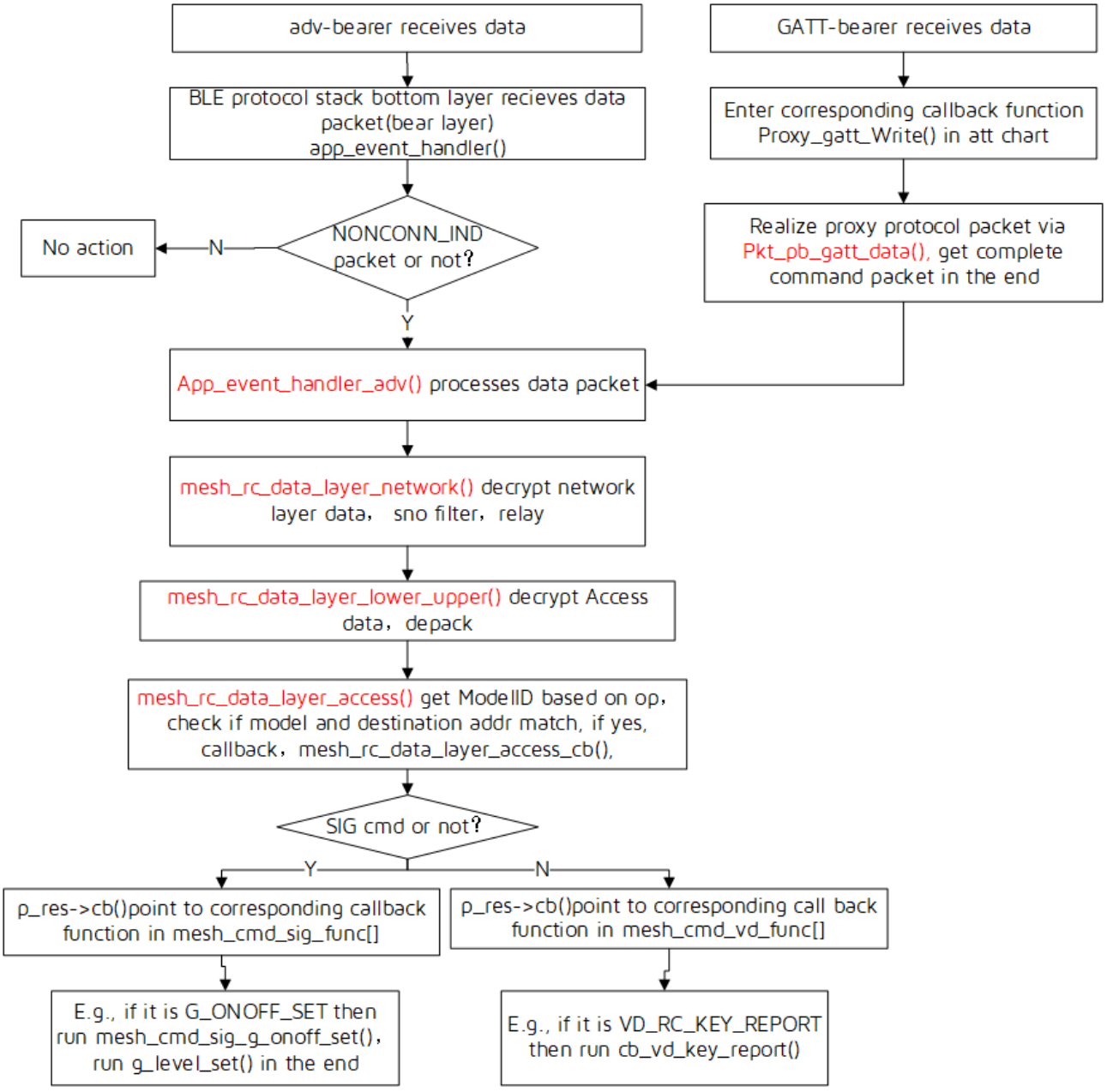


Figure 1.7: Packet Reception Flow

1.6.4 Packet Reception Callback Function Introduction

generic model:

The interface of generic model is in the file "vendor/common/generic_model.c". For callback function, please see the structure "mesh_cmd_sig_func[]". For example, generic on message command. After this command is received, as specified in the packet reception flow, procedure will finally flow to the "mesh_cmd_sig_g_on/off _set()", in which user implements the effect of turning on/off light or setting

gradient parameter. The gradient effect is processed in the "light_transition_proc" and the processing interval is LIGHT_ADJUST_INTERVAL(20ms).

vendor model:

The interface of vendor model is in the file "vendor/common/vendor_model.c". Refer to "mesh_cmd_vd_func[]". User can add vendor command as needed. If such command is received, as specified in the SIG mesh spec, procedure will flow to corresponding callback function, for example, "cb_vd_key_report()".

1.6.5 SIG_mesh Channel

The SIG_mesh supports two types of communication channels:

adv-bearer: Implement mutual communication based on advertising mechanism, no need to establish BLE connection, the communication channel is the standard 37/38/39.

gatt-bearer: Implement communication based BLE connection, the communication channel is 1-36.

1.7 Telink Debug Method Introduction

1.7.1 Tdebug Tool Debugging

This is a stable/reliable debugging method with no effect of MCU performance, it can check/verify global variables on real time; it can also check running status of functions. To check if a function is executed or not, user can define a global variable, then count, and by checking the global variable via Tdebug, user can know if the function is executed or not. For example:

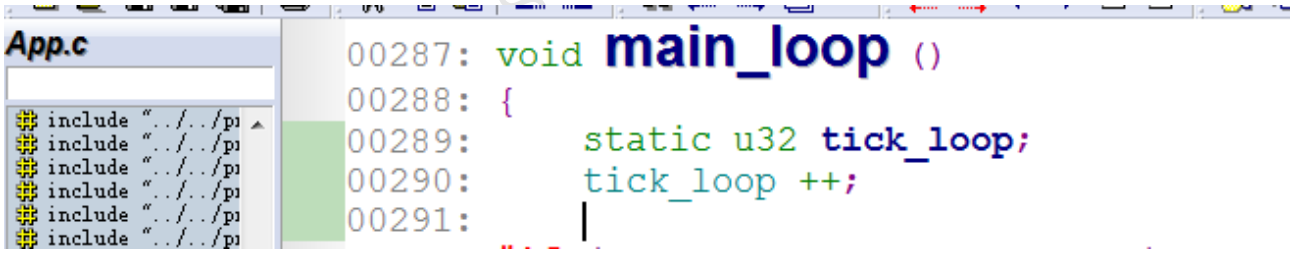


Figure 1.8: Check global variable via Tdebug

Tdebug is described as following, please check "Help" -> "User guide" in BDT tool for detail.

Note:

> Right click in the 8th step to get this manual, as shown in figure below.

Sort the 9th step by name, then find tick_loop (tick_loop is static variable, the name maybe duplicate in the codes, so the compiler add a suffix of .12397 to distinguish), right click, then click Refresh, it will read all global variables and refresh, the value of tick_loop will refresh, too.

If you want to change global variables, change in value chart, then press enter. To read back, right click, then press Refresh.

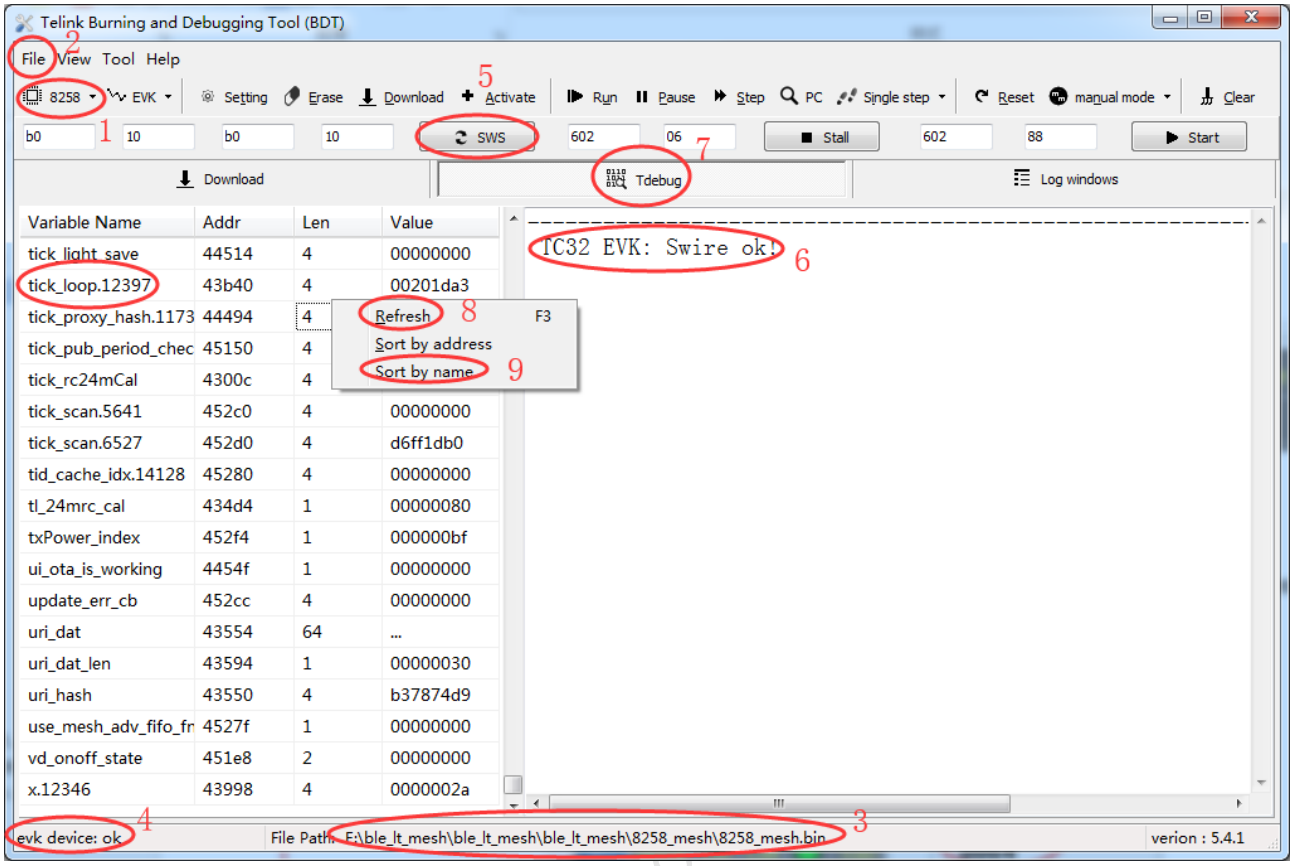


Figure 1.9: Tdebug overview

To read structure variables/arrays longer than 4 bytes but shorter than 1K bytes, click "..." in 1, and the read value will show in 2.

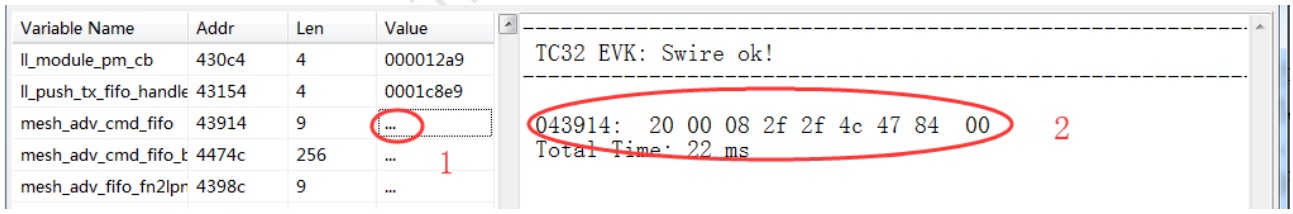


Figure 1.10: Read structure variables or arrays

If it is longer than 1K bytes, a file will be generated and saved to Telink Burning and Debugging Tool -> config -> user -> Read.bin in BDT tool.

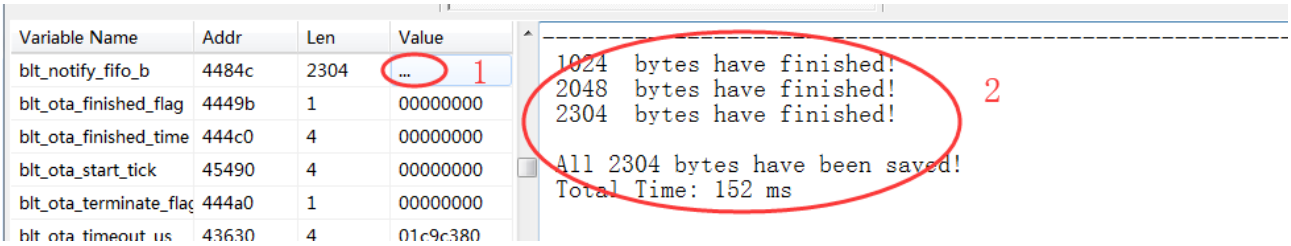


Figure 1.11: Read.bin file

1.7.2 Log Print Debugging

The SDK versions after 2.9 support log print via uart output port simulated by GPIO, the default speed is 1Mbps. Please be noted, to guarantee the correctness of log output, when run log output, irq_disable() is executed by default, so too much log data will slow down MCU performance and RF packet processing, as well as make the mesh unstable, thus harm the debugging procedure. So please use log output as little as possible, and shut as many log as possible after the debug is finished.

For log, you can configure print level(TL_LOG_LEVEL) and print module(TL_LOG_SEL_VAL).

To rule out unnecessary logs in firmware by default, TL_LOG_LEVEL is set to TL_LOG_LEVEL_ERROR, only print level less than or equal to TL_LOG_LEVEL_ERROR will be printed. TL_LOG_LEVEL_LIB is for printing library codes, or important non-library-code log. TL_LOG_LEVEL_USER is for user, and is not in library. It is recommended to use LOG_USER_MSG_INFO() for printing. To use LOG_MSG_INFO(), please verify 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
```

Figure 1.12: Print level

Print module (i.e., TL_LOG_SEL_VAL): to print this module, the corresponding module like TL_LOG_USER should be included.

```
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;
```

Figure 1.13: Print module

The log will be printed only when the corresponding print level and print module meet the requirements.

Note:

> There is a maximum length limit on the length of the data to be printed, when exceeded, the data will be truncated. If you want to avoid truncation, you can change the size of the log_dst[] array.

Log Printing Setup

Step 1 Define HCI_LOG_FW_EN as 1

PRINT_DEBUG_INFO means to use GPIO to simulate UART, and it can only support TX UART, but not RX UART.

Step 2 Set print pin

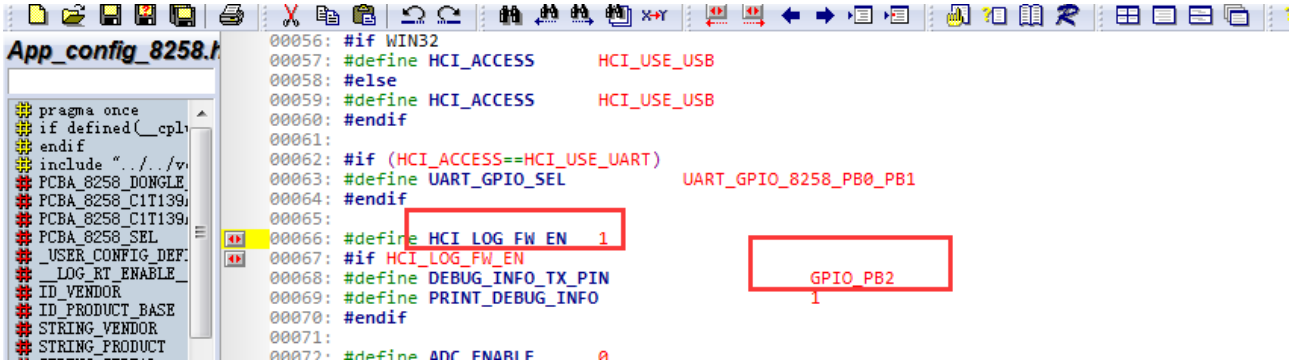


Figure 1.14: Set print pin

Step 3 Set the baud rate. Default we use 1 Mbps.

When simulate UART output by IO, the interrupt is disabled (SIMU_UART_IRQ_EN=1), so the speed of log is the fast the better under this mode, do not reduce band rate.

Note:

> Some USB adapter board may have serious mis-alarming when the speed is under 1Mbps rate, to avoid this, users can reduce UART speed (which will reduce log printing speed), or change USB adapter, such as model CH340G.

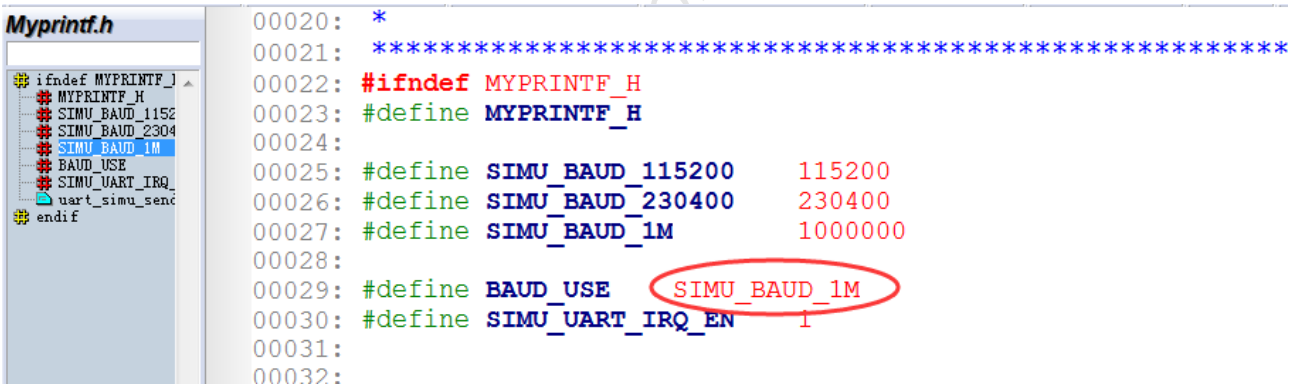


Figure 1.15: Set baud rate

Step 4 Choose log module

In the grading log definition, besides the above print level, there is log module. To print log, print level and log module should both be set to the right value.

To simplify log in Demo SDK, for TL_LOG_SEL_VAL, only TL_LOG_USER is enabled, other log are disabled.

TL_LOG_USER is not be called anywhere by default, to add print, user may run the following commands.

LOG_USER_MSG_INFO(pbuf, len, format,...), where:

- pbuf: for transferring a buffer into character and printing. If there is no, set this to 0.
- len: length of pbuf.

If other API is needed, for example, LOG_MSG_INFO, check TL_LOG_LEVEL first.

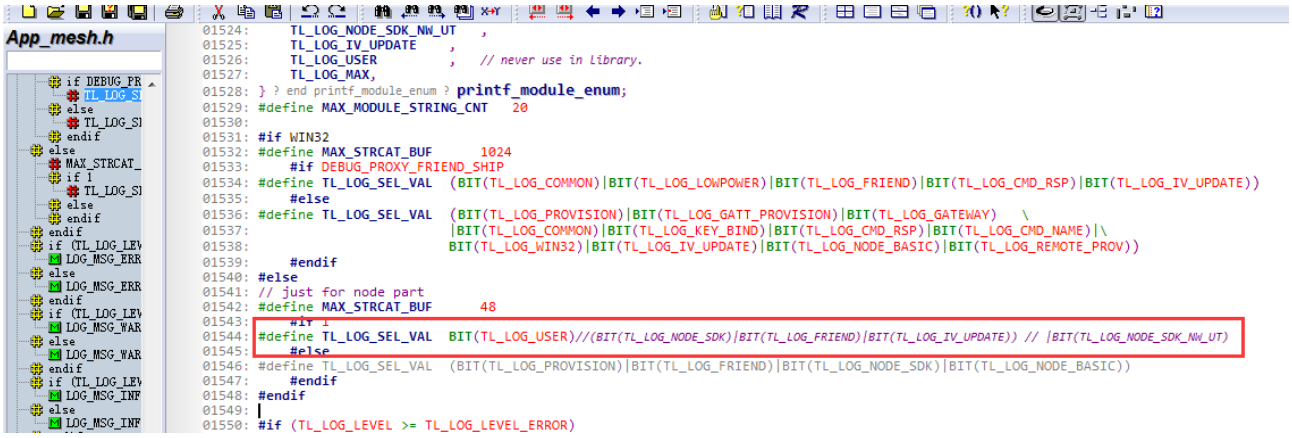


Figure 1.16: Choose log module

Step 5 To open pre-set debug log, set TL_LOG_SEL_VAL, for example:

```
#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 Basic Modules

This part introduces mesh related information, for detail, please refer to MCU Basic Modules in AN_19011501-C2_Telink Kite BLE SDK Developer Handbook.

2.1 Flash and RAM map

2.1.1 Flash Map Introduction

Take the default FlashMapB85m512K and FlashMapB91m1M of Demo SDK for example:

Refer to sdk/doc/SIGMeshFlashmap_yyyymmdd.xlsx for the corresponding flash map documentation.

SIG Mesh B85m 512K flash map

Addr	content	size(K)	Comments
00000 ... 2FFFF	Firmware A (Pingpong OTA)	192	
30000 ... 3FFFF	Sig Mesh Parameters 1	64	
40000 ... 6FFFF	Firmware B (Pingpong OTA)	192	
70000 ... 75FFF	Sig Mesh Parameters 2	24	
76000 ... 76FFF	Mac (0x76000-0x76005)	4	
77000 ... 77FFF	Frequency Offset (0x77000) FLASH_ADR_STATIC_OOB(0x77800)	4	static OOB is optional
78000 ... 7FFFF	User Parameters	32	For User define

Figure 2.1: FlashMapB85m512K

For more information about "Sig Mesh Parameters 1", please check the SDK's FLASH_ADR_MESH_KEY, FLASH_ADR_MD_CFG_S .....

For more information about "Sig Mesh Parameters 2", please check the SDK's FLASH_ADR_MD_VD_LIGHT, FLASH_ADR_MD_SCENE …….

SIG Mesh B91m 1M flash map

Addr	content	size (K)	Comments
00000 … 5FFFF	Firmware A (Pingpong OTA)	384	
60000 … 7FFFF	reserved for future use	128	should not be used to store parameters unless without flash protection funtion
80000 … DFFFF	Firmware B (Pingpong OTA)	384	
E0000 … EFFFF	Sig Mesh Parameters 1	64	refer to table on the right for details
F0000 … F5FFF	Sig Mesh Parameters 2	24	refer to table on the right for details
F6000 … FDFFF	User Parameters	32	For User define
FE000 … FEFFF	Frequency Offset (0xfe000) FLASH_ADR_STATIC_OOB(0xfe800)	4	static OOB is optional
FF000 … FFFFF	Mac (0xff000-0xff005)	4	

Figure 2.2: FlashMapB91m1M

For more information about "Sig Mesh Parameters 1", please check the SDK's FLASH_ADR_MESH_KEY, FLASH_ADR_MD_CFG_S …….

For more information about "Sig Mesh Parameters 2", please check the SDK's FLASH_ADR_MISC, FLASH_ADR_RESET_CNT …….

2.1.2 RAM map (8258 64K)

Check ./boot.link for detailed configuration file.

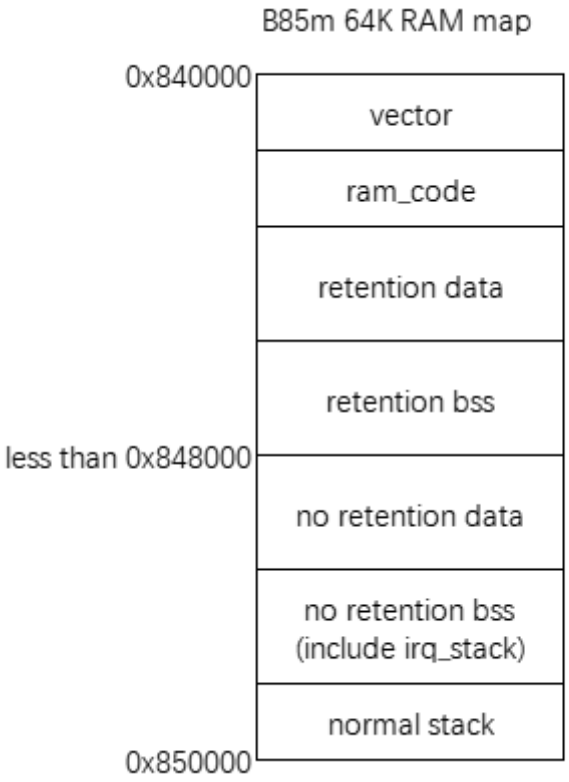


Figure 2.3: RAM Map

- data(retention)：contains variables with attribute_data_retention prefix, and all global variables whose initial value is not 0, default attribute is retention data; (data is shown in the list file of the B85m SIG mesh SDK, not retention_data).
- bss(retention)： contains variables with attribute_bss_retention prefix, and all global variables whose initial value is 0, default attribute is retention bss; ; (bss is shown in the list file of the B85m SIG mesh SDK, not retention_bss).
- data(no retention)： contains global variables with attribute_no_retention_data prefix. (Note that, for some reason, in the list file of the B85m SIG mesh SDK it shows retention_data instead of no_retention_data).
- bss(no retention)： contains global variables with attribute_no_retention_bss. Also contains irq_stack, whose size is defined by IRQ_STK_SIZE, default is 0x300, the size is smaller than 0x200 in demo SDK. (Note that, for some reason, in the list file of B85m SIG mesh SDK it shows retention_bss instead of no_retention_ bss).
- normal stack：after bss, before 64K RAM. The initial value is 0xffffffff，to speed up RAM initialization, only 3K RAM will be initialized.

Note:

- The end address of "retention bss" needs to be less than 0x848000 when working in retention sleep mode. This is because the maximum retention RAM is 32KB.
- The prefixes attribute_bss_retention and attribute_no_retention_bss cannot be used for variables that are not initialized to 0. Otherwise the initialization value is invalid and defaults to 0.

2.2 Checking of Stack Overflow and Retention RAM Overflow

Take B85m as example:

2.2.1 Checking Method of Stack Overflow

In order for the stack not to overflow, first of all, make sure that the end address of the no retention bss is less than the end address of the RAM, that is, leave space for a normal stack.

The end address of no retention bss can be used with the tdebug Tool to sort variables by address, and the address after the last variable, is the end address.

It can also be calculated by the *.lst file generated by compiling.

Then, test all functions, check if normal stack and irq_stack overflow.

2.2.1.1 Checking Method of Normal Stack Overflow

The initial value is 0xffffffff. The demo SDK need a stack of around 2.5KB, because all used stack is less than 3KB, to speed up RAM initialization, only 3KB RAM will be initialized. Check by reading RAM, if the 61KB address, i.e., 0x84F400–0x84F403 is not 0xFFFFFFFF, that means the stack is over 3KB, and there is a possibility that the Normal stack has overflowed. To further confirm whether the stack has overflowed, you can initialize the entire normal stack by modifying it as follows:

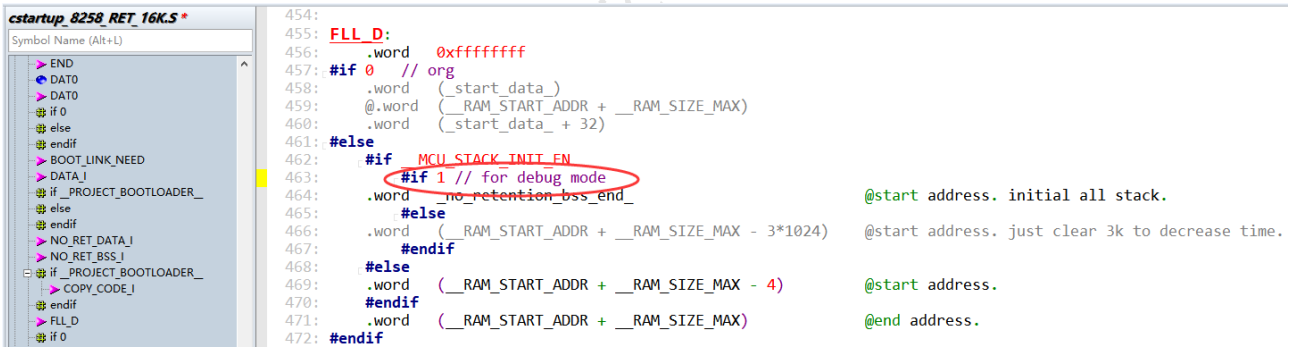


Figure 2.4: stack_debug_mode

Where no_retention_bss_end is the same as the VMA address of the sdk_version below. Check by reading RAM and if you find that the 4 bytes at the no_retention_bss_end location is not 0xFFFFFFFF, it means that the Normal stack overflowed.

8258_mesh.lst

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .vectors      00000210  00000000  00000000  00008000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .ram_code     000021d4  00000210  00000210  00008210  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         0001a7c0  000023f0  000023f0  0000a3f0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00001d18  0001cbb0  0001cbb0  00024bb0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         00000260  00842d00  0001e8c8  0002ad00  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          00003d94  00842f60  0001eb28  0002af60  2**2
                  ALLOC
  6 .retention_data 0000000c  00846cf4  0001eb28  0002ecf4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .retention_bss 00000154  00846d00  0001eb34  0002ed00  2**2
                  ALLOC
  8 .sdk_version  00000024  00846e54  0001eb34  0002ee54  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .TC32.attributes 00000010  00000000  00000000  0002ee78  2**0
                  CONTENTS, READONLY
 10 .comment      0000001a  00000000  00000000  0002ee88  2**0
                  CONTENTS, READONLY
```

Figure 2.5: address_no_retention_bss_end

2.2.1.2 Checking Method of irq_stack Overflow

Initialize the value to 0x00000000, check the irq_stk[] variable by tdebug and observe the usage; if you find that the 4 bytes of irq_stk[0–3] are non-zero, it means that it has overflowed.

2.2.2 RAM Remaining Size Analysis

According to the light_8258.lst file generated by the compiler (in the same directory as the bin file) the analysis is as follows:

(Note that for some reason the retention_data shown in the list file for the B85m SIG mesh SDK refers to no_retention_data and the retention_bss refers to no_retention_bss)

8258_mesh.lst

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .vectors      00000210  00000000  00000000  00008000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .ram_code     000021d4  00000210  00000210  00008210  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         0001a7c0  000023f0  000023f0  0000a3f0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00001d18  0001cbb0  0001cbb0  00024bb0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         00000260  00842d00  0001e8c8  0002ad00  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          00003d94  00842f60  0001eb28  0002af60  2**2
                  ALLOC
  6 .retention_data 0000000c  00846cf4  0001eb28  0002ecf4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .retention_bss 00000154  00846d00  0001eb34  0002ed00  2**2
                  ALLOC
  8 .sdk_version  00000024  00846e54  0001eb34  0002ee54  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .TC32.attributes 00000010  00000000  00000000  0002ee78  2**0
                  CONTENTS, READONLY
 10 .comment      0000001a  00000000  00000000  0002ee88  2**0
                  CONTENTS, READONLY
```

Figure 2.6: lst file

Idx Name: Segment name

Size：Size of bytes occupied by this segment

VMA：Actual running address

LMA：Storage address in flash

Remaining RAM = end address of RAM - address of last byte of no retention bss. (If there is a retention_data or retention_bss segment, the end address of the retention_bss is used)

The 825X RAM start address is 0x840000 and the 8258 RAM size is 64KB, so the RAM end address is 0x850000.

The way to get the address of the last byte of the bss:

For SDKs with an sdk_version section, the last byte of the bss is equal to the first address of the sdk_version. The sdk_version does not take up any RAM space, because it is not loaded from flash to RAM in the cstartup.

The SDKs without an sdk_version section can be obtained in one of two ways:

light_8258.lst

```
16 Idx Name          Size      VMA       LMA       File off  Algn
17   0 .vectors      000001a0  00000000  00000000  00008000  2**4
18                   CONTENTS, ALLOC, LOAD, READONLY, CODE
19   1 .ram_code     000016d0  000001a0  000001a0  000081a0  2**2
20                   CONTENTS, ALLOC, LOAD, READONLY, CODE
21   2 .text         0000c004  00001870  00001870  00009870  2**4
22                   CONTENTS, ALLOC, LOAD, READONLY, CODE
23   3 .rodata       00000468  0000d874  0000d874  00015874  2**2
24                   CONTENTS, ALLOC, LOAD, READONLY, DATA
25   4 .data         000006d0  00842200  0000dcdc  0001a200  2**2
26                   CONTENTS, ALLOC, LOAD, DATA
27   5 .bss          00001869  008428d0  0000e3ac  0001a8d0  2**4
28                   ALLOC
29   6 .TC32.attributes 00000010  00000000  00000000  0001a8d0  2**0
30                   CONTENTS, READONLY
31   7 .comment      0000001a  00000000  00000000  0001a8e0  2**0
32                   CONTENTS, READONLY
```

Figure 2.7: lst file

(1) bss VMA (i.e. bss start address) + bss size, for example, the above figure is: (0x8428d0 + 0x1869) = 0x844139.

If there is a retention_data or retention_bss segment, the end address of the retention_bss is used, i.e. retention_bss VMA (i.e., retention_bss start address) + retention_bss size.

Note: The retention_data and retention_bss here are actually no retention RAM zones, it is for some reason that boot.link needs to name the data and bss as retention zones, as well as the noretention_data and noretention_bss are named as retention_data and retention_bss zones.

(2) With the BDT tool the variables are sorted by address, the address of the last variable + the size of this variable, as follows. 0x844138 + 1 = 0x844139, (the high "8" has been omitted in the BDT tool)

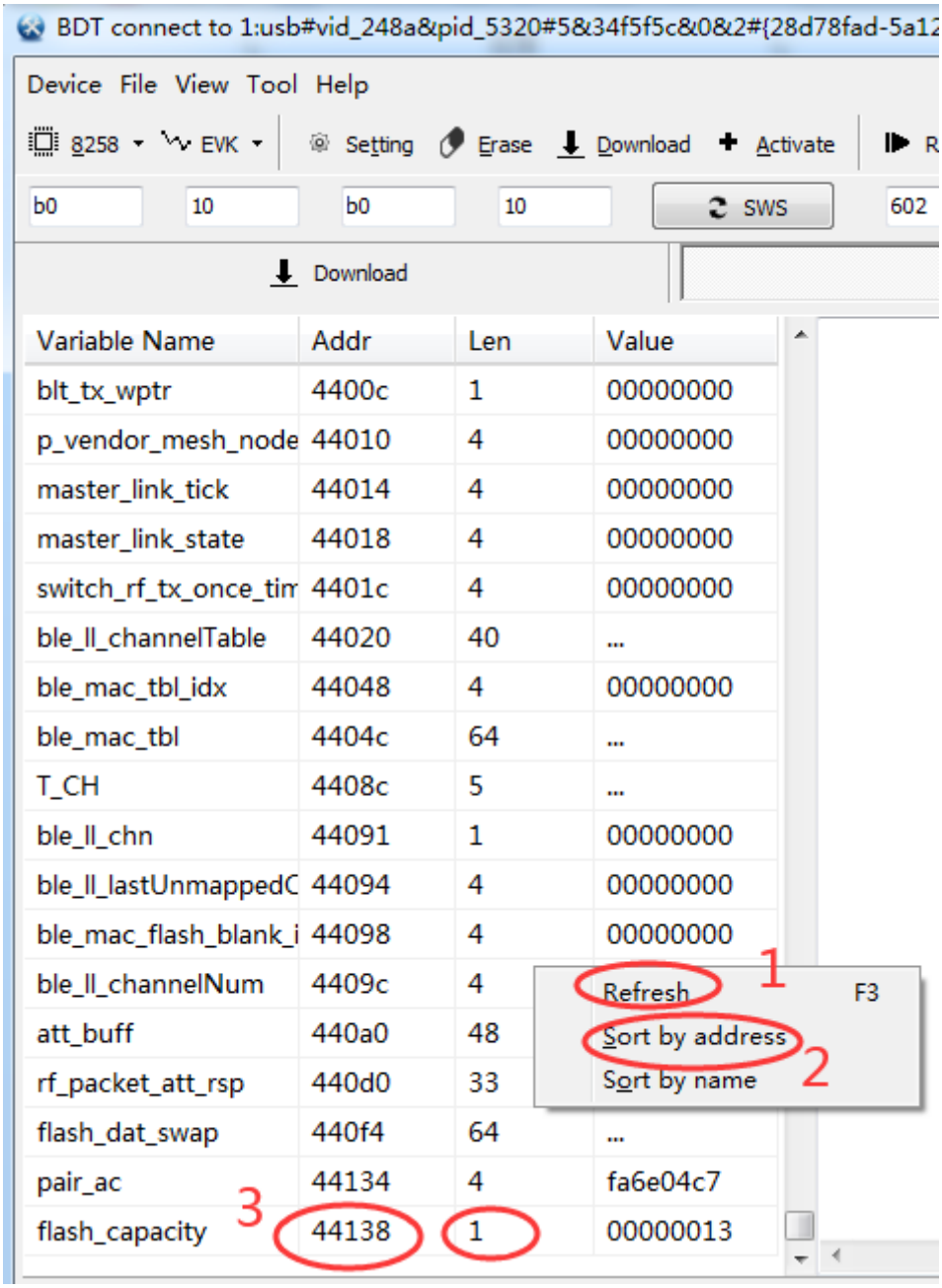


Figure 2.8: RAM_sort

According to the above chart, the remaining RAM of this firmware = 0x850000 - (0x8428d0 + 0x1869) = 0xBEC7 = 47.7K.

It should be noted that not all of the remaining RAM can be used directly, for this remaining RAM we need to allocate a section to the normal stack (for the B85m SIG mesh stack it requires a reserved amount of not less than 2.5k, for the B91m SIG mesh stack requires a reserved amount of not less than 4k, for the private mesh stack it requires a reserved amount of not less than 512). Taking the B85m SIG mesh in the above figure as an example, it means that the real RAM that can be used in this firmware is about 47.7-2.5 = 45.2K.

2.2.3 Checking Whether The Stack Overflows Using 8258 as An Example

RAM Overflow Check:

RAM overflow occurs when the stack usage is beyond the remaining RAM area, and out-of-bounds use of the BSS area or other RAM areas causes the system to crash.

In this system, there are two stack areas, one is the normal stack and the other is the irq stack. The use of the stack is indeterminate and depends mainly on the depth of the function calls that have actually been run and the size of the local variables used in the function.

2.2.3.1 Checking Whether The Normal Stack Overflows

(1) A simple way is: the remaining RAM size (please refer to 1.1 for calculation), the SIG mesh SDK should be larger than 2.5k, and the private mesh should be larger than 512, this is an empirical judgment, no actual read confirmation.

(2) Another way is: test all the functions once, (the function with trigger reboot cannot be included), and then check if there is any overflow in the normal stack, i.e., after checking the address at the end of the bss if there is still a consecutive area of 0xFF (the initial value of the normal stack is 0xFF), and then check if there is any overflow, if yes, then the RAM is not overflowed. Check the address after the end address of the bss, for example, check the last 4K. (The unused RAM before 3K, that is, before 0x84F400, is not initialized to speed up initialization, so it will be a random number). However, none of our stacks should exceed 3K, so only focus on 3K. The reason for reading 4K is mainly for the address 4K alignment, that is, in the read to the document inside the address of the lower 12bit and the actual address is the same, so it is more convenient to view. View as follows: (press the "Tab" key in the "84f000" control inside), the generated file in the BDT tool is located in the directory ". /BDT/release_v5.4.4/config/user/", e.g. ". /BDT/release_v5.4.4/config/user/log_17_17_48_addr0x0084f000.bin". Because the BDT tool has to do judgment, when the length of the read data is greater than or equal to 1K, it will be automatically saved as a file, the format of the file name is "log + timestamp + start address".

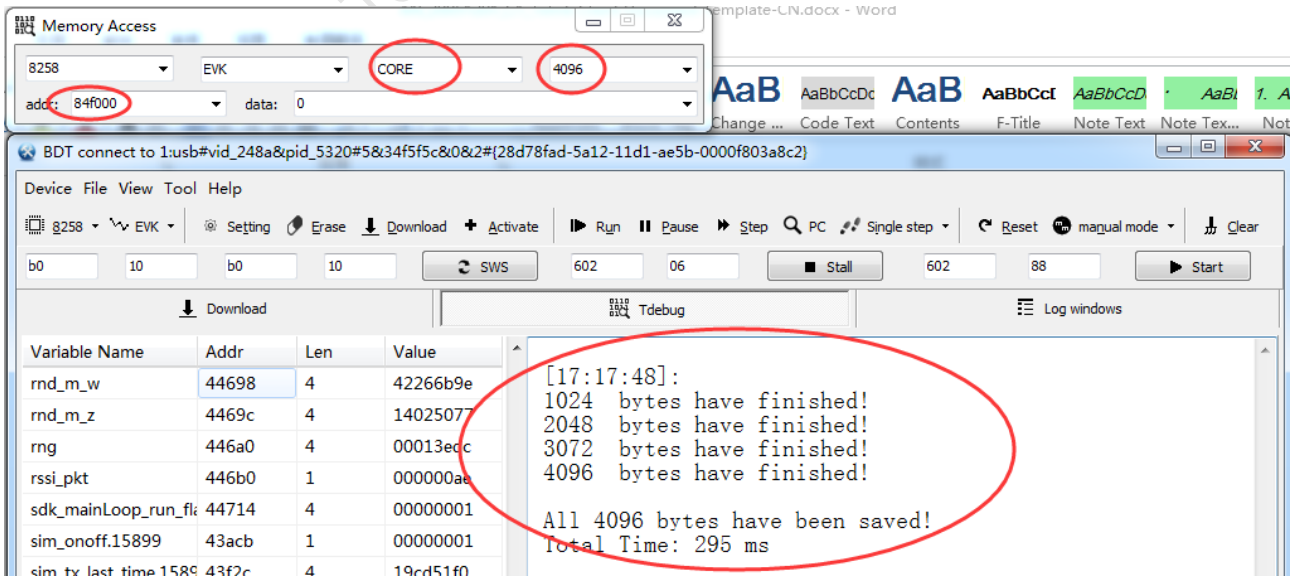


Figure 2.9: RAM_read

The read file, for example as below, where 84F000+0xF0C is the maximum stack usage for the current run. The stack usage is about 0x100. We see that there are many more consecutive "0xFFFFFFFF", so the stack is not overflowing.

log_17_17_48_addr0x0084f000.bin

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000e30h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e40h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e50h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e60h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e70h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e80h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000e90h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000ea0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000eb0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000ec0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000ed0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000ee0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000ef0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
00000f00h: FF FF FF FF FF FF FF FF FF FF FF FF 04 00 00 00 ;
00000f10h: 3C 2C 84 00 10 04 00 00 00 00 00 00 3C FF 84 00 ; <,?........<  ?
00000f20h: 00 00 00 00 60 28 84 00 00 00 00 00 D6 7B 00 00 ; ....`(?....謠..
00000f30h: AC 22 84 00 58 FF 84 00 00 00 00 00 00 00 00 00 ; ??X  ?........
00000f40h: 5C FF 84 00 64 28 84 00 64 28 84 00 D7 25 84 00 ; \  ?d(?d(???
00000f50h: 0C D7 00 00 12 00 00 00 F4 7F 00 00 30 22 84 00 ; .?.....?..0"?
00000f60h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000f70h: 64 28 84 00 78 28 84 00 D0 3A 84 00 04 00 00 00 ; d(?x(???....
00000f80h: B8 FF 84 00 10 04 00 00 13 00 00 00 B1 07 00 00 ; ??........?..
00000f90h: 74 25 84 00 00 00 00 00 00 D4 6E 49 01 00 00 00 ; t%?.....詎I....
00000fa0h: 82 40 00 00 01 00 00 00 74 25 84 00 00 00 00 00 ; 俜......t%?....
00000fb0h: 00 D4 6E 49 00 00 00 00 F0 41 00 00 01 00 00 00 ; .詎I....餉......
00000fc0h: 00 00 00 00 00 D4 6E 49 00 00 00 00 DC 42 00 00 ; .....詎I....歷..
00000fd0h: 00 00 00 00 20 06 80 00 20 00 80 00 00 00 00 00 ; .... .€. .€.....
00000fe0h: 20 00 80 00 00 00 00 00 86 06 00 00 EA 28 00 00 ;  .€.....?..?..
00000ff0h: 5C 02 00 00 00 00 00 00 00 00 00 00 0A 01 00 00 ; \...............
```

Figure 2.10: RAM_result

2.2.3.2 Checking Whether The Irq Stack Overflows

The irq stack is used by interrupt function, the current configured size is IRQ_STK_SIZE, 0x300, (this size is not recommended for customer to change smaller), all interrupt callback function will also use this stack. So customer should avoid defining very large local variables in the interrupt callback function.

Checking irq stack is similar to the second method of checking normal stack, test all the functions once, (the function with trigger reboot cannot be included), then check whether there is still a continuous 0x00 area in this buf of irq_stk in BDT (the initial value of irq stack is 0x00), and if yes, it means that there is no overflow of RAM. It shows in the following figure:

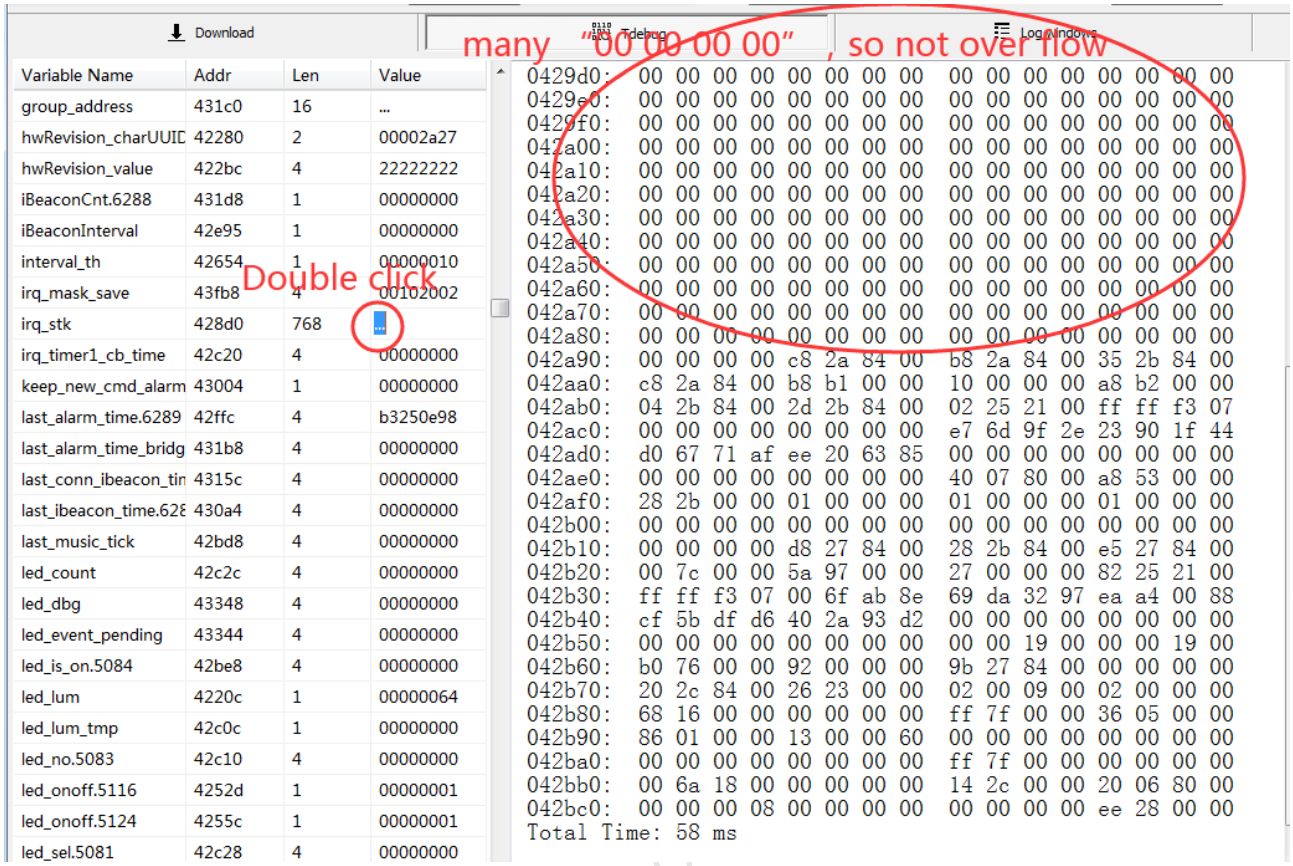


Figure 2.11: IRQ_stack_check

2.2.4 Size Calculation of Retention RAM

For SIG mesh SDK, if the retention function is turned on, i.e. the macro PM_DEEPSLEEP_RETENTION_ENABLE is enabled, when compiling, it will be checked by boot.link document: if the retention use is more than 32KB, then an error will be reported. It is shown in the following figure:

```
./drivers/8258/lpc.o ./drivers/8258/qdec.o ./drivers/8258/random.o ./drivers/8258/s7816.o ./drivers/8258/spi.o ./drivers/8258/timer.o
./drivers/8258/uart.o ./drivers/8258/watchdog.o  ./drivers/8258/flash/flash_mid011460c8.o ./drivers/8258/flash/flash_mid1060c8.o
./drivers/8258/flash/flash_mid11325e.o ./drivers/8258/flash/flash_mid1160c8.o ./drivers/8258/flash/flash_mid13325e.o
./drivers/8258/flash/flash_mid134051.o ./drivers/8258/flash/flash_mid136085.o ./drivers/8258/flash/flash_mid1360c8.o
./drivers/8258/flash/flash_mid1360eb.o ./drivers/8258/flash/flash_mid14325e.o ./drivers/8258/flash/flash_mid1460c8.o
./boot/8258/cstartup_8258_RET_16K.o   -llt_8258_mesh -lsoft-fp -lsig_mesh_825x
C:\TelinkSDK\opt\tc32\bin\tc32-elf-ld.exe: **Retention Area Overflow, Check your data/bss usages**
C:\TelinkSDK\opt\tc32\bin\tc32-elf-ld.exe: **Retention Area Overflow, Check your data/bss usages**
C:\TelinkSDK\opt\tc32\bin\tc32-elf-ld.exe: **Retention Area Overflow, Check your data/bss usages**
make: *** [firmware.elf] Error 1
```

Figure 2.12: retention_ram_size_overflow

Calculate the size for Retention RAM according to the list documentation:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .vectors      00000210  00000000  00000000  00008000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .ram_code     00002bb4  00000210  00000210  00008210  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         0001a0dc  00002dd0  00002dd0  0000add0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00003050  0001ceac  0001ceac  00024eac  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         00000240  00843700  0001fefc  0002b700  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          0000243a  00843940  0002013c  0002b940  2**4
                  ALLOC
  6 .retention_data 00000004  00845d7c  0002013c  0002dd7c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  7 .retention_bss 00000114  00845d80  00020140  0002dd80  2**2
                  ALLOC
  8 .TC32.attributes 00000010  00000000  00000000  0002dd80  2**0
                  CONTENTS, READONLY
  9 .comment      0000001a  00000000  00000000  0002dd90  2**0
                  CONTENTS, READONLY
```

Figure 2.13: retention_ram_list

The retention area contains vectors, ramcode, data, bss segments.

Note that in the mesh SDK (mesh SDK only), data and bss are placed in the retention area by default, i.e., they can be defined without the attribute_data_retention or attribute_bss_retention prefixes, and for some other reasons, the retention data and retention bss shown above are actually non-retention, only using the name.

So the end address of the retention area is calculated as: .bss VMA (i.e. bss start address) + bss size, for example, the above figure is: (0x843940 + 0x243a) = 0x845d7a; Retention size is: 0x845d7a - 0x84000 = 0x5d7a = 23.4k;

2.3 Startup File cstartup.s and Link File boot.link

Unlike the BLE base SDK, the mesh SDK uses only one copy of cstartup.S and boot.link. Take B85m as an example, B85m only uses cstartup_8258_RET_16K.S, which is applicable to the configuration of 16K retention and 32K retention. Because it has already done the automatic identification of the size, the space after the end address of the retention data is used as an ordinary no retention RAM for storing the no retention data/bss, etc.

2.4 Clock

The MCU clock is defined by CLOCK_SYS_CLOCK_HZ. The default value is 32MHz for B85m mesh project and 48MHz for B85m gateway project; the default is 48MHz for both B91m mesh and gateway projects.

2.4.1 System clock & System Timer

The system clock is the clock of MCU programs.

The system timer is a read-only timer, providing timing for BLE time sequence, as well as for users.

For Telink 826x IC, the time source of system timer is system clock; for Telink 8x5x IC, system timer is separated with system clock. As shown in figure below, system timer is 16M, generated by 2/3 divider from the external 24M Crystal Oscillator.

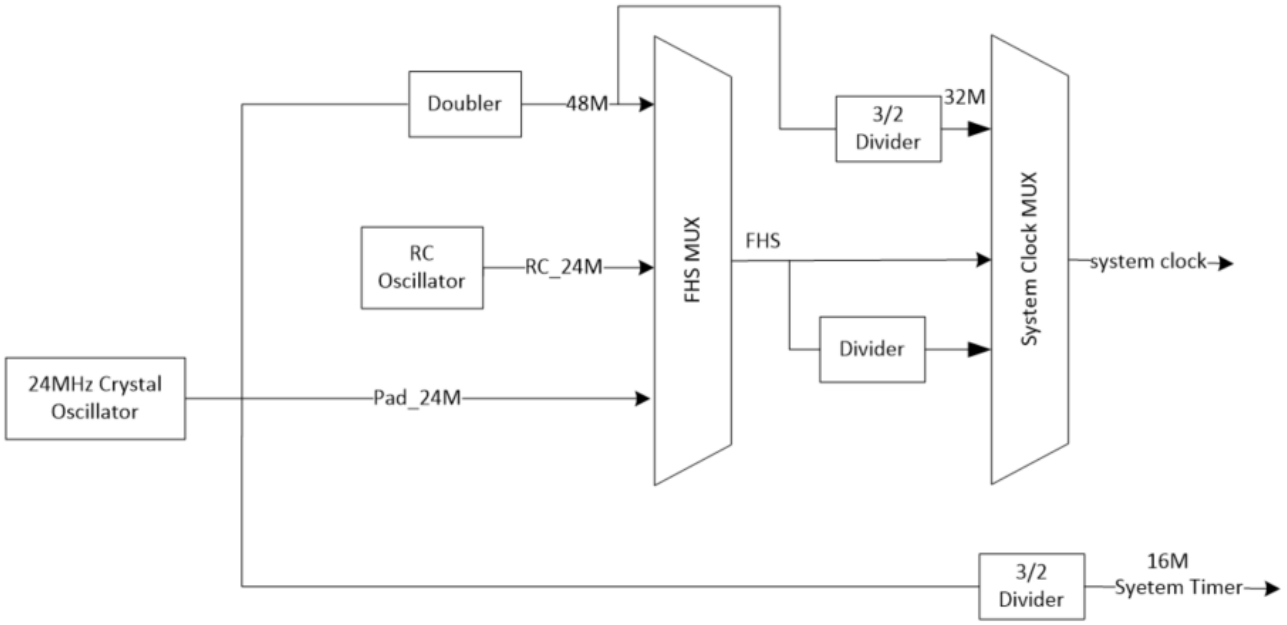


Figure 2.14: System Clock & System Timer

As can be seen in above figure, the external 24M Crystal Oscillator will be double to 48M, then divided by the dividers and generate 16M/24M/32M/48M as system clock, such clocks are called crystal clock (e.g., 16M crystal system clock, 24M crystal system clock); internal 24M RC Oscillator can also generate 24M RC clock, 32M RC clock and 48M RC clock, which we call RC clock(BLE SDK does not support RC clock).

For BLE SDK, we recommend crystal clock.

Call the following API configuration system clock when initialization, choose corresponding clock in the definition of enum variable SYS_CLK_TYPEDEF.

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

8x5x System Timer is different from system clock, user should know the clock source of each hardware module in MCU, whether it is system clock or system timer. The following case is an example, where the system clock is 24M crystal, system clock is 24M, and system timer is 16M.

In app_config.h, the definitions of system clock and the corresponding S, mS, uS are shown as below:

```
#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),
};
```

All hardware modules whose clock source is system clock must use only CLOCK_SYS_CLOCK_HZ, CLOCK_SYS_CLOCK_1S when set module clock; in other words, if the module's clock set is the clock defined above, then the clock source of the module is system clock.

For example, in PWM driver, PWM cycle and interval are set as following, means the clock source of PWM is system clock.

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

System Timer is the fixed 16M, so for this timer, the s, ms, us are defined as following in SDK code:

```
//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,
};
```

The following system timer related API should use similar CLOCK_16M_SYS_TIMER_CLK_xxx to define time, as shown below.

```
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)
```

System Timer is BLE timing standard, so, all BLE timing related parameters and variables should use CLOCK_16M_SYS_TIMER_CLK_xxx to define time.

2.4.2 System Timer Usage

System Timer will start working after cpu_wakeup_init in Main function finishes initialization, user can read the value of System Timer tick.

System Timer tick is 32bit long, it will increase by 1 for each time cycle, i.e, 1/16 us, the value is range from 0x00000000 to 0xffffffff. The value of tick is 0 when system boot, it takes about (1/16) us * (2^32) = 268s to reach the maximum value, i.e., System Timer tick repeats the cycle every 268s.

System tick will not stop when MCU is running.

Users can read System Timer tick via clock_time() function.

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

The whole BLE timing sequence is designed based on System Timer tick, and System Timer tick is widely used in the program for timing and over timing record, it is highly recommended to use System Timer tick for timing and over timing determination.

For example, a simple software timing, it is based on inquiry, with moderate real-time character and accuracy, it is for application with lower requirement for time error. Method:

1) start timing: set a u32 variable, read and record current System Timer tick.

```
u32 start_tick = clock_time();    // clock_time() return System Timer tick value
```

2) Check the difference between current System Timer tick and start_tick, see if it is surpass the timing value, if it is, then the timer is triggered, the program will take corresponding action, and the timer will be cleared or start a new timing cycle depends on requirement.

Suppose the timing time is 100 ms, to inquire if the time is up in the following way:

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

The limiting case of the system clock tick going from 0xffffffff to 0 is solved by converting the difference value to a u32 type.

To deal with the u32 issue caused by different system clock, the SDK provides a unified function, users can use the following function to inquire for any system clock source.

```
if( clock_time_exceed(start_tick, 100 * 1000))  //the unit for the second parameter is us
```

Please be noted, one cycle of 16M clock is 268s, so this function is only applicable to timing no more than 268s. Timing longer than 268s need an extra software counter.

For example, after 2s A is triggered (for only once), the program will take B action.

```
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 Introduction

See Summary of mesh_1.1_feature for mesh-related spec downloads.

This section follows the chapter order in MshPRFv1.0.1.pdf, this is only a brief, for detail, please refer to respective chapters in SIG MESH spec.

3.1 Layered architecture

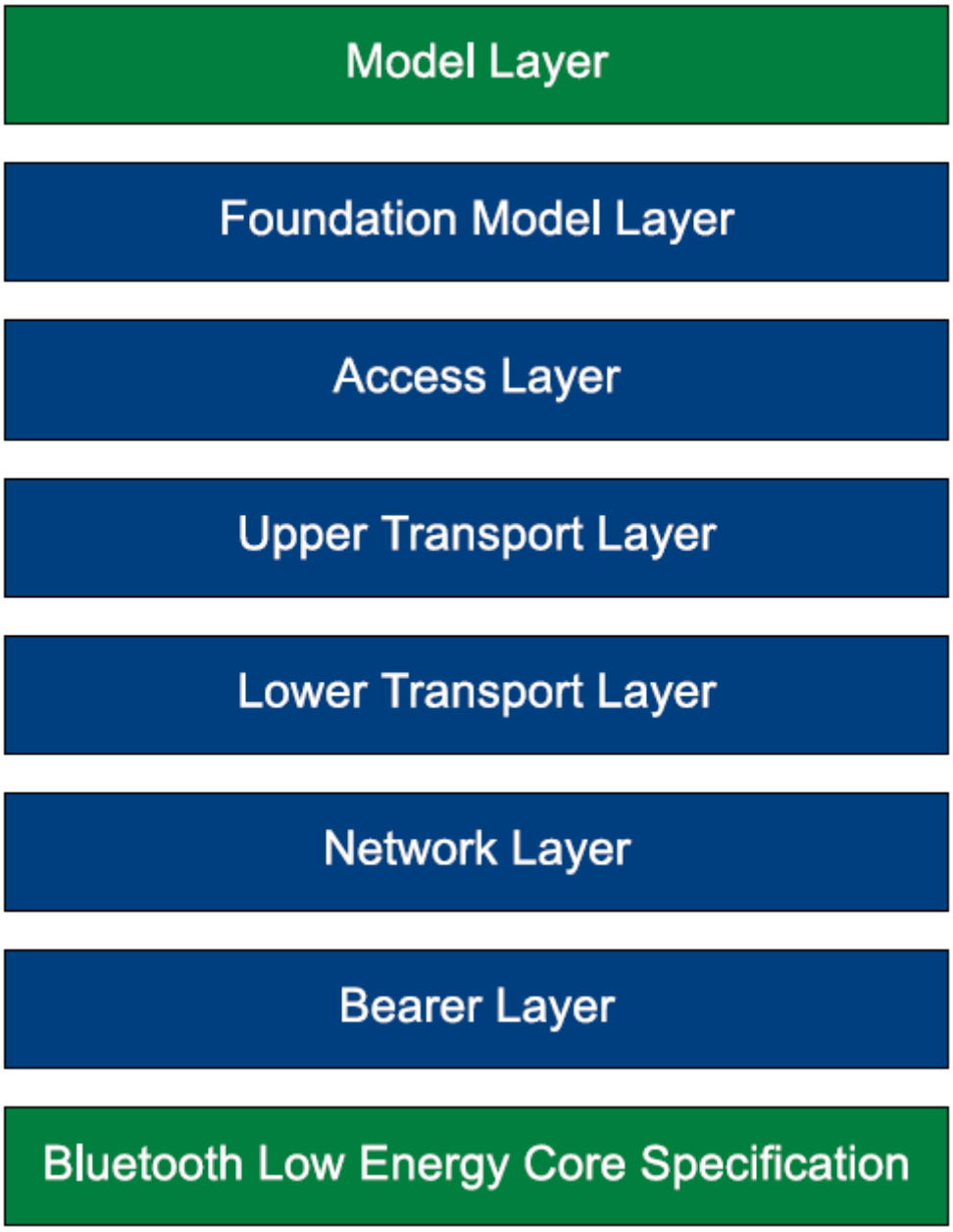


Figure 3.1: Layered Architecture

3.1.1 Model layer

The Model defines a node supporting function, each model defines its own op code and status. E.g., generic on/off model defines Generic ON/OFF/GET/STATUS.

When provision, provisioner will get all model id that the node supports via get composition data, and get to know what functions the node supports. Only when the node supports a certain model, the corresponding op code defined by the same model will be sent to the node.

There are 2 kinds of models, server model and client model.

Server model: it is a model which can be controlled. It has its own status, can be changed/obtained by other nodes, e.g., on/off server model can receive on/off set/get command, can response to on/off status command, but it cannot send out on/off set/get command, nor handle on/off status command.

Client model: it can control server node, it has no status of its own. E.g., on/off client model, it can send out on/off set/ get command, it can also handle received on/off status command, but it cannot send out on/off status command, nor handle on/off set/get command.

3.1.2 Foundation Model layer

Foundation Model is similar to model, it is basic model, contains Configuration Server model，Configuration Client model，Health Server model，Health Client model.

All configured nodes must contain Configuration Server model, all provisioner must contain Configuration Client model. The 2 models contain subscription add/delete op code, and both of the models' access layers are encrypt with device key, so only provisioner node can send out set/get command of configuration model.

3.1.3 Access layer

Combine op code with parameter in prescribed format.

3.1.4 Transport layer

Decrypt/encrypt with app key or device key (for configuration model). Determine if it need segmentation or reassembly.

To compliant with protocols that does not support long packet like BLE4.2, the maximum payload is set to 31byte.

3.1.5 Network layer

For transmit: contains sequence number for packet, encrypt data with network key and iv index. Sequence number will increase by 1 after transmission.

For reception: decrypt data with network key and iv index, then determine if sequence number is valid (if it is bigger than received value), waive if invalid.

3.1.6 Bearer layer

Send out encrypted packet to mesh network via LL_TYPE_ADV_NONCONN_IND(0x02).

3.2 Architectural concepts

3.2.1 States

Node states, e.g., on/off States, lightness States.

3.2.2 Bound states

2 bound states, like on/off and lightness. When on/off switch from 1 to 0, lightness will switch to 0, too; when on/off switch from 0 to 1, lightness will switch from 0 to the value before turn off. Similarly, when lightness switch from 0 to non-0 values, on/off value will switch from 0 to 1.

3.2.3 Messages

The encrypted packet sent to mesh network. Also called as mesh packet/mesh command.

3.2.4 Node & Elements

Node is a complete node or Bluetooth module, while element is an addressable entity within a node.

Node has only 1 address, while element can have 1 or multiple continuous addresses. The first element address is called primary address, which is the same with Node address.

Multiple element addresses are needed when a node has multiple same type states. E.g., a switch with 3 sockets need to control on/off state by Generic ON/OFF command, the task cannot be completed if there is only 1 address, in this case, and multiple element addresses are needed.

Although CT light has only one on/off states, it needs 2 generic level models, one is for lightness, the other is for temp; thus it needs 2 elements.

Similarly, HSL light needs 3 elements because it needs 3 generic level models for lightness, Hue and Sat, respectively.

When build network, node will report element number in provision flow interaction flow. E.g., if the number is 2, provisioner will assign an address to node, for example, 0x0002, node will then assign 0x0002 to element 1 and 0x0003 to element 2. Provisioner will assign from 0x0004 for the next node when provisioning.

3.2.5 Models

Please refer to 3.3.1 Model layer.

3.2.6 Publish & subscribe

- Publish：element send out status spontaneously, configure publish address and publish cycle parameter with command Config Model Publication Set. When publish address is configured, each time when status changes, node will execute publish status action spontaneously. Cycle publish parameter will determine whether it need to send it by a certain cycle.
- Subscribe: when a node receives published status message (e.g., generic on/off status) or control message (e.g. generic on/off), it will determine whether to handle this message based on the model's Subscribe list[].Subscribe list[] contains group address or virtual address, unicast address and 0xffff is invalid. Add new elements with Config Model Subscription Add, Config Model Subscription Virtual Address Add.

Determination rule

1) When received destination address is not unicast address, check if it can find a matching address in corresponding model's Subscribe list.
2) When destination address is unicast address, check if it matches its own element address.
3) When destination address is 0xffff, then means the message should be received/handled.

3.2.7 Security

Encrypt/decrypt need Network key, IV index, App key or device key.

A message need to be encrypted twice, encrypt the whole access layer(including op code，parameters) with app key or device key, and encrypt network layer with network key + iv index, network layer is the packet sent to mesh network, contains source address, destination address and sequence number.

When encrypt access layer, if the corresponding model of op code is config model, use device key, otherwise use app key.

For segment message, because the access layer was encrypted by the whole payload, so the decryption will be done only when all the segment packets are received.

SDK supports 2 network keys (NET_KEY_MAX) and 2 app keys (APP_KEY_MAX) by default.

Multiple network key manage multiple network.

Multiple app keys manages products of different security levels. For example, there are lights and lock in the same mesh network, and lock has higher security level, in this case, user can assign an independent app key to the lock, and the app key is only open to certain mobile app(provisioner), and will not share when sharing network, thus guarantee a higher security level.

3.2.8 Sequence Number Storage

As described in 3.1.5 Network layer, Sequence Number(SNO) of mesh message increases by 1 each time it sends out command, when reception determines SNO, it should be bigger than received value, otherwise the value is invalid. This requires store SNO to flash every time it sends out commands, which is too often. To avoid this, we defined the following: store SNO only when it increase by MESH_CMD_SNO_SAVE_DELTA(default value is 0x80). To guarantee SNO is bigger than

used value, the reading SNO will be added by MESH_CMD_SNO_SAVE_DELTA when boot up. Check MESH_CMD_SNO_SAVE_DELTA mesh_flash_save_check() and mesh_misc_retrieve for reference.

3.2.9 Friendship

Friendship is the relationship built between Friend Node (FN) and Low Power Node(LPN) according to prescribed establish friend ship flow. After friendship is established, when LPN sleep, FN will store any message sent to LPN by other nodes. When LPN wakes up, it will send POLL inquiry command to FN, and FN will answer with stored message. This method can lower power consumption, but it need friend node in the network, and will cause delay in command receiving and answering.

For a more details, please refer to the mesh spec and the subsequent LPN chapters.

3.2.10 Features

Mesh features:

- Relay feature: node will reduce message's TLL value by 1 after it receive the message, and then send out relay, when the received TLL value is less than or equal to 1, then no relay. With relay, the mesh network can obtain longer transmission distance. TLL is to control the delay time of the last node that receives the message. The TLL default value is TTL_DEFAULT(0x0A) in SDK, this macro is verify-able, provisioner can also configure this with Config Default TTL Set, maximum value is 127.
- Proxy feature: proxy is the protocol for mobile app connect to mesh network. In mesh network, app is an independent node, with its own Node address. Most app can not define transmitting packet discretionarily, neither monitoring mesh network all the time (switch to wifi for some time), so mobile app need to connect to a node with BLE GATT, the node will send out the data it receives from the app, and when it receives the answer message from the mesh network, it will sends back to mobile app with GATT according to proxy protocol.
- App sends message to node with ATT_OP_WRITE_CMD(0x52), node reply app with notify, i.e. ATT_OP_HANDLE_VALUE_NOTI(0x1B).
- Low Power feature: please refer to Friendship introduction in 3.2.9.
- Friend feature: please refer to Friendship introduction in 3.2.9.

3.2.11 Mesh Topology

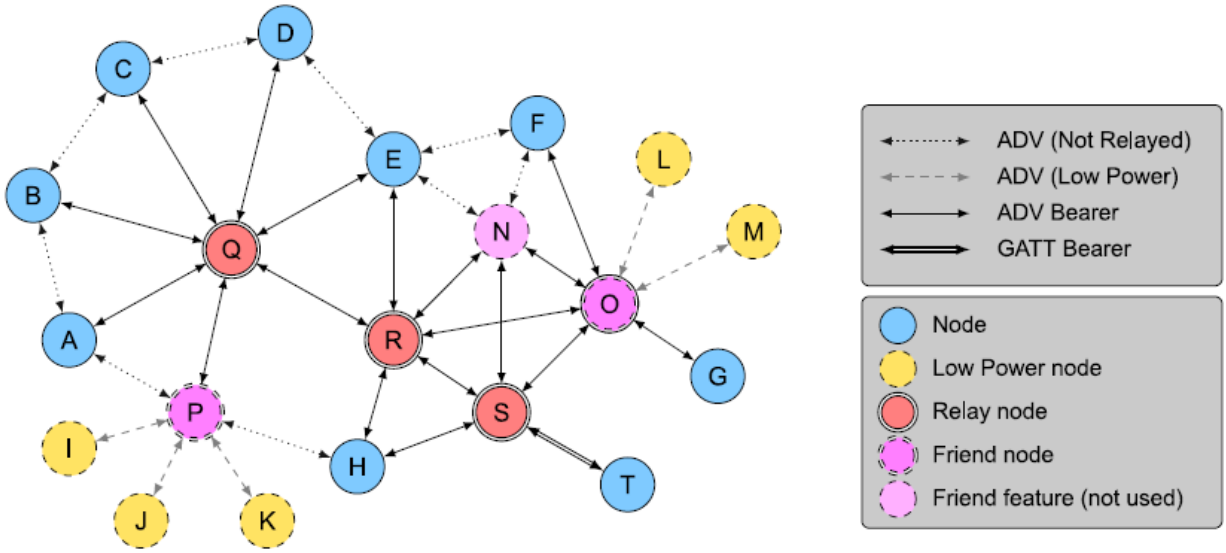


Figure 3.2: Mesh Topology

All nodes without low power support Relay, Friend. All nodes support ADV provisioning or GATT provisioning by default in this SDK.

3.3 Mesh networking

3.3.1 Network layer

Address:

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

Figure 3.3: 16 bit Address Allocation

- Unassigned address：0 for Unassigned address
- Unicast address for element address
- Group address：for group control and publish—-subscribe scheme.
- Virtual address：use together with 16BYTE label UUID, Virtual address is the value that calculate UUID with hash algorithm. When group address (total 16384) is not enough, this can be used to expand.

Network PDU:

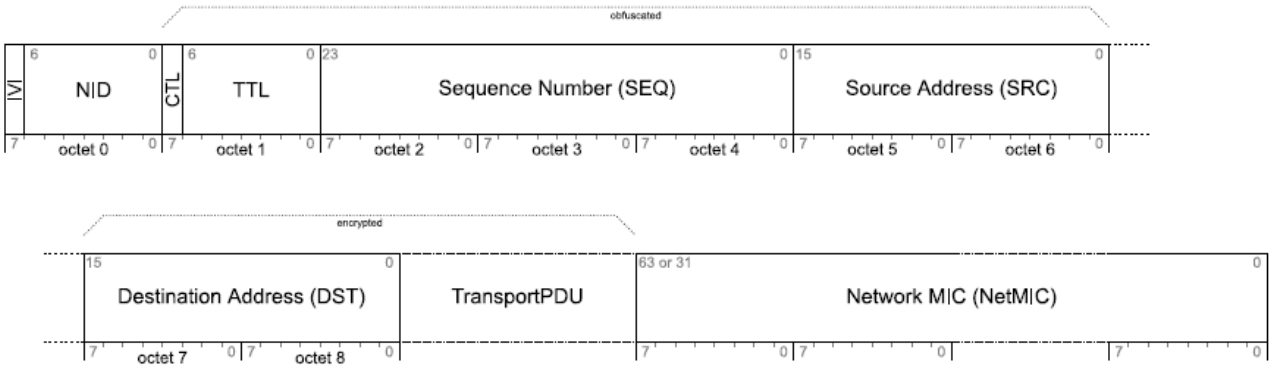


Figure 3.4: Network PDU Format

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

Figure 3.5: Network PDU Field Definitions

- IVI：iv index (i.e.,the lowest bit of iv_idx_st.tx[3], stored as big endian)
- NID：network key related
- CTL：flag for control message

Network transmit count/interval

Network transmit count is how many times it need to repeat to send out a command. The rf packet is exactly the same, including SNO. The purpose of send command repeatedly is to increase reception success rate. For example, for a 2-node mesh network, if the success rate of each transmit is 80%, that means the packet losing rate is 20%, so, theoretically, the packet losing rate is 6th power of 20%, which is 0.0064%, i.e., the success rate is 99.993%. This of course also depends on RF environment.

Our SDK's default re-transmit count is 5, i.e., TRANSMIT_CNT_DEF(5), the total transmit time = n+1 = 6.

Network transmit interval is the interval of 2 adjacent re-transmitted packets, the default value in SDK is 30-40ms, defined by TRANSMIT_INVL_STEPS_DEF, calculate in the following way: ((TRANSMIT_INVL_STEPS_DEF + 1)*10 + (0—-10))ms.

Network transmit count and transmit interval can also be configured by SIG config command CFG_NW_TRANSMIT_SET.

In conclusion, to send a network packet, e.g., generic ON/OFF no ack (this command need no de-packing), SDK need about 40 * 6 = 240ms.

Reliable retry

Reliable retry is the retry on app layer, used for commands with status answer, e.g., generic ON/OFF. When send out a network packet (including network transmit), the program will check if the status is received or not, if not, then it will retry, and the sequence number in the network packet will thus change. The program will retry twice at the maximum by default.

3.3.2 Access layer

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

Figure 3.6: Access Payload Field

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

Figure 3.7: Opcode Format

There are 3 kinds of op code, 1byte, 2byte and 3byte. 1byte and 2byte are defined by SIG, 3byte is defined by vendor, in which 2 bytes are vendor ID（CID）, a vendor id supports at most 64 vendor opcode in the whole mesh network.

Access layer contains op code and parameter, supports 380 byte at the maximum.

3.3.3 Transport layer

To compliant with protocols that does not support long packet like BLE4.2, the adv's maximum payload is set to 31byte. The effective payload of a single packet is 11bytes, others are occupied by communication protocols, so when Access layer is longer than 11byte, it need to be segmented, thus, for vendor op code, if the parameters is longer than 8 byte(8=11-3), the mesh protocol stack will segment the message spontaneously.

3.3.4 Mesh beacon

The following figure shows unprovisioned device beacon PDU.

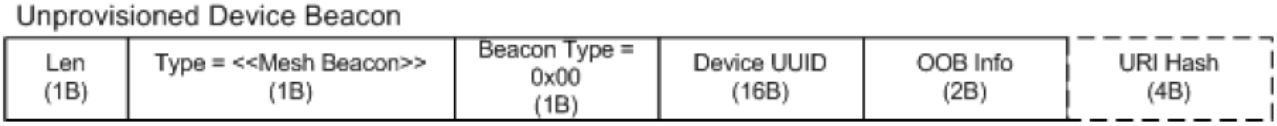


Figure 3.8: Unprovisioned device beacon PDU

Node can be identified by Device UUID. For some mobile phone, for example, IOS cannot get mac, nor can get mac in future remote provision, so in SIG mesh, node is identified by Device UUID instead of by mac.

Unprovisioned beacon is transmit via non-connectable ADV packet, used for PB-ADV provision mode.

Please refer to spec 3.9.2 Unprovisioned Device beacon for Oob info and URI Hash.

Before provision, unprovisioned beacon will be transmit via unprov_beacon_send(), the transmit interval is defined by beacon_send.inter = MAX_BEACON_SEND_INTERVAL, the default value is 2s.

After provision, security beacon will be sent out via mesh_tx_sec_nw_beacon(). User can also enable-disable this transmit command via CFG_BEACON_SET. Please refer to SecNwBc operation in 4.4, the transmit interval is defined by SEC_NW_BC_INV_DEF_100MS, the default value is 10s.

3.3.5 IV update flow

This is IV index update flow. Both network layer and access layer decryption/encryption need IV index. As described before, mesh network requires network PDU's sequence number accumulate all the time, and the length of sequence number is 3 bytes. So when sequence number approach its maximum, IV index needs to be updated, otherwise the sequence number will be reset to 0, and will be invalid in reception end. So IV index is the expand of sequence number.

Nodes will start and execute IV index update flow spontaneously. When a node is noted that its sequence number is bigger than IV_UPDATE_START_SNO (0xC00000), it will start IV update flow. The IV index increases by 1 for each IV update.

Check SPEC V1.0 3.10.5 IV Update procedure for details.

3.3.6 Heartbeat

Mesh heartbeat, sent out periodically, can be used for on/off line check (periodical publish can do the same thing), and hops calculation, i.e., to calculate how many times heartbeat message hopped before it get received.

Count received heartbeat, calculate the hops of each heartbeat, get the min hops and max hops, thus know the structure of the whole network and the message transmit ion reliability of each node. One heartbeat subscription configuration can only monitor/calculate 1 node.

hops is calculated in the following way:

hops = InitTTL - RxTTL +1

- InitTTL：TLL in heartbeat publish set
- RxTTL：TLL in received message network PDU

Nodes do not send heartbeat by default, check "Heartbeat demonstration" section for detailed configuration.

3.3.7 Health

Health model related message is used for node's warning/error status, e.g., battery warning/error messages. Check spec 4.2.15.1 Current Fault for detail, as shown in table below.

Table 3.1: Health model related messages

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

Value	Description
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 Debugging Tool Instructions

4.1 Download Firmware

Please refer to "Help" -> "User guide" for BDT tool detailed instruction. The following is instruction for frequent used operations.

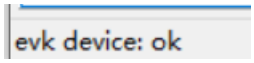
Before start, download 8258_mesh.bin into each node (8258 Dongle), then burn provisioner nodes, there are 2 provisioner modes, master dongle mode connected via GATT and gateway mode(ADV mode).

GATT mode: download 8269_mesh_master_dongle.bin in to 8269 Master Dongle (8269 Dongle).

Gateway mode: download 8258_mesh_gw.bin to 8258 gateway Dongle.

For example, user can download firmware to 8258 light node wit the following steps:

1) Hardware connection: connect miniUSB port on EVK board with PC USB port with USB cable, the light on EVK board will flash if the connection succeed. 8258 Dongle connect with EVK board USB port via USB port.



will show in the left lower corner of the tool.

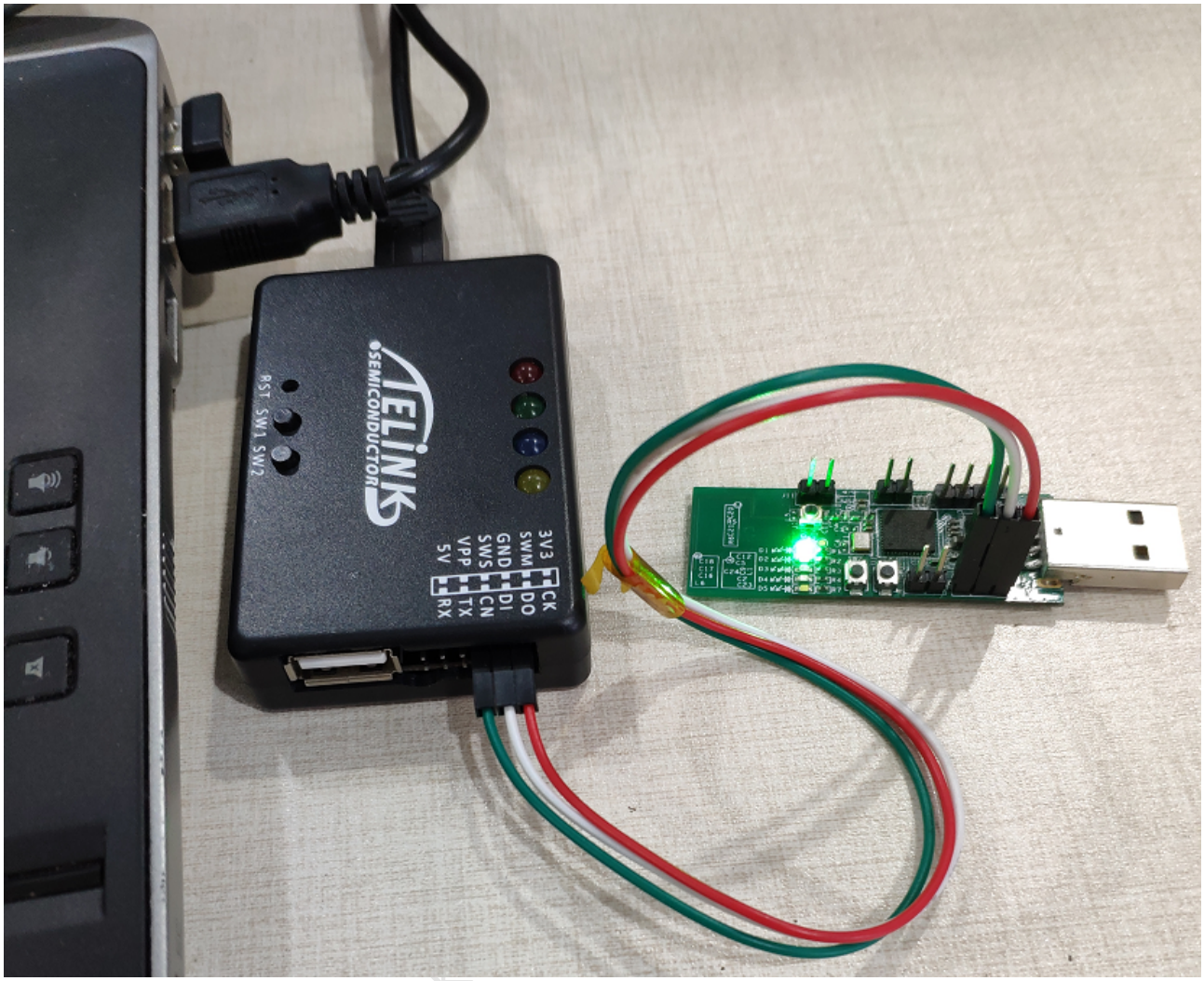


Figure 4.1: Hardware connection

2) Download 8258_mesh.bin to 8258 Dongle flash with Telink BDT tool.

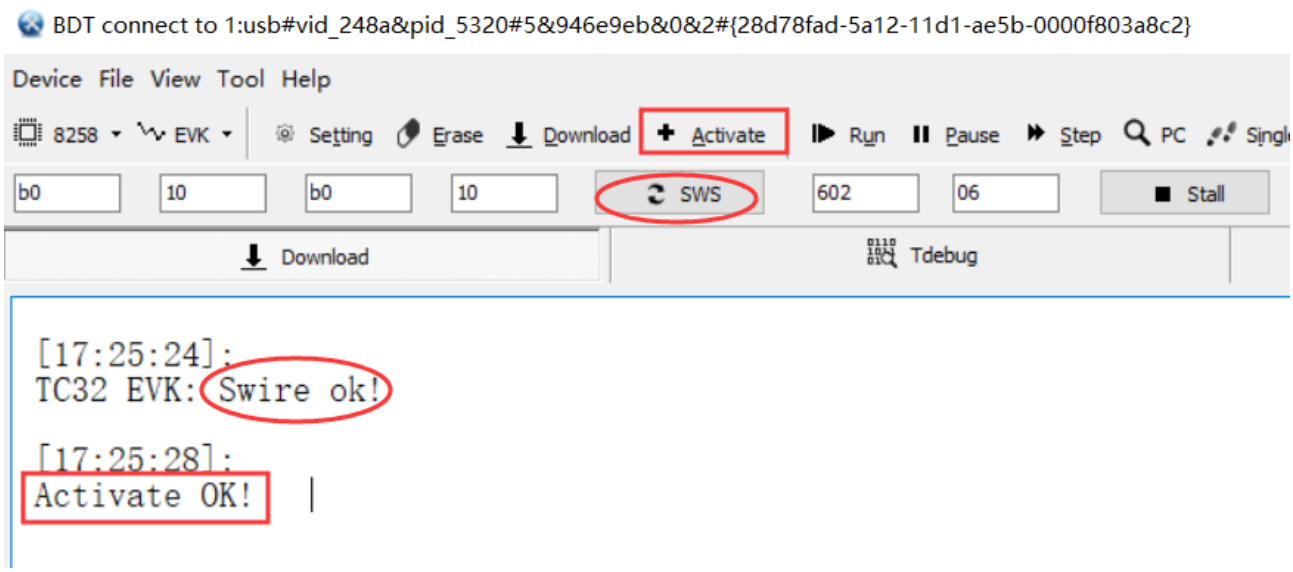
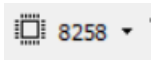
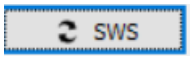


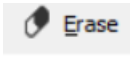
Figure 4.2: BDT interface

Step 1 Open BDT, click 

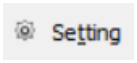
to choose the right part no, then click 

to check if EVK and 8258 dongle can communicate normally, if yes, Swire ok will show. If not, it may because the SoC is in sleep mode, click 

to awake the SoC, this is especially important for low power equipment, Activate OK will show when succeed. Active contains MCU restart.

Step 2 Click 

to erase 8258 Dongle flash.

Note: click 

to set start address and size for the erase action.

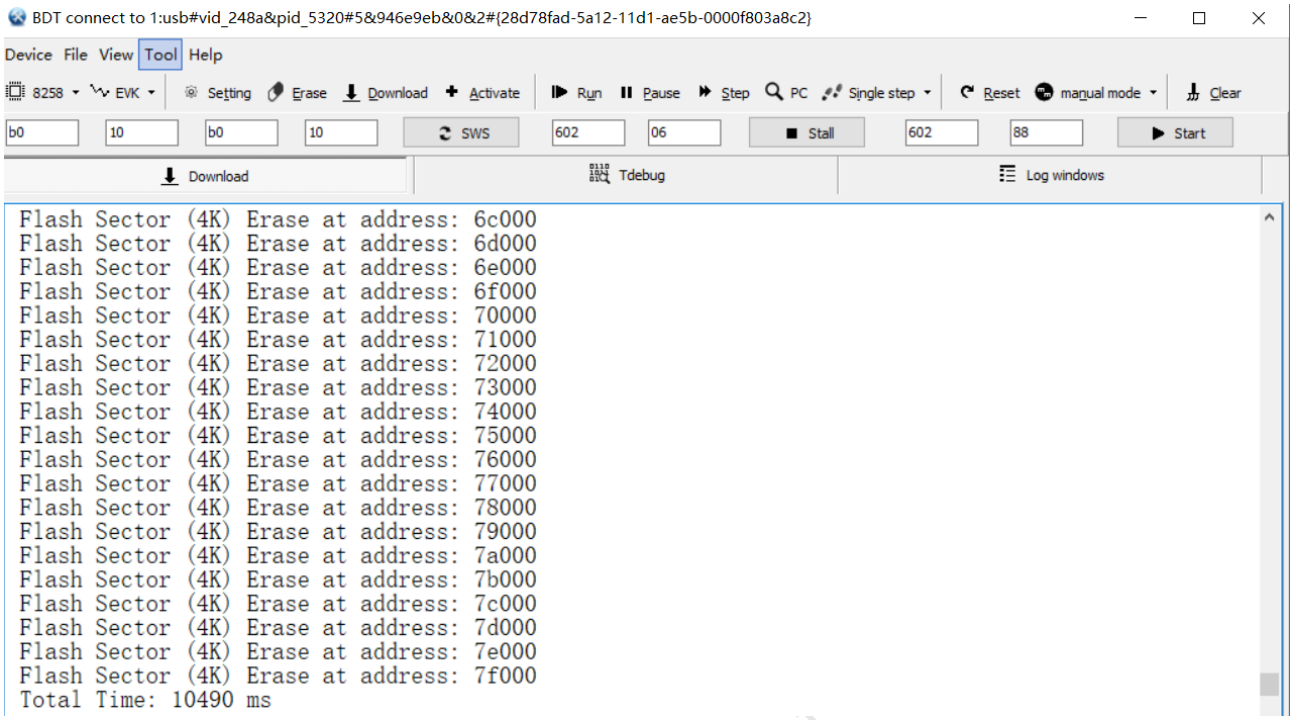
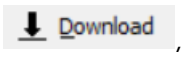


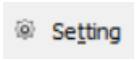
Figure 4.3: Erase Flash

Step 3 Click "File", then click"open", choose corresponding "8258_mesh.bin" file, click to open, then the BDT corresponding file:

Figure 4.4: Bin file

Step 4 Click 

, burn the chosen 8258_mesh.bin in flash address start from 0.

Note: click 

to set start address and size for the download action, default vaule is 0.

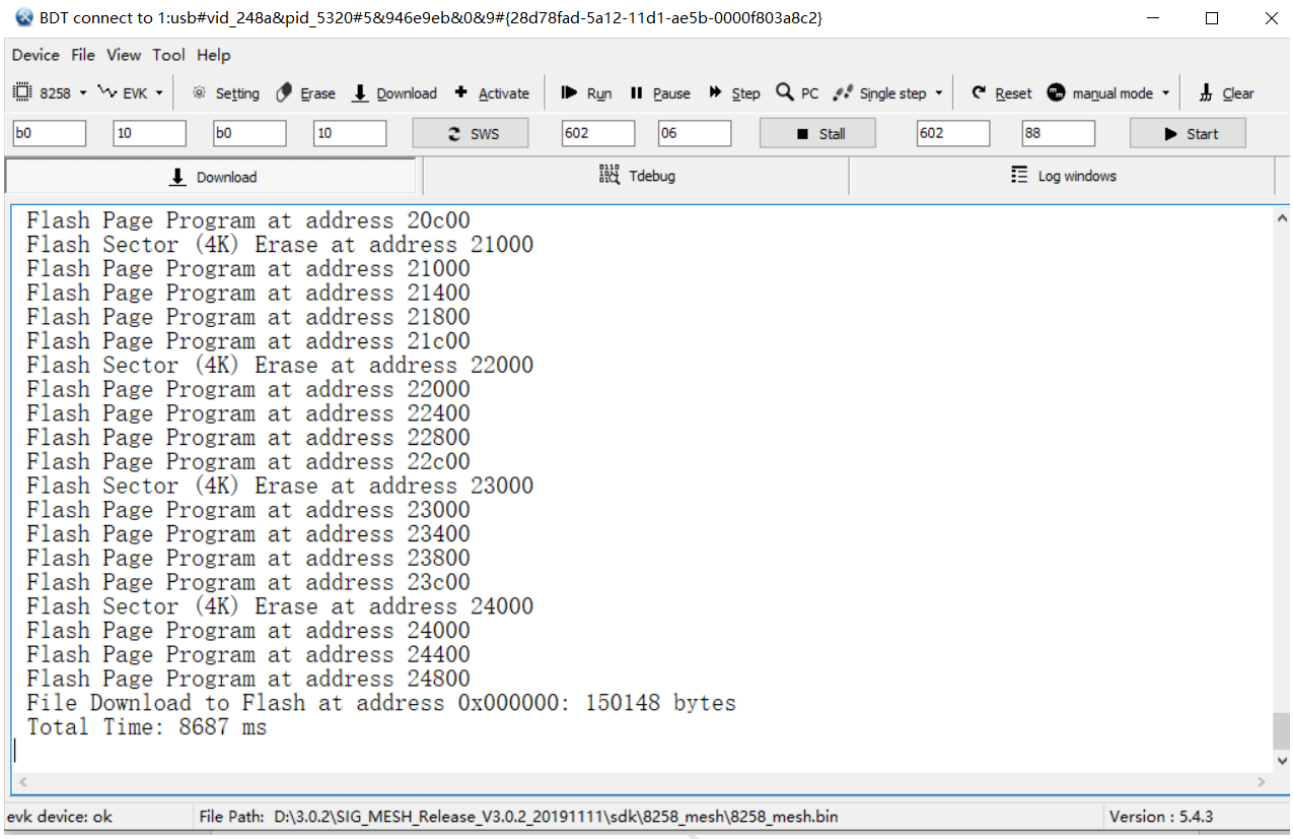


Figure 4.5: bin file burned into flash

Firmware Burning Steps

Step 1 Click "Tool"—"Memory Access", dialogue box pops up.

Note:

> This action only applicable when burning light nodes and gateway nodes, GATT master dongle does not need this.

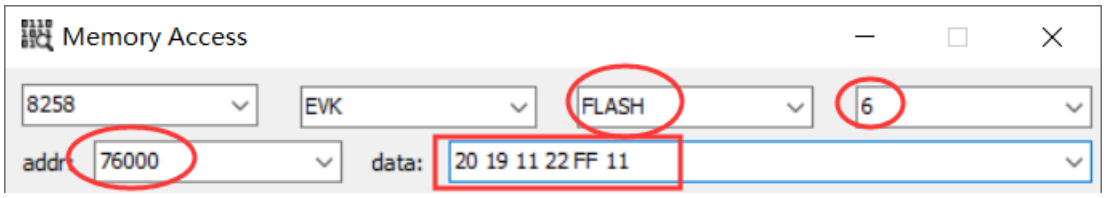


Figure 4.6: Input information

Input the 6-byte MAC as shown above, click enter in data field to write. Click Tab in Addr field to read, this is the read-back confirmation.

If the mac field keeps empty, when 8258 dongle reboot, it will detect 0x76000 has no mac, then it will assign a random mac and save it in 0x76000 in flash.

Step 2 After reboot, 8258 Dongle can work as a light node.

Above is the BDT frequent used operations. Users can also click "Help" -> "User guide" for details of other functions and operations.

4.2 BLE Connection and Adding Light in Gateway USB Mode

1) Open sig_mesh_tool.exe, plug the gateway dongle with burned 8258_mesh_gw.bin in PC USB port.
2) As shown below, "Found" means 8258 gateway Dongle connected correctly with PC tool, and the communication works. The tool will choose tl_node_gateway.ini automatically based on connected hardware.

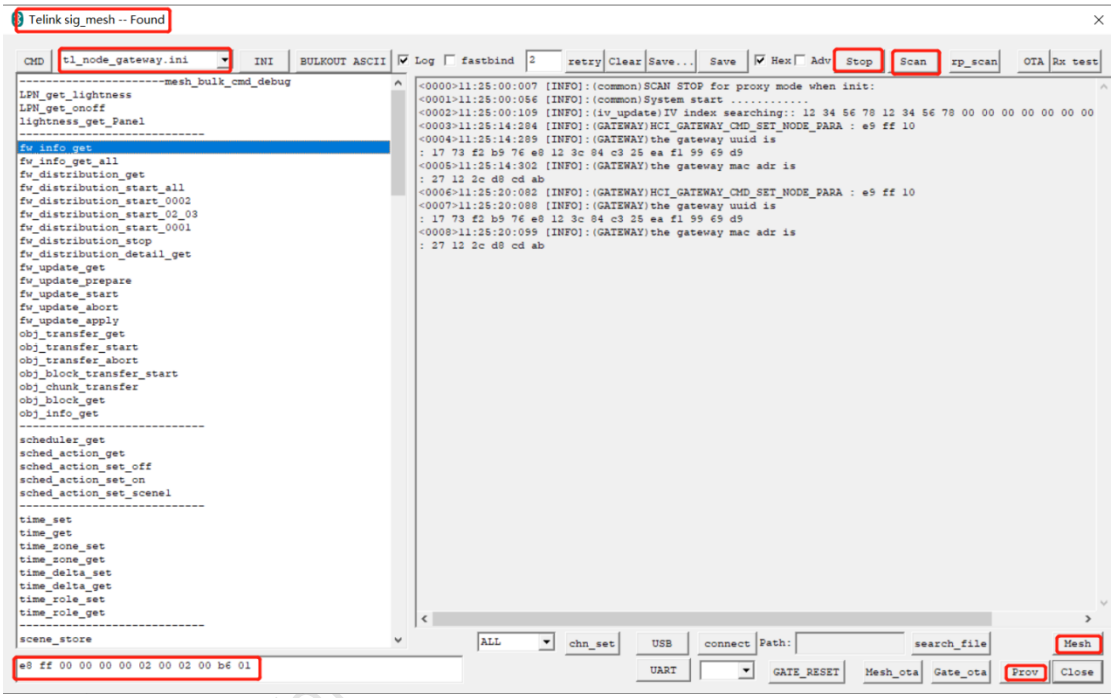


Figure 4.7: SIG_MESH_TOOL interface

3) Boot 8258mesh node.
4) Click "Scan" to open "ScanDev" window, showing corresponding mac address, including rssi and frequency offset.

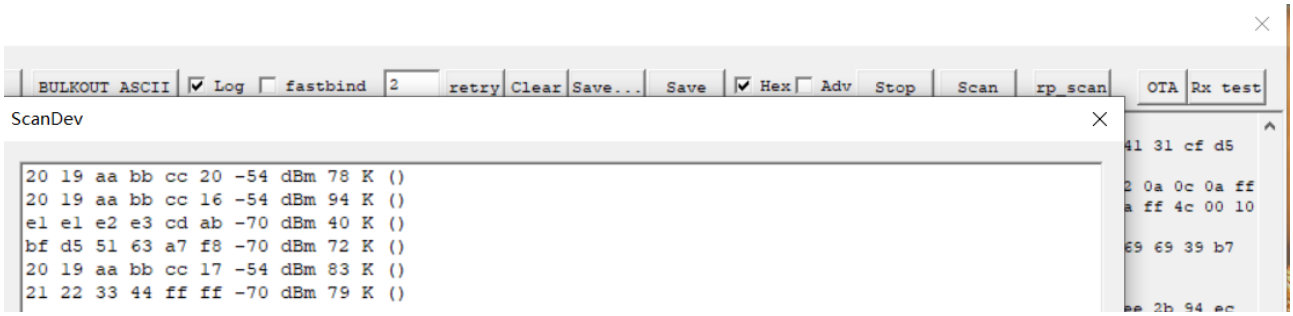


Figure 4.8: ScanDev window

5) Click corresponding item in "ScanDev" to choose node.

Gateway mode: double click to choose the node, no connection or command transmission.

8258 gateway Dongle mode: double click will build BLE connection, if the red light on 8258 gateway Dongle turns on, means BLE connection is built successfully, "Stop" is to stop current BLE connection, when the white light on 8258 gateway Dongle turns on, means BLE connection is stopped. Currently supports only single node BLE GATT connection.

6) Click "Prov" to open "provision" window.

Note:

> "Provision" and "bind_all" is forbidden after initialization, users can not use the 2 button at the same time.

The "network_key" is generated randomly when first open "provision", it can be modified before click "SetPro_internal".

Figure 4.9: provision window

7) Click "SetPro_internal" to set network initial parameters, print "Set internal provision success" in log window to show that the parameters are set successfully.

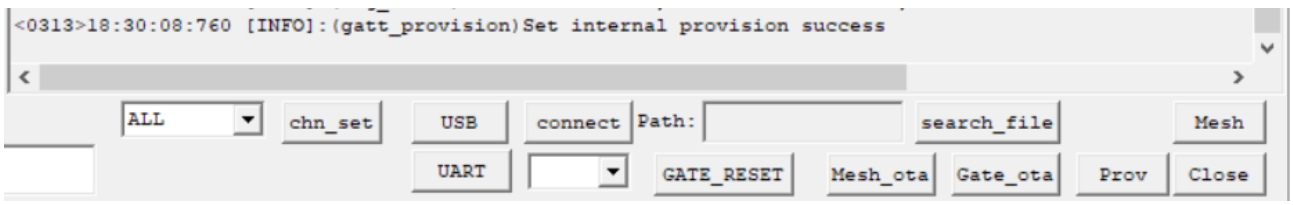


Figure 4.10: Set internal provision success window

Once click "SetPro_internal", the corresponding parameters like netkey cannot be modified, so "SetPro_internal" will turn to grey. Parameters will be saved in mesh_database.json and will be read automatically next time open the tool. If network_key need to be modified, then the whole network should be dismissed, and reset to Factory settings.

Now "Provision" is enabled. unicast_addr is the primary address to be assigned to provision, user can change it manually, but it highly recommended not to.

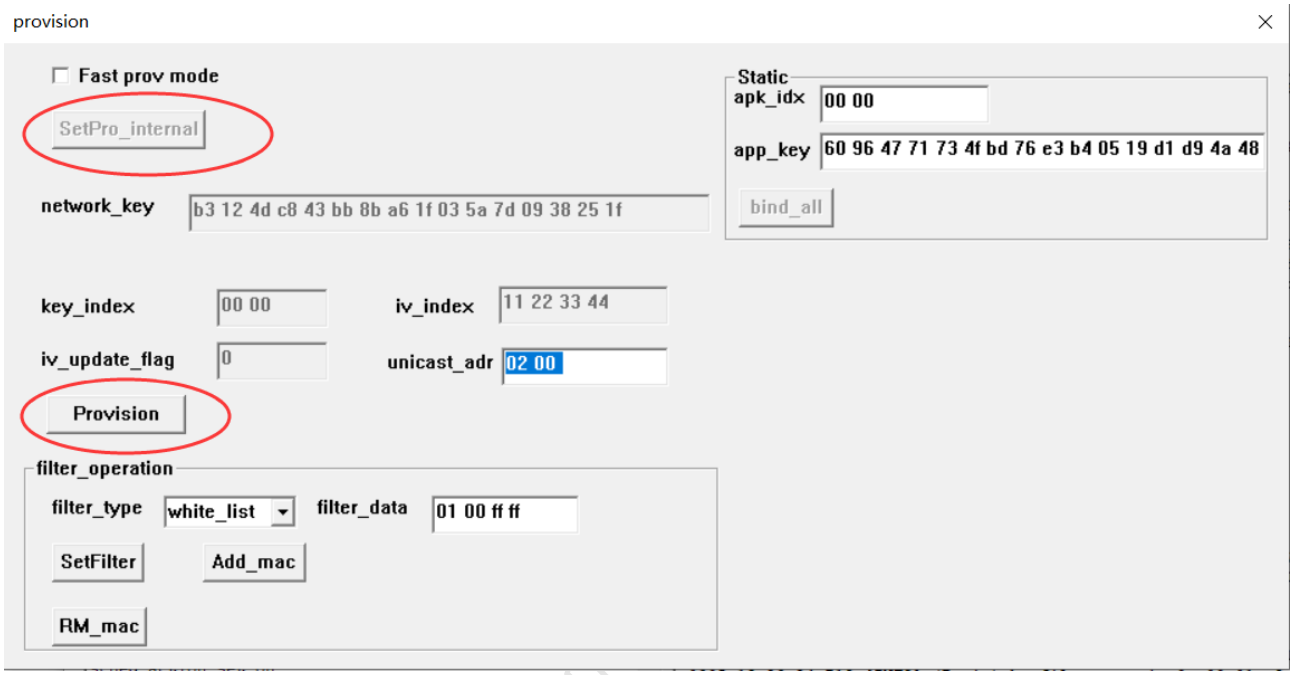


Figure 4.11: Provision enabled

8) Click "Provision" to execute SIG provision flow to add corresponding node into network. The red LED will flash 4 times to show the connection success. Log information is shown below:

Gateway mode log:

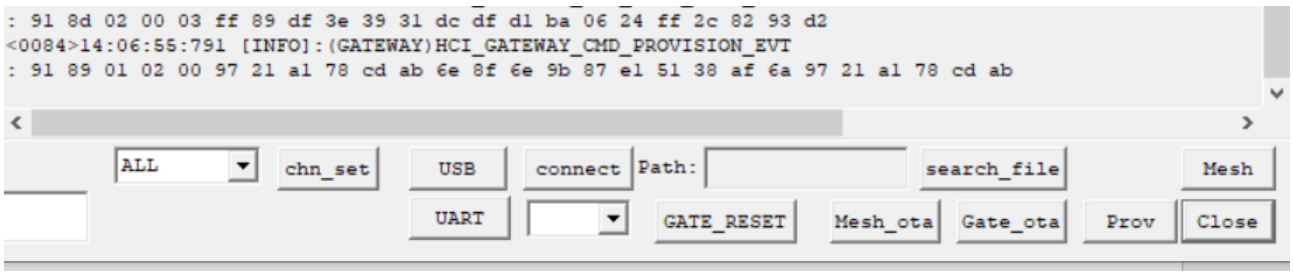


Figure 4.12: Gateway mode log

GATT Master dongle log:

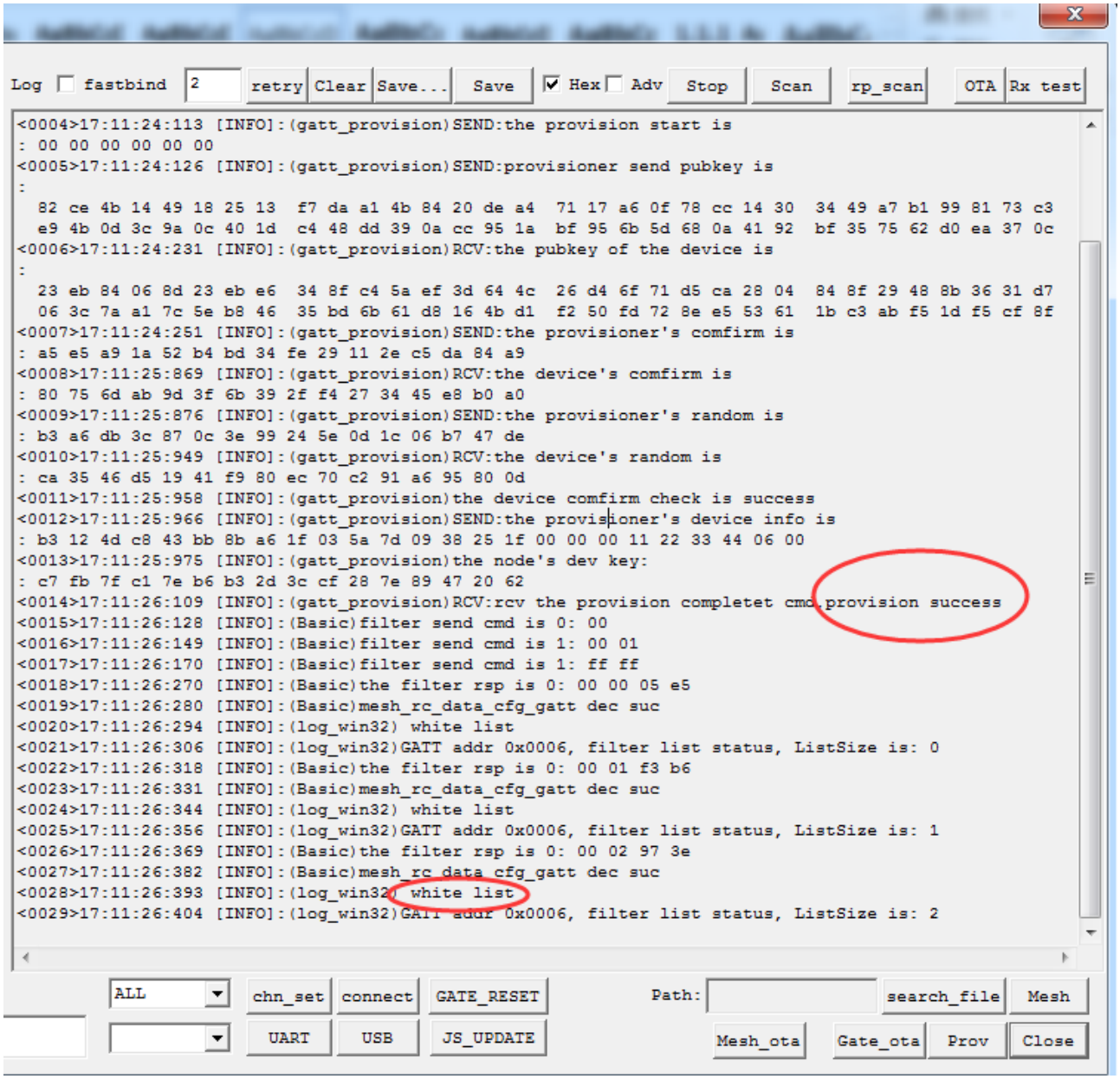


Figure 4.13: GATT Master dongle log

9) Click "bind_all" after configure app_key, first get composition data will be sent to get all model ids, then bind app_key to all models.

After Bind_all, unicast_adr will automatically accumulate based on the elements number the current node contains, and calculate primary address for next node provision.(e.g., CT light has 2 elements, then unicast_adr will increase by 2 each time a CT light is added).

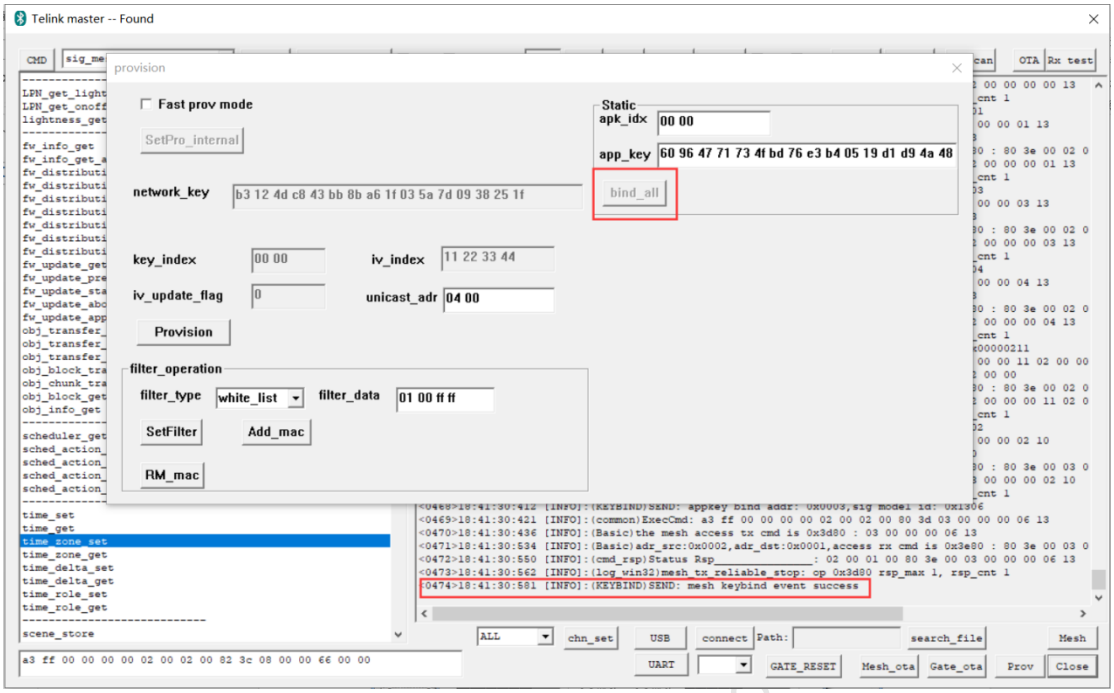


Figure 4.14: Click bind_all

10) After binding App_key, click mesh to enter mesh UI, user can turn on/off light here.

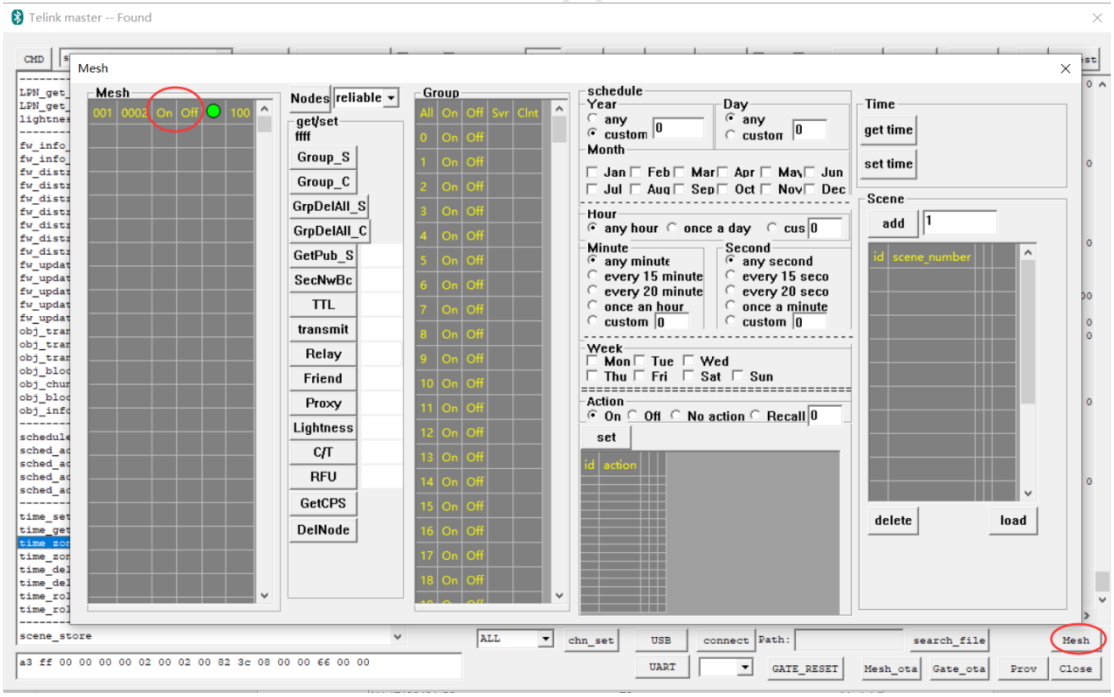


Figure 4.15: mesh UI

11) Dismiss network

Both GATT master dongle and Gateway mode can follow the following way:

Choose a node, then click "DelNode" to delete this node. Refer to 4.5.3 "DelNode" instruction for detail.

Note:

> In GATT master dongle mode, please delete no-GATT-direct-connected nodes first, then delete GATT direct connected nodes. Delete GATT directly connected node will disconnect current GATT.

4.3 BLE Connection and Adding Light in Gateway UART Mode

Configure UART ports:

1) Choose HCI_USE_UART in HCI_ACCESS in gateway firmware, re-compile.
2) Insert port tool to PC, connect tx/rx with gateway rx/tx.
3) Open "sig_mesh_tool.exe".
4) Click UART then choose the pop-up COM port.
5) Click "Connect", if the connection succeed, the button will change to Disconnect. Now UART can execute gateway functions.
6) All other operations are similar with that of USB mode, please refer to section 4.2 for detail.

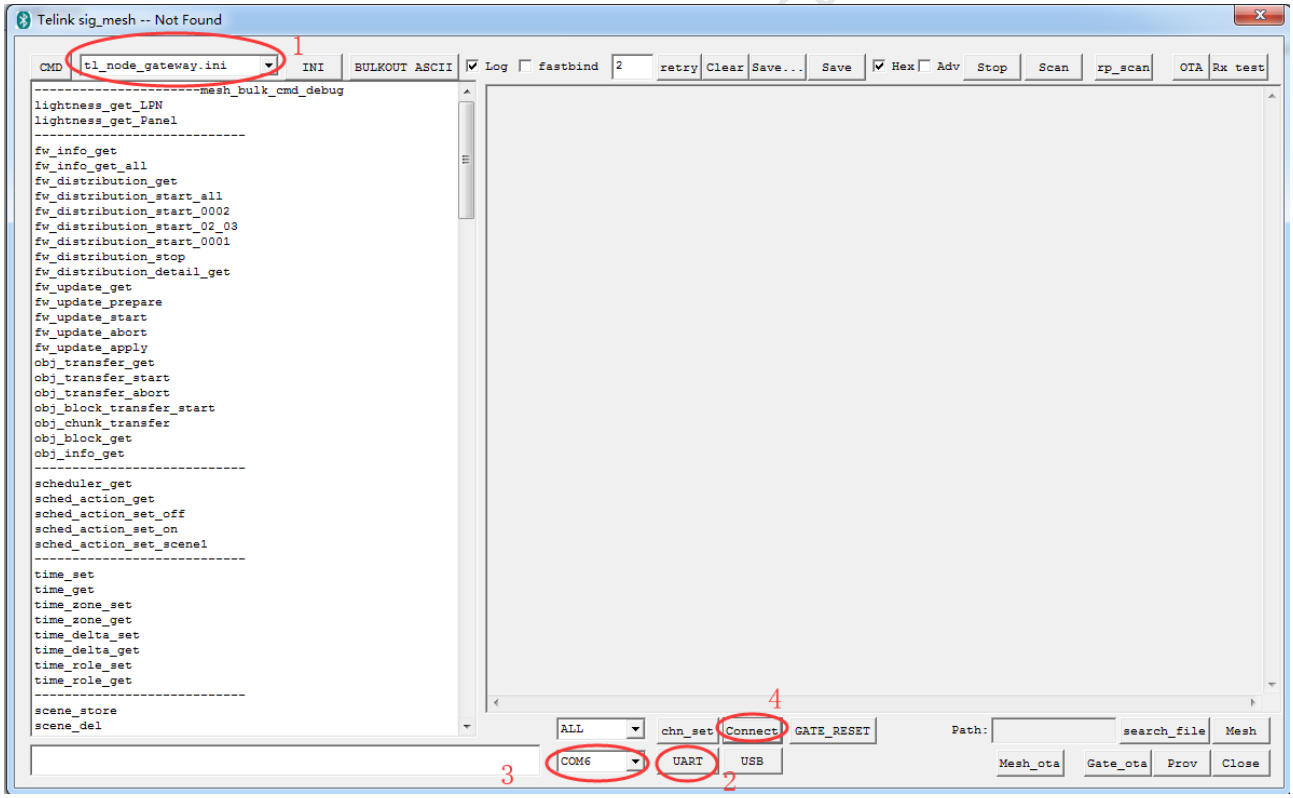


Figure 4.16: Configure UART port

4.4 BLE Connection and Adding Light in GATT master dongle Mode

1) Open sig_mesh_tool.exe", plug 8269 Master Dongle with burned program in PC USB port.

2) As shown below, "Found" means8269 Master Dongle connected correctly with PC tool, and the communication works. The tool will choose sig_mesh_master.ini automatically based on connected hardware.

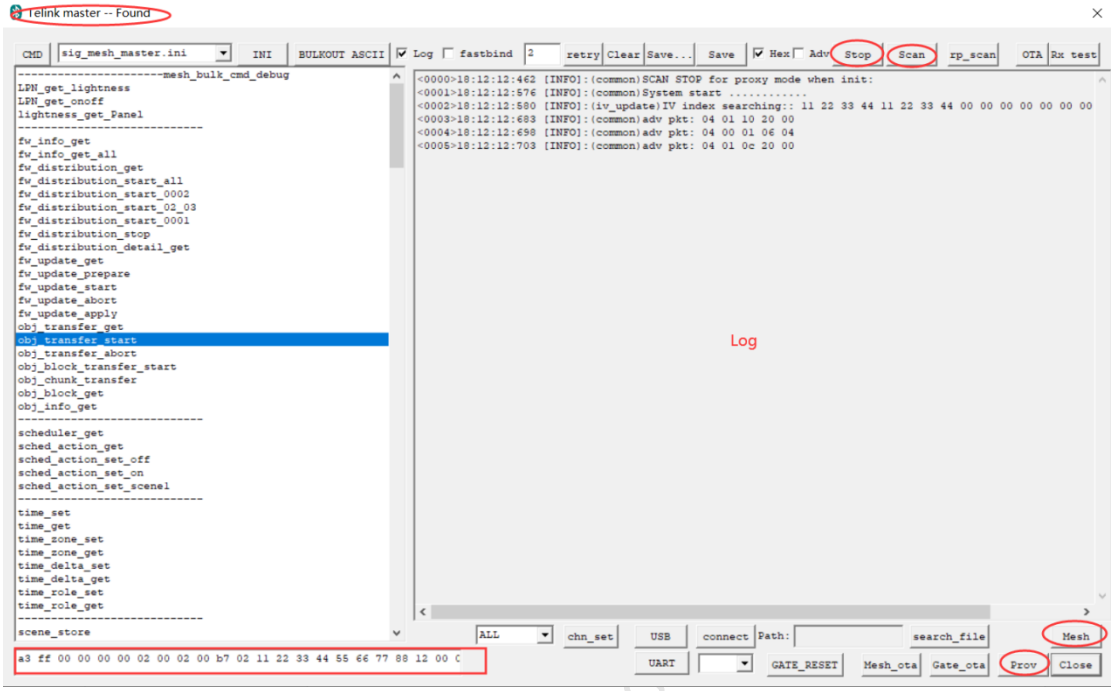


Figure 4.17: SIG_MESH_TOOL interface

3) Please refer to section 4.2 for further steps.

4.5 Control Corresponding Nodes

GATT master dongle and gateway have the same operation and UI.

4.5.1 UI Display and on/off Control of Single/All Node(s)

1) Click "Mesh". A "Mesh" window will pop up.

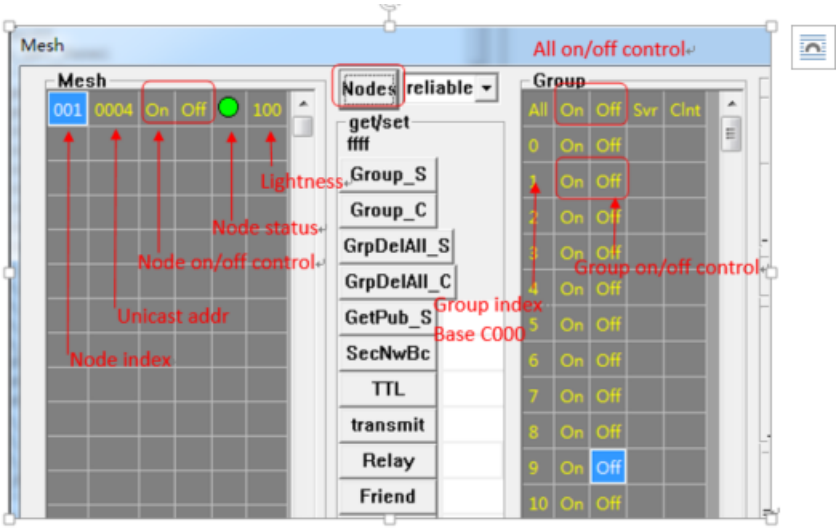


Figure 4.18: mesh window

2) By clicking "Nodes" in "Mesh" window, user can refresh all light status.

If click "Nodes" when choose reliable in the right drop-down box, it will send out lightness get command. The UI will be refreshed according to lightness status.

If click "Nodes" when choose unreliable in the right drop-down box, it will send out lightness get command. But on/off command is no ack. Node will not reply status in this case, so the UI will not be refreshed, use publish to refresh UI.

If click "Nodes" when choose online status in the right drop-down box, it will not send out command, only initialize UI to null, then refresh UI according to returned online status data.

Note: online status is a private mode, the node's firmware should enable ONLINE_STATUS_EN.

The lightness display in 0-100 scale, which is switched from SIG defined 0-65536 scale.

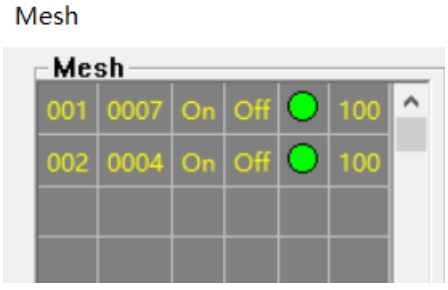


Figure 4.19: Node status

3) Single node operation: click "On"/"Off", the corresponding light will control the switch status. The node status will be reported to the tool to refresh the corresponding status.

4)  : "on",  : "off",  : off-line

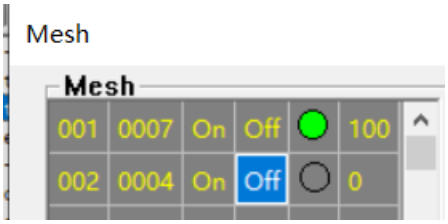


Figure 4.20: Single node control

5) All on all off control: Click "On"/"Off" besides "All", all nodes in the mesh network will switch to on/off.

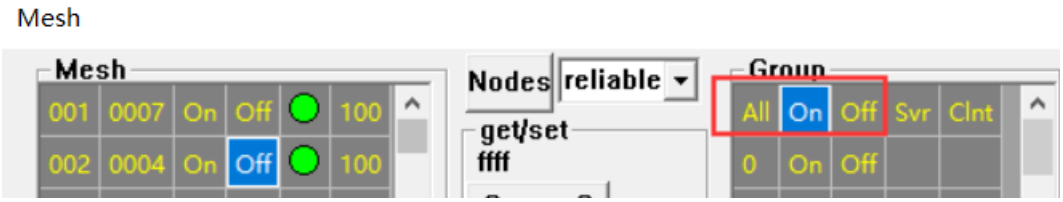


Figure 4.21: All node control

4.5.2 Group Control (Subscription Demo)

Click index number of the light node, e.g., 002, to get the node address and show in position in below figure. The default value is 0xffff, means no node is chosen.

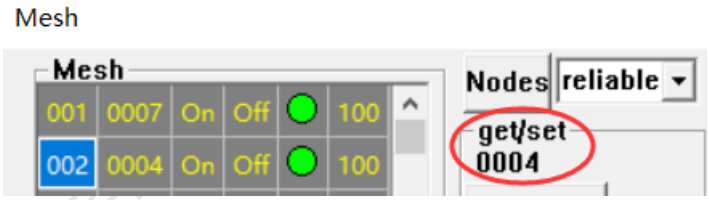


Figure 4.22: Obtain node address

User can click/right click "Svr" box in Group Control, to add/delete this light node in/from corresponding group. The √ in "Svr" means the node has been added to the group, the blank in "Svr" means the node is not in the group.

- "Svr" column is for generic on/off server model (0x1000)
- "Clnt" column is for generic on/off client model (0x1001)

Normally, node supports only server model, so only "Svr" column is operated.

Group index and group address's relation is described as: group address = group index + 0xC000.

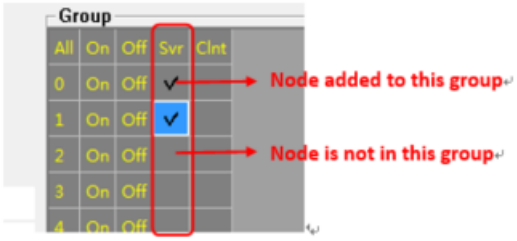


Figure 4.23: Allocate one light to multiple groups

User can click the corresponding "On"/"Off" to control the group in Group controls.

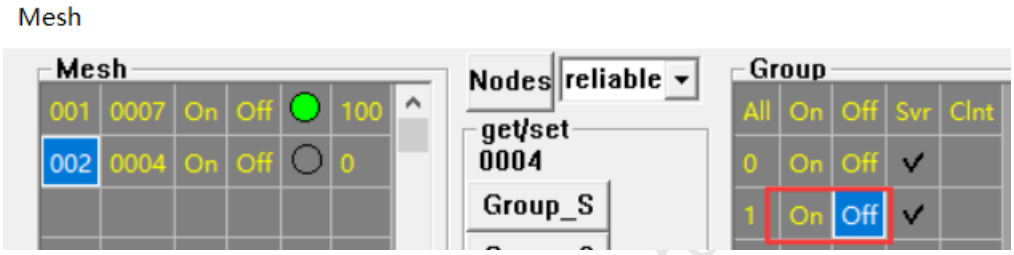


Figure 4.24: Group control

4.5.3 Configure Node Parameter with UI

As shown in figure below, click position 1 to choose the node, then the tool will send get group (subscription list) command out automatically to get the node's group and show it in corresponding UI. Then determine if current node supports scene, time, scheduler function, if yes, send get commands out automatically to get scene, time and scheduler list.

Send SCENE_REG_GET to get all valid scene index, and display in UI list.

Send TIME_GET to get time of current node, and display in UI. If the node is just booted, the time will be 0, which means the node is waiting for configuration, in this case, the time will keep 0, and do not do the timing action. User can configure time in 2 ways, 1, send time set command via app/gateway, 2, when the node boots, configure its time model's publish character to get time status other nodes published.

Send SCHD_GET to get all valid scheduler index, then based on the returned index value, send SCHD_ACTION_GET respectively to get detail parameters, and display in UI list. As shown in below figure, the provision is just completed, no scheduler is added, so no need to send SCHD_ACTION_GET.

For the same reason, group, scene, time and scheduler are all blank.

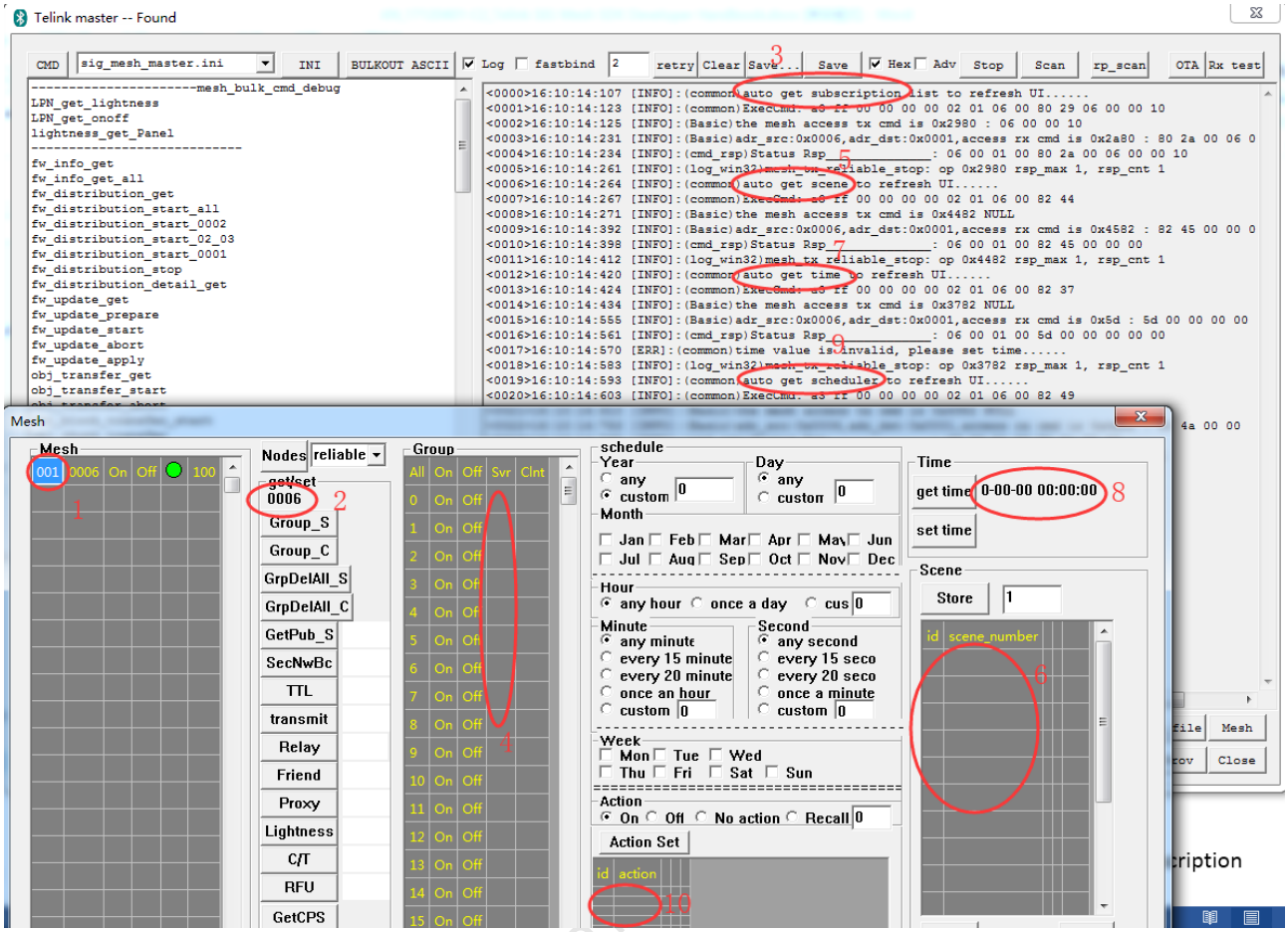


Figure 4.25: Configure node parameter with UI

The following button is based on current chosen node.

"Group_S"

By clicking this button, CFG_SIG_MODEL_SUB_GET will be sent to get subscription address list of on/off server model of current node, and display it in "Svr" column.

"Group_C"

By clicking this button, CFG_SIG_MODEL_SUB_GET will be sent to get subscription address list of on/off client model of current node, and display it in "Clnt" column.

Most nodes do not support on/off client model, so this is not a frequent used button.

"GrpDelAll_S"

By clicking this button, CFG_MODEL_SUB_DEL_ALL will be sent to delete subscription address list of on/off server model of current node, and clear "Svr" column.

"GrpDelAll_C"

By clicking this button, CFG_MODEL_SUB_DEL_ALL will be sent to delete subscription address list of on/off client model of current node, and clear "Clnt" column.

"GetPub_S"

By clicking this button, CFG_MODEL_PUB_GET will be sent to get publish address of on/off server model of current node, and display the return value in later display box.

Modify publish address in input box, then press "Enter", CFG_MODEL_PUB_SET will be sent, and the corresponding publish address is the value in input box.

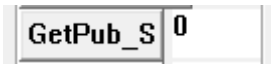


Figure 4.26: "GetPub_S"

Other parameters are default values. Check command sending log:

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

Figure 4.27: Command sending log

Other publish parameters can be modified by cfg_pub_set_sig of INI command, modify the parameter to wanted value, then send.

"SecNwBc"

By clicking this button, CFG_BEACON_GET will be sent, the return value will display in right display box. This command determine whether to send security network beacon or not.

Modify value in input box, then press "Enter", CFG_BEACON_SET will be sent, and the corresponding parameter value is the value in the input box.

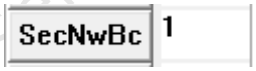


Figure 4.28: "SecNwBc"

"TTL"

By clicking this button, CFG_DEFAULT_TTL_GET will be sent, the return value will display in right display box. This command gets the default TTL value of the node. SDK default value is defined by TTL_DEFAULT.

Modify TLL value in input box, then press "Enter", CFG_BEACON_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.29: "TTL"

"transmit"

By clicking this button, CFG_NW_TRANSMIT_GET will be sent, the return value will display in right display box. This command gets the network transmit value of the node. The lower 3bit is network transmit count, the higher 5bit is network transmit interval.

SDK default values are defines as following:

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

Figure 4.30: SDK default value

Note:

> transmit count(5) + network transmit interval(2) is 0x15.

Modify network transmit value in input box, then press "Enter", CFG_NW_TRANSMIT_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.31: "transmit"

"Relay"

By clicking this button, CFG_RELAY_GET will be sent, the return value will display in right display box. This command gets the relay enable value of the node.

Modify Relay value in input box, then press "Enter", CFG_RELAY_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.32: "Relay"

"Friend"

By clicking this button, CFG_FRIEND_GET will be sent, the return value will display in right display box. This command gets the friend feature enable value of the node.

Modify Friend value in input box, then press "Enter", CFG_FRIEND_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.33: "Friend"

"Proxy"

By clicking this button, CFG_GATT_PROXY_GET will be sent, the return value will display in right display box. This command gets the proxy feature enable value of the node.

Modify Proxy value in input box, then press "Enter", CFG_GATT_PROXY_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.34: "Proxy"

"Lightness"

By clicking this button, LIGHTNESS_GET will be sent, the return value will first switch from 0-65535 scale to 0-0x64 and then display in right display box.

Modify Lightness value in input box, then press "Enter", LIGHTNESS_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.35: "Lightness"

"C/T"

By clicking this button, LIGHT_CTL_TEMP_GET will be sent, the return value will first switch from 800-20000 scale to 0-0x64 and then display in right display box.

Modify C/T value in input box, then press "Enter", LIGHT_CTL_TEMP_SET will be sent, and the corresponding parameter value is the value in the input box.

Figure 4.36: "C/T"

"RFU":

Reserve for future.

"GetCPS"

By clicking this button, COMPOSITION_DATA_GET will be sent, the return value will display in right display box.

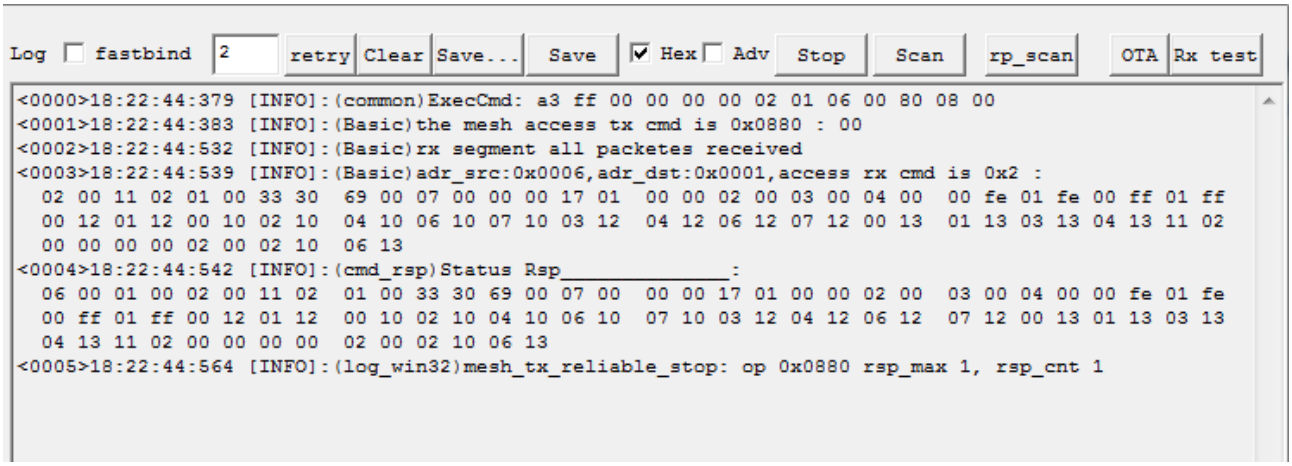


Figure 4.37: Return value

"DelNode":

By clicking this button, NODE_RESET will be sent to delete current node from the network. If the deletion succeed, the red LED light of the node will flash for 8 times, and the node will run reboot operation.

4.6 Time model operation

1) The node's time model is disabled by default in Firmware, MD_TIME_EN needs to be set to 1. Gateway 8269 is disabled by default, need to be enabled, gateway 8258 is enabled by default. After setting MD_TIME_EN to 1 for the node and gateway, compile and burn, and regroup the network.
2) Double click to choose the node.

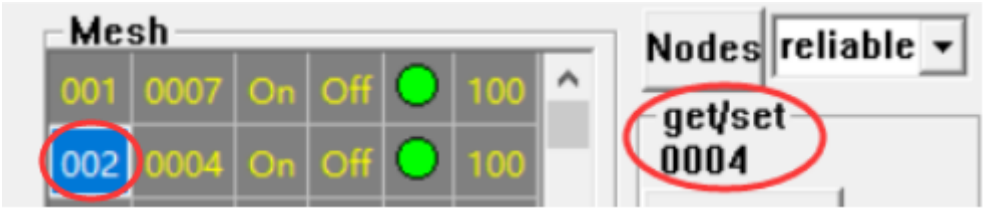


Figure 4.38: Double click to choose the node

3) Click "set time", the tool will send current time of the PC to the node via "TIME_SET". Time will display as following, and will refresh automatically.

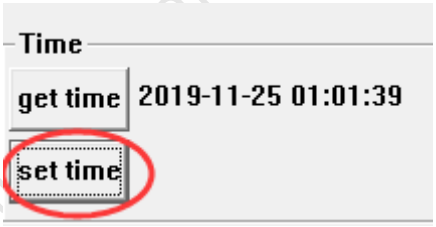


Figure 4.39: "set time"

Note: the parameters of time set when sending is shown in detail below:

```
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;
```

Figure 4.40: time set parameters

- TAI_sec: the value compares with time zone 0.
- zone_offset: set current time zone, unit is 15 minutes.

e.g.: Beijing time2019/1/1 09:00:00(UTC+8) configuration:

```
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);
}
```

Figure 4.41: Switch between TAI and local time

The above function is to show how to switch local time to TAI, but normally it not in this way. Time set is set from Mobile APP or PC, and both have API to get current TAI_sec and zone_offset, e.g., PC firmware operate in the following way:

(OFFSET_1970_2000 is because PC's base time is 1970 while SIG MESH's base time is 2000)

```
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);
    }
}
```

Figure 4.42: PC firmware operate

Note:

When the node is powered off, the time will be lost, i.e. g_TAI_sec is equal to 0, so it will not start timing. You need to receive the timeset command from app or gateway, etc., or receive TIME_STATUS information from other nodes that have not been powered off to publish. after that the clock will work properly. How to configure node publish TIME_STATUS message: When Telink SIG mesh app is networking, it will check if there is a time model inside the model list of the composition data, and if yes, it will automatically send the publish command to the time model. If it is a gateway, or a master dongle, etc., you need to send the command manually.

4.7 Scene model operation

1) The node's scene model is disabled by default in Firmware, MD_SCENE_EN need to be set to 1. Gateway 8269 is disabled by default, need to be enabled, gateway 8258 is enabled by default.
2) Double click to choose the node.
3) Set the node's status to the scene wanted one via UI or INI. E.g., generic on/off set and lightness set.
4) Input scene number, then click "Store", and scene adding command (SCENE_STORE) will be sent, to set node's current status to corresponding scene ID, and list all the configured scene ID in the list, as shown in figure below.

The processing function after the node receives the SCENE_STORE is: mesh_cmd_sig_scene_set().

Note:

The SCENE_STORE have only scene number, no light status information. Node will automatically save currently status information like on/off, lightness as scene status when receive scene adding command.

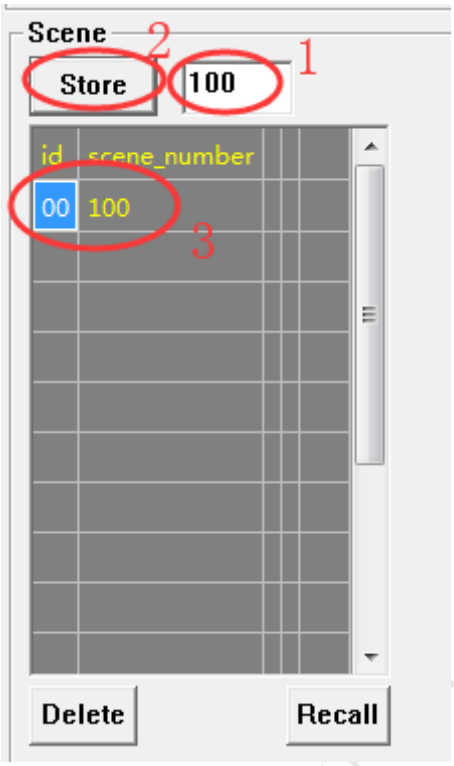


Figure 4.43: Input scene number

5) Recall scene, i.e., set light status to status defined by scene. Click buttons in the following figure, then click "Recall".

The processing function after the node receives a SCENE_RECALL is: mesh_cmd_sig_scene_recall().

Note:

> Recall scene will change light status, but it is not reported because publish status is not configured, to refresh UI, configure publish parameter in corresponding model.

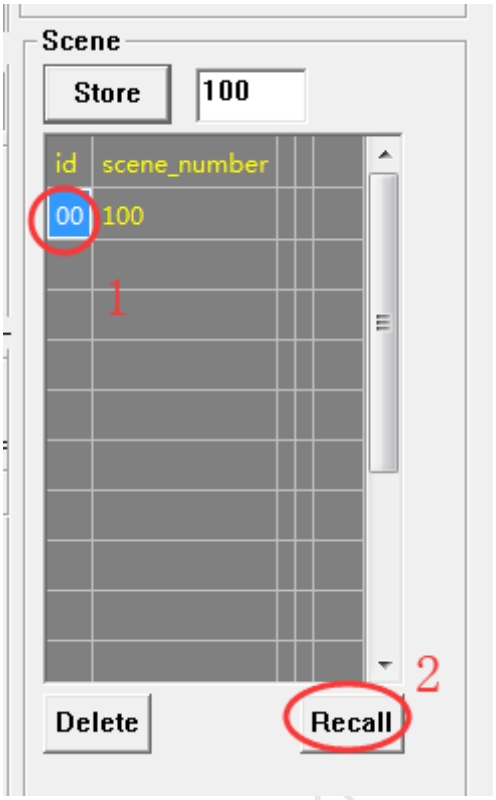


Figure 4.44: Recall scene

6) Modify scene. No modify command, modify with scene store.

7) Delete scene. Click buttons in the following picture, then press "Delete".

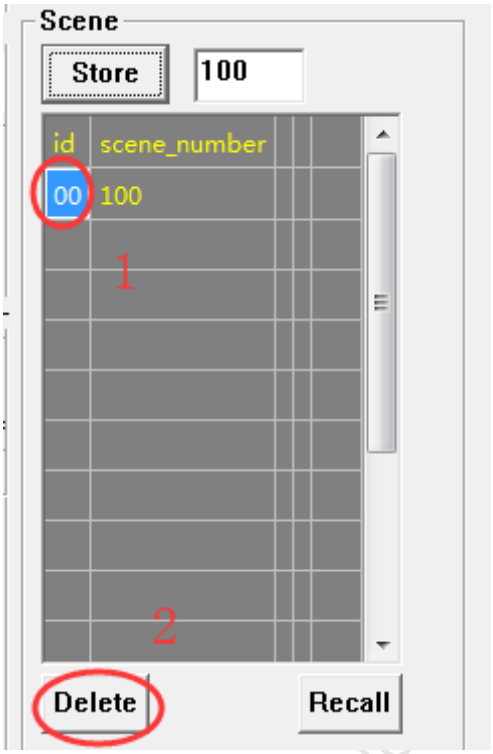


Figure 4.45: Delete scene

4.8 Scheduler model operation

1) Parameters (refer spec for details)

- Year：any：each year, custom：a specific year，base：year 2000, ie, 0 means year 2000，19 means year 2019
- Month：can choose 1/multiple/all. Note：both blank and all means choose all
- Day：any：each day, custom：a specific day
- Week：can choose 1/multiple/all. Note：both blank and all means choose all
- Hour：any：each hour, once a day：randomly respond once a day, and the random number is generated daily; custom：a specific year，
- Min：any：each minute, every 15 means responds on 0/15/30/45, every 20 means responds on 0/20/40, once an hour：randomly respond once an hour, and the random number is generated hourly, custom：a specific minute
- Second：similar with Min.

2) Double click to choose the node. If the action column is blank, that means the ID's schedule of the node is not configured yet.

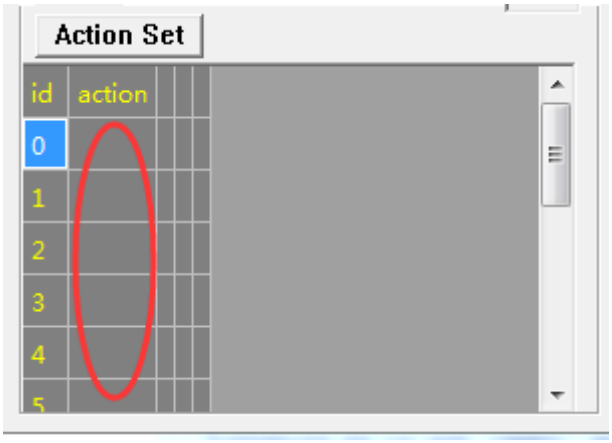


Figure 4.46: Action Set

3) Click ID column, choose the to-be configured scheduler's ID (maximum 16 by definition in SIG, range from 0-15), the chosen ID will show in blue background, and the schedule parameter of the ID will be refreshed to the UI above.

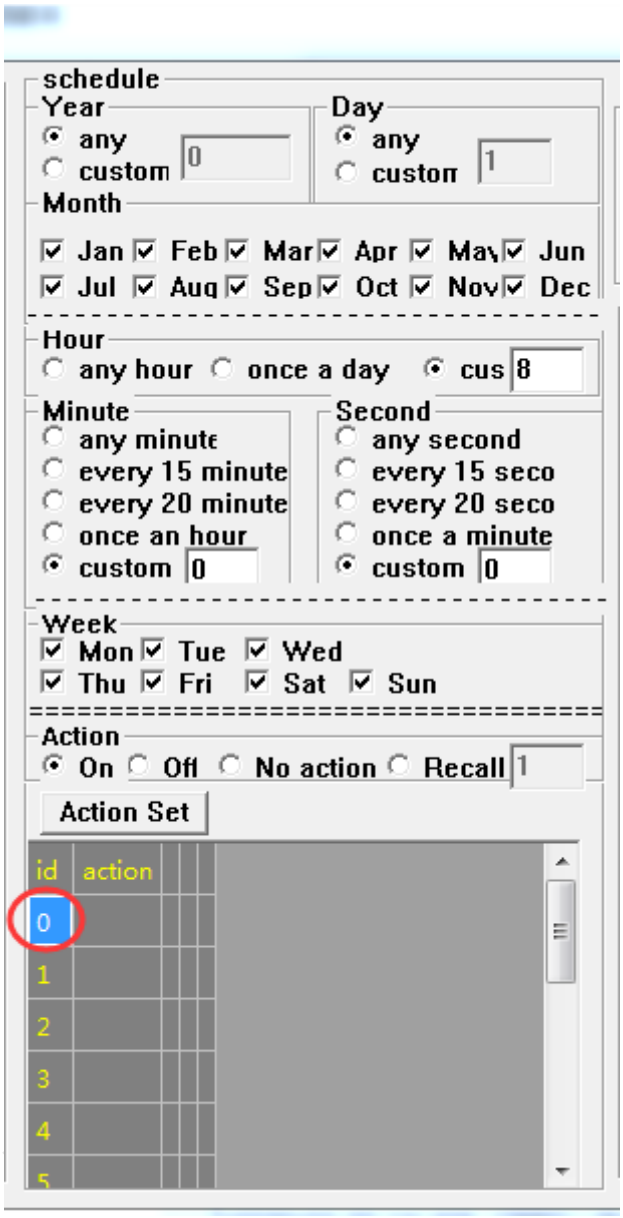


Figure 4.47: Click "id"

4) User can modify schedule parameter, then click "Action Set" to send SCHD_ACTION_SET to configure.

Because of the UI display limit, only action parameters are shown in the list, i.e., "on", "off", "no action", "recall", if the field is blank, that means the schedule is not configured yet.

Click a specific value in ID column to check detail information of a schedule id.

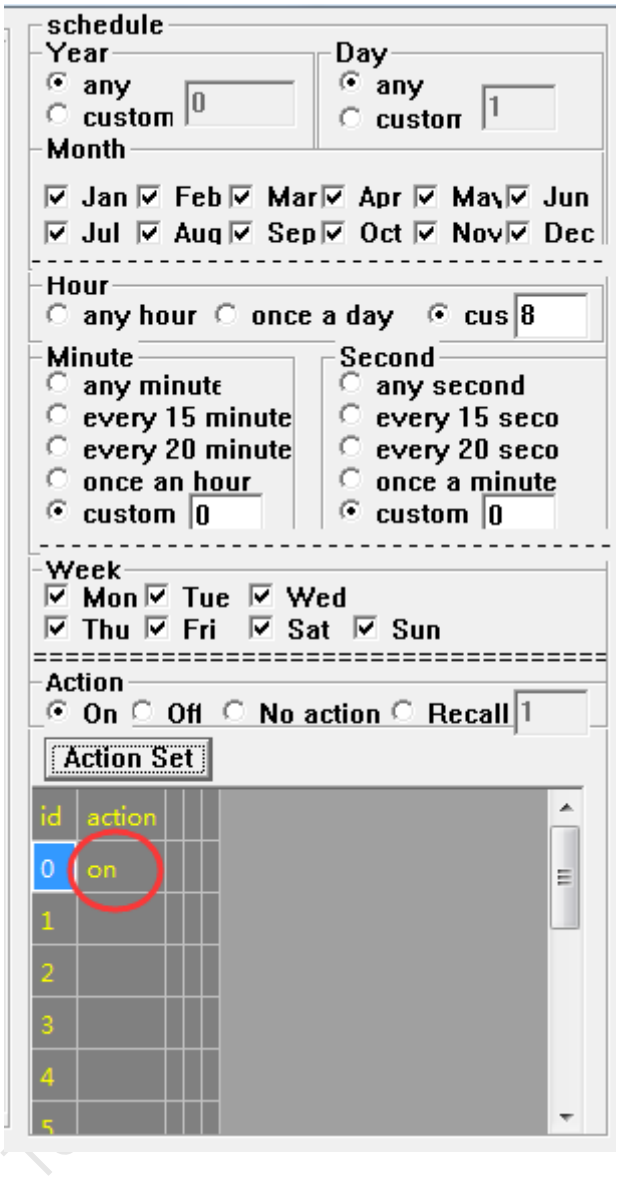


Figure 4.48: schedule parameter

5) Delete Schedule

No specific delete command in SIG. Set action of the schedule to "No action" to delete the schedule.

5 Factory Test Mode

5.1 Purpose

Factory test mode is used to manufacture, to execute some common control tests without provision, e.g., on/off, lightness control, CT control and etc. Gateway and GATT master dongle support this mode while APP does not support for now.

5.2 Factory Test Mode Parameters

- unicast address: it is the lower 15bit of MAC by default, if the lower 15bit is 0, then take 1 as unicast address.
- The network key, app key, device key，IV index use the compiled default value.

5.3 Default Test-able Commands

The control-able models are defined by factory_test_model_array[], while the useable commands of configure model are defined by factory_test_cfg_op_array[].

```
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};
```

Figure 5.1: Default testable commands

Controls under factory mode do not need provision, please refer to section 4.5. Please be noted, the mode needs all nodes unprovisioned, including gateway and master dongle.

6 Important SDK Modules

6.1 Configure Mesh SDK Default Feature

1) Mesh nodes describe their supporting features in composition data (model_sig_cfg_s_cps.page0.head.feature), SDK initialization is shown as below:

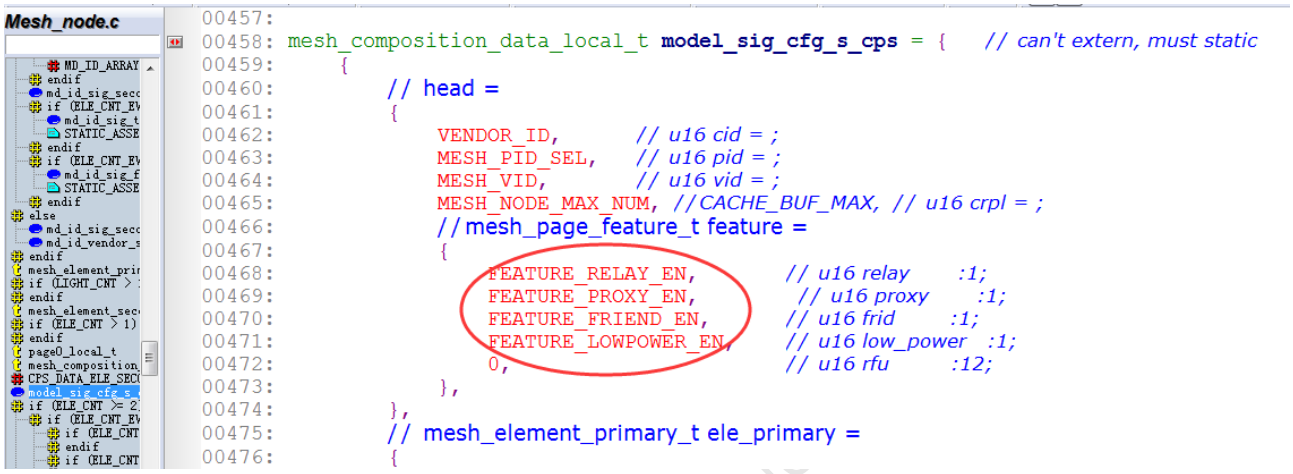


Figure 6.1: SDK initialization

2) Composition data defines supports or not, enable/disable can also be defined under "support" status. Please refer to the configuration action model_sig_cfg_s.frid in mesh_global_var_init().

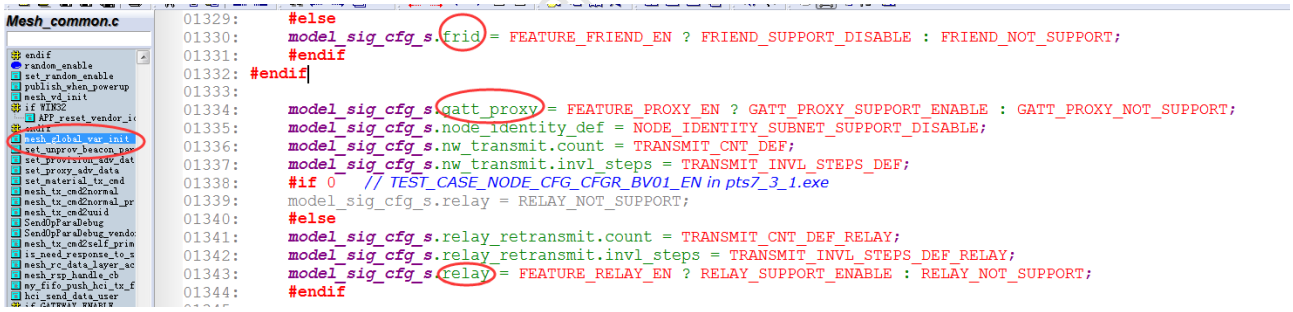


Figure 6.2: Enable/disable the configuration

During working procedure, user can enable/ disable these features with the following commands: CFG_FRIEND_SET, CFG_RELAY_SET and CFG_GATT_PROXY_SET.

During working procedure, user can enable/ disable these features with the following commands: CFG_FRIEND_SET, CFG_RELAY_SET and CFG_GATT_PROXY_SET.

3) For default features of each compiling project, please refer to Demo Project in SDK Instruction.

6.2 Common Macro Definitions

Some common macros and APIs can also be found in "common api".

6.2.1 LIGHT_CNT and ELE_CNT_EVERY_LIGHT

See "Definition of the number of elements of a node" for an introduction.

6.2.2 ONPOWER_UP_SELECT

ONPOWER_UP_SELECT defines the state setting of the lamp when the lamp node is powered off and powered back on, which can be set to one of the following states:

- ONPOWER_UP_OFF: After powering up, the lamp is set to the OFF state.
- ONPOWER_UP_DEFAULT: After powering up, the lamp is set to the ON state.
- ONPOWER_UP_STORE: After powering up, the light stays in the same state as it was before the power was off.

See chapter "3.1.4 Generic OnPowerUp" in the mesh model spec "MshMDL_v1.1.pdf" for details:

3.1.4 Generic OnPowerUp

The Generic OnPowerUp state is an enumeration representing the behavior of an element when powered up. The values for the state are defined in Table 3.3.

Value	Description
0x00	Off. After being powered up, the element is in an off state.
0x01	Default. After being powered up, the element is in an On state and uses default state values.
0x02	Restore. If a transition was in progress when powered down, the element restores the target state when powered up. Otherwise the element restores the state it was in when powered down.
0x03–0xFF	Prohibited

Table 3.3: Generic OnPowerUp states

Figure 6.3: OnPowerUpType.png

6.2.3 MESH_POWERUP_BASE_TIME

It is used to define how long after a node has been powered up, and then after a random time, it starts sending lightness status or onoff status to notify the gateway or cell phone, etc. that the node is currently online.

See mesh_vd_init() and system_time_run() for details:

```
void mesh_vd_init()
{
......
    publish_powerup_random_ms = rand() % 1500;  // 0--1500ms
    STATIC_ASSERT(MESH_POWERUP_BASE_TIME >=200);
    publish_powerup_random_ms += MESH_POWERUP_BASE_TIME; // 200ms: base time.
```

```
......
}

void system_time_run(){
......
        if(publish_powerup_random_ms && clock_time_exceed_ms(0, publish_powerup_random_ms)){
            publish_powerup_random_ms = 0;
            publish_when_powerup();
        }
......
}
```

6.2.4 Checking Whether a Node has been Provisioned

is_provision_success();

6.3 Definition of the Number of Elements of a Node

The number of elements a node contains can be one or more. It is determined by the values of these two macros: ELE_CNT_EVERY_LIGHT and LIGHT_CNT to determine that each element occupies a unicast address.

```
#define ELE_CNT                         (LIGHT_CNT * ELE_CNT_EVERY_LIGHT)
```

- ELE_CNT_EVERY_LIGHT means a product unit consists of several elements, for example, a color temperature lamp consists of two elements. The reason why we need two elements is that the color temperature lamp has two states, brightness value and color temperature value. The brightness can be controlled by commands such as level set, and the color temperature can also be controlled by commands such as level set. If there is only one element, when the color temperature node receives the level set command, there is no way to distinguish whether to control the brightness or the color temperature, so it needs two elements. Similarly, HSL (RGB) light needs three elements.
- LIGHT_CNT：Indicates that a BLE mesh module has several product units.

For products supporting server model, for example, when one BLE module drives two color temperature lamps, LIGHT_CNT needs to be set to 2.

For products that support the client model, such as remote control products, such as Switch project, the destination address of control commands sent by keys can be modified by modifying the publish address. Hence, for the keys with the same command, the number of keys need to be independently configured with a publish address needs to be consistent with the number of keys need to be configured with LIGHT_CNT. For example, the Switch project of demo SDK has 4 pairs of keys that send group address, so LIGHT_CNT is set to 4.

6.4 Grouping Features and Share-model

For a description of the spec counterpart, please refer to "4.2.4 Subscription List" in MshMDL_v1.1.pdf, among others.

(1) SIG Mesh spec defines that we can configure the group number independently for each model, so currently each model has a copy of the independent group number data, the maximum number of storage is 8 (SUB_LIST_MAX), as shown in the following figure.

```
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;
```

Figure 6.4: Configure group index

(2) The commands for models that use device key encryption and decryption do not support multicast addresses because the device key is different for each node. The models that use device key are config model and so on, see MODEL_ID_DEV_KEY[] for details:

```
const u32 MODEL_ID_DEV_KEY[] = {
    SIG_MD_CFG_SERVER,              SIG_MD_CFG_CLIENT,
    SIG_MD_REMOTE_PROV_SERVER,      SIG_MD_REMOTE_PROV_CLIENT, // no para
    SIG_MD_DF_CFG_S,                SIG_MD_DF_CFG_C,
    SIG_MD_BRIDGE_CFG_SERVER,       SIG_MD_BRIDGE_CFG_CLIENT,
    SIG_MD_PRIVATE_BEACON_SERVER,   SIG_MD_PRIVATE_BEACON_CLIENT,
    SIG_MD_SAR_CFG_S,               SIG_MD_SAR_CFG_C,   // save in model_sig_cfg_s_t now.
    SIG_MD_ON_DEMAND_PROXY_S,       SIG_MD_ON_DEMAND_PROXY_C, // save in model_sig_cfg_s_t now.
    SIG_MD_LARGE_CPS_S,             SIG_MD_LARGE_CPS_C, // no para to save
};
```

(3) The sig mesh spec also stipulates that within the same element, those with state binding relationship or model extension relationship should share the group number information. For example, after configuring a group number for the onoff model, the lightness model will be automatically bound to this group number. So SUBSCRIPTION_SHARE_EN needs to be turned on by default. For details, please refer to "Summary of xxx models" in the model spec "MshMDL_v1.1.pdf", e.g., "6.7 Summary of lighting models" in the Figure 6.12: Relationships between lighting models - Part 1". You can also check the sub_share_model_sig_onoff_server_extend[] in the SDK, which contains all the models that have extensions to the onoff model.

(4) Some models have no state binding relationship with each other, for example, onoff model, vendor model, sensor model, you need to send the command to configure the group number for these three models. In some applications, there are some proprietary requirements, i.e., when binding the group number to a model, you want to automatically bind the group number to the models that do not have extended model or state binding relationship, so as to reduce the time of group number configuration. At this time, you can enable SHARE_ALL_LIGHT_STATE_MODEL_EN to realize. Note that this is a custom rule. After enabling this macro switch, put the model IDs of the group numbers that need to be auto-bound together in the specified array. If it is a SIG model, put it in the array sub_share_model_sig_onoff_server_extend[]. If it is a vendor model, put it in sub_share_model_vendor_server_extend[].

For other details, please refer to the codes corresponding to the macros SUBSCRIPTION_SHARE_EN and SHARE_ALL_LIGHT_STATE_MODEL_EN.

6.5 Method for a Node to Get the Group Number

- Getting it through global variables

Each model has a list of group numbers, in the case of the onoff model, obtained through the model_sig_g_onoff_level.onoff_srv[i].com.sub_list[] to get it.

Double click on the value of model_sig_g_onoff_level in BDT tool to get its information. After getting the information, refer to the structure definition of model_sig_g_onoff_level and find the position of sub_list to see the group number.

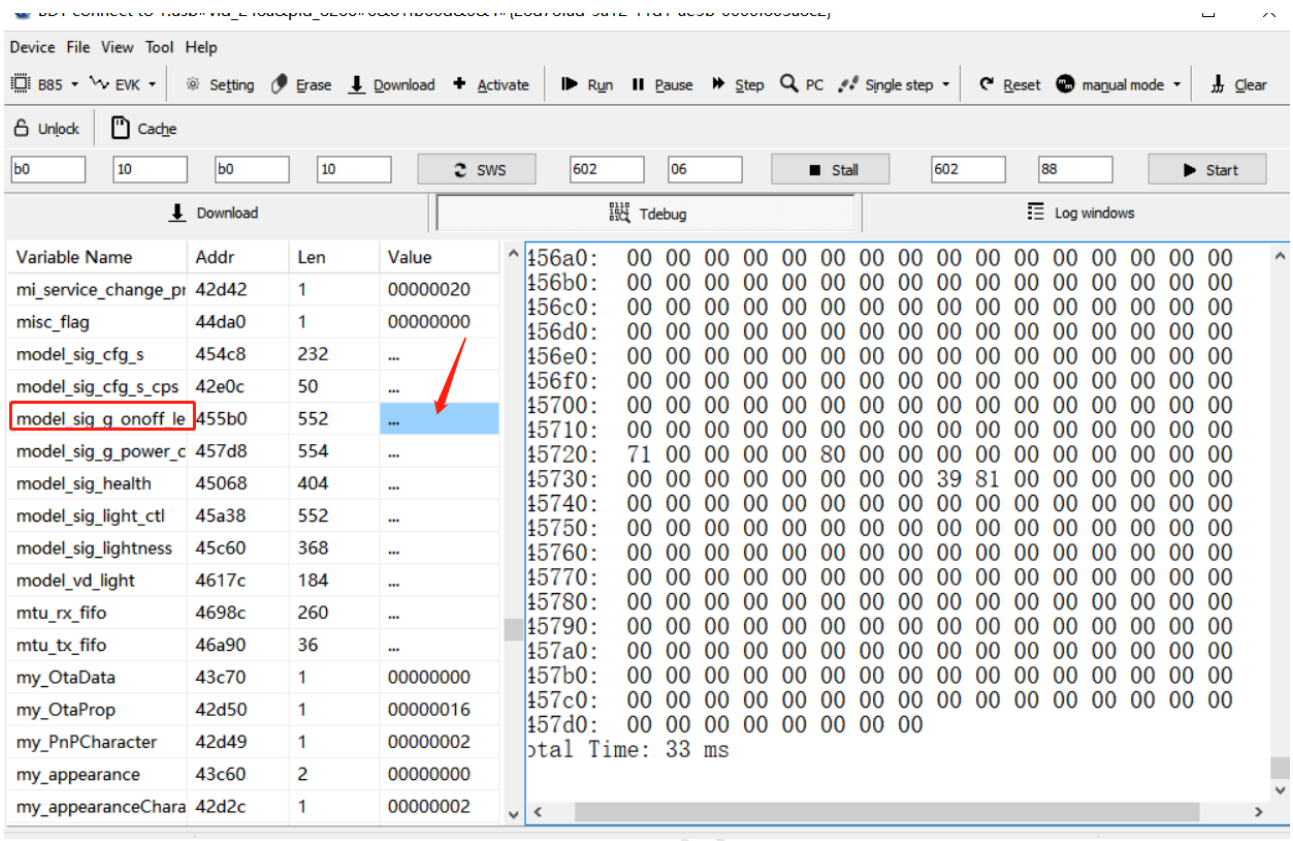


Figure 6.5: Global variable to get group number

- Get group number by model ID

The p_model pointer returned by this function mesh_find_ele_resource_in_model() is then available via p_model->sub_list[].

- Get group number of all model ID

Get the group number of all model_id's by iterating over all model global variables in MeshSigModelResource[] and then using the first method "get by global variable".

6.6 Heartbeat demonstration

The heartbeat function is detailed in the "heartbeat" section.

No heartbeat message is sent by default, user can configure this by sending command HEARTBEAT_PUB_SET. After the command is sent, the node will send out heartbeat message. Below is an example: send heartbeat message every 2 seconds, and the corresponding INI command:

```
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
```

The parameters are described as following:

- 80 39: op code

- 01 00：destination address of heartbeat is 0x0001
- ff：CountLog, 0xff means infinity
- 02：PeriodLog, period is 2 powers (02-1) i.e. 2 seconds
- 05：InitTTL, set TLL value for network layter when sending heartbeat message. This value can be customized and is not required to be equal to model_sig_cfg_s.ttl_def, as it depends on how many hops the user wants the nodes to be in range to receive the heartbeat message.
- 07 00：features, once any of relay, friend, proxy feature changes status(switches between enable and disable) the heartbeat message will be immediately reported.
- 00 00：NetKeyIndex。

Heartbeat packet can be seen in firmware tool.

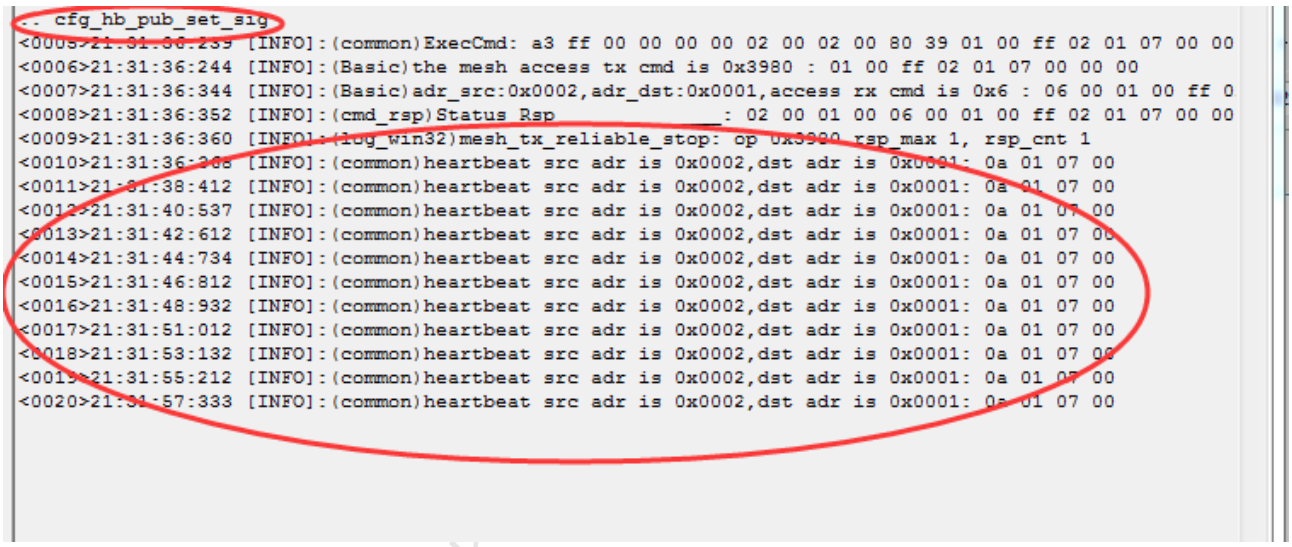


Figure 6.6: Heartbeat packet

- 0a：heartbeat opcode, please note that heartbeat is control message.
- 05：InitTTL, same value as that of heartbeat set message parameter.
- 07：Features, same value as that of heartbeat set message parameter.

When the receiver receives a heartbeat message, it will execute the callback mesh_process_hb_sub(), in which it can get the InitTTL value of the heartbeat message access layer parameter area and the ttl value of the network layer, and then subtract the two values, it will be able to know how many hops the heartbeat has gone through before it reaches the current node.

Taking the example that the ttl value of the network layer of the received heartbeat is equal to 2, the specific calculation is: hops = p_hb->iniTTL- (p_bear->nw.ttl) + 1 = 5 - 2 + 1 = 4; that is to say, the heartbeat has gone through 4 hops before it reaches the current node.

6.7 Mesh ADV Send Timing

The SDK user_init initialization calls bls_set_advertise_prepare (app_advertise_prepare_handler) to register the broadcast packet send callback function, the user is allowed to access and modify the contents of the

broadcast packet before sending the broadcast. In app_advertise_prepare_handler(rf_packet_adv_t * p), the p pointer points to the data area to be sent, and modifying the contents pointed to by p modifies the contents to be sent.

After registering app_advertise_prepare_handler(), this function will be called to broadcast packets once every 10ms by default, which is defined by ADV_INTERVAL_MIN. The reason for defining 10ms is that the unit of network transmit interval is 10ms. For example, if the network transmit interval is equal to 2, the initial value of mesh_tx_cmd_busy_cnt is set to network_tx_cmd_busy_cnt when sending a network message. Whenever app_advertise_prepare_handler() is called, mesh_tx_cmd_busy_cnt will be reduced by one, and then after reduced to 0, it will do the delay of 0~10ms, which means that it realizes the instantaneous transmission interval defined by the spec, and then the RF packet corresponding to the next transmit count can be sent.

6.8 API for Mesh ADV Payload Setting

6.8.1 Unprovisioned Device Beacon

The unconnectable broadcast packets sent by an unprovision device are for ADV provisioner discovery. The corresponding payload setting API is unprov_beacon_send(), which push the data into the mesh_adv_cmd_fifo via mesh_tx_cmd_add_packet(), and then checks the mesh_adv_cmd_fifo in app_advertise_prepare_handler( ) and sends out the data when it sees data. For details of the data format, please refer to the section "3.10.2 Unprovisioned Device beacon" in the V1.1 spec. For details of the sample data, please refer to the section "8.4 Beacon sample data".

6.8.2 Mesh Provisioning Service Advertising

A connectable broadcast packet sent by an unprovision device are for discovery by the GATT provisioner. The corresponding payload setting API is set_adv_provision(). For details of the data format, see section "7.1.2.2.1 Advertising" of the V1.1 spec. For sample data, see "8.5 Provisioning Service sample data".

6.8.3 Mesh Secure Network Beacon

The unconnectable broadcast packet sent by an provisioned node is mainly used to broadcast the IV index, as well as IV update, and for the key refresh process. The corresponding payload setting API is mesh_tx_sec_private_beacon_proc(). See "3.10.3 Secure Network beacon" in this section of the V1.1 spec for details on the data format. See "8.4 Beacon sample data" for the details of sample data.

6.8.4 Mesh Proxy ADV

The connectable broadcast packets sent after successful provisioning are for discovery and connection by the GATT proxy client. It contains network ID and node identity. The corresponding payload setting API is set_adv_proxy(). See "7.2.2.2.1 Advertising" in this section of the V1.1 spec for details on the data format. For sample data, see section "8.6 Mesh Proxy Service sample data".

6.9 Mesh Receiving Transmitting Self-defined Packet

Self-defined Packet Transmitting

When it is needed to send beacons that are not defined in the mesh spec, such as ibeacon, set BEACON_ENABLE to 1. See BEACON_ENABLE related code for details.

The SDK will call bls_set_advertise_prepare (app_advertise_prepare_handler) to register packet and send call back function when initialization, the packet can be visited and modified before sending out, SDK call the packet sending function once every 10ms by default. If user want to send self-defined packet, send it in a similar way of sending mesh-connectable packet in gatt_adv_prepare_handler. Control packet sending interval by clock_time_exceed software timing, rf_packet_adv_t * p to packet to be sent, user can modify the contents of the packet pointed to by p according to packet format(please refer to set_adv_provision()), then set the return value ret to 1, means will send packet.

Receiving/Filtering Connectable Packet

The SDK call adv_filter_proc() during RF rx interrupt to filter received packets, return 0 to abandon received packet, return 1 to keep this packet, receive and compress into blt_rxfifo without filtering. All connectable packet will be filtered by default. If user want to receive connectable packet, then open USER_ADV_FILTER_EN, in user_adv_filter_proc(), set the packet you want to return 1. It is not recommended to set all connectable packet to return 1, because this will do no filter to the packets, all packets will be pushed into blt_rxfifo, including those packets sent by other no-mesh BLE products, this may be beyond the storage capability of our receiving buffer, thus result in losing mesh message as well as the mesh packet receiving.

blt_sdk_main_loop () will check blt_rxfifo, if there is data need to be processed, it will call app_event_handle(), use may process the received connectable packet in the if(LL_TYPE_ADV_NONCONN_IND ! = (pa->event_type & 0x0F)) branch of this callback function, as shown below:

```
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
        }
```

Figure 6.7: Receiving and filtering connectable packet

6.10 Method to Modify the Maximum Number of Nodes in a Mesh Network

The Mesh products need to set the maximum number of nodes in the design phase, otherwise when sending a command to get a certain state of all nodes, such as Lightness Get All, the cache buffer will not be enough to store the location, resulting in a Lightness Get All being processed repeatedly.

The maximum number of nodes is set by MESH_NODE_MAX_NUM (default 105).

```
#if WIN32
#define MESH_NODE_MAX_NUM       1000  // 1000
#elif (FEATURE_LOWPOWER_EN)
#define MESH_NODE_MAX_NUM       105 // no need to many for LPN to save retention RAM.
#elif DEBUG_CFG_CMD_GROUP_AK_EN
#define MESH_NODE_MAX_NUM       305
#else
#define MESH_NODE_MAX_NUM       105 // gateway and node should keep the same, because of mesh
 ↪  command cache..
#endif
```

To modify the number of network nodes: Modify MESH_NODE_MAX_NUM (default 105) to the desired value. Note The gateway and nodes should be configured to the same value.

The corresponding RAM consumption for each additional node is shown in the following table:

	must	must	must	must	depend on MD_REMOTE_PROV (remote provision)	depend on MD_MESH_OTA_EN (device firmware update)	total (Max)
	sizeof (cache_buf_t)	sizeof (status_record_t)	gateway_seg_buf	sizeof (VC_node_info)	sizeof(VC_node_info_t ->dev_key_candi)	sizeof(fw_distribut_srv_proc_t ->fw_receiver_list_t)	
mesh node	6	0	0	0	0	0	6
gateway	6	4	2	20	16	7	55

Figure 6.8: RAM_Cost_for_each_node

The cache_buf is used to cache sequence number and so on to cache the sequence number of all nodes, etc.

Note: After the gateway sets MESH_NODE_MAX_NUM more than 200, an error will be prompted here when compiling:

```
STATIC_ASSERT(ARRAY_SIZE(gw_node_info) <= (FLASH_ADR_VC_NODE_INFO_END -
↪  FLASH_ADR_VC_NODE_INFO)/sizeof(VC_node_info_t)); // make sure enough flash area to save
```

Because the default 4KB flash sector can not store so many nodes' information, so you need to find a contiguous flash area to store. Then modify FLASH_ADR_VC_NODE_INFO and FLASH_ADR_VC_NODE_INFO_END accordingly.

6.11 Telink Customized Mode for Sending Mesh Messages via Extended Broadcast Package extend_adv

Extended advertising packet: extend ADV

6.11.1 Function Introduction

The B85 chip and protocol stack support sending extend ADV, but the SIG mesh spec does not define sending mesh messages via extend ADV yet. In some scenarios, we need to use extend ADV to improve the efficiency of sending messages, such as transmitting compressed image data, performing mesh OTA, etc. Therefore, we define a mode to send messages via extend ADV.

This mode specifies that one of the formats of the extend ADV defined by the BLE spec is used, as shown below:

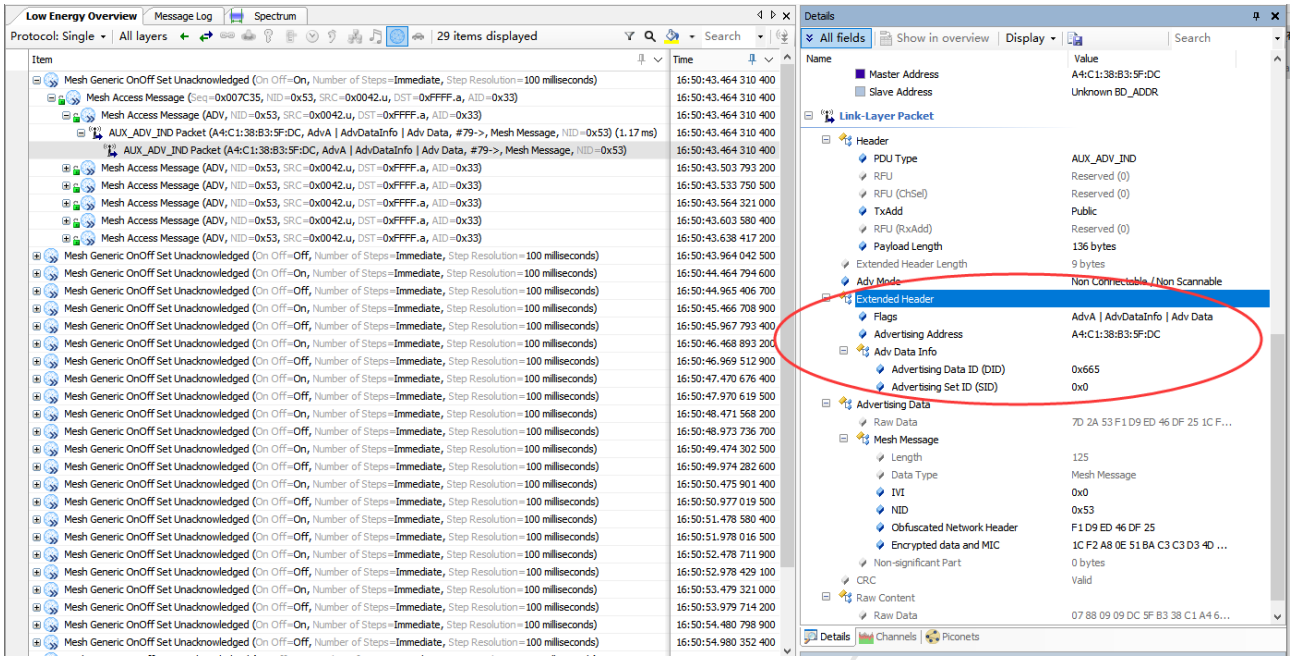


Figure 6.9: extend_ADV format

And the ADV payload is increased from 31 bytes to 245 bytes (ADV_EXTEND_PAYLOAD_MAX), that is, the length of network PDU is increased by 214bytes (CONST_DELTA_EXTEND_AND_NORMAL), and the logic of other packet sending remains unchanged, including transmit interval and transmit count. When the sent access layer, i.e. (opcode + parameters) exceeds (11+214 = 225) bytes, the segment packet grouping process will also be executed.

To summarize, when transmitting at full load, the packet sending speed is increased to about 225/11 = 20 times the original speed.

6.11.2 Test Methods

6.11.2.1 Node Configuration

The EXTENDED_ADV_ENABLE of firmware SDK is set to 1.

After enabling EXTENDED_ADV_ENABLE, by default, firmware SDK only sends mesh OTA command in extended ADV format, such as FW_UPDATE_START, BLOB_CHUNK_TRANSFER and BLOB_BLOCK_STATUS, while other packets are still sent in the same way as before, i.e., segmented packets, because we have to consider the common commands can also be interconnected with that from other manufacturers. See function: is_not_use_extend_adv(); for details. (Note: Older versions before V3.3.3 are is_extend_unseg2short_unseg()).

If there is a need for all commands with access layer less than 225 bytes (including op code) to be unsegment packet, return 0 in is_not_use_extend_adv().

If adding the rule that all vendor op codes are sent in extend ADV format, then return 0 in is_not_use_extend_adv() when judging that (IS_VENDOR_OP(op)) returns 1.

6.11.2.2 Provisioner Configuration

- sig_mesh_tool.exe Upper Configuration

The value of the "ExtendAdv" control needs to be modified in the following way:

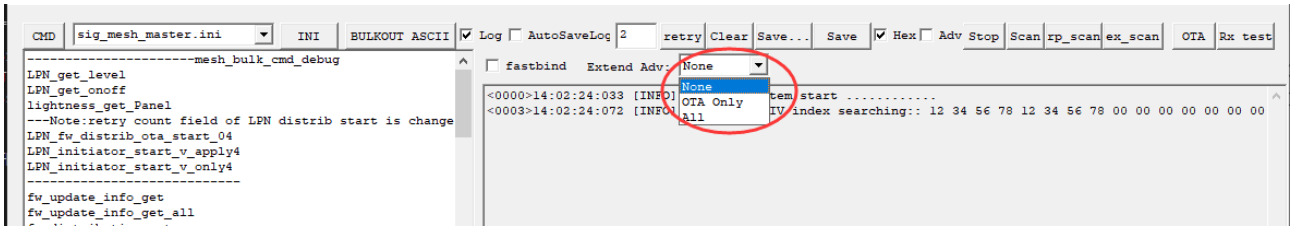


Figure 6.10: Extend_Adv_Option

See is_not_use_extend_adv() for details on how to handle this function.

(1) Select None.

(2) Select OTA only, then the host computer only sends the default FW_UPDATE_START, BLOB_CHUNK_TRANSFER, BLOB_BLOCK_STATUS, BLOB_PARTIAL_BLOCK_REPORT and other ops to the mesh OTA with extend ADV(). The purpose of sending with extend ADV is only to speed up the mesh OTA, other commands are compatible and nodes that do not support extend ADV can control each other.

(3) Selecting all means that the host computer sends all commands with an access layer length (opcode + parameters) less than 225 bytes in single-packet extend ADV format.

- Mobile App Configuration

  Please turn on the Extended Long Pack option:

  setting – setting – Extend Bearer Mode select "Extend GATT & ADV".

6.11.2.3 Precaution

Currently, only the B85 and B91 support the extend ADV function. Other chip models are not supported at this time.

6.12 Application of Soft Timer

6.12.1 Introduction of Soft Timer

(This is just an introduction to how to use it, details can be found in the B85 single connection handbook).

In order to facilitate users to do some simple timer tasks, Telink BLE SDK provides blt software timer demo, and all the source code is provided. Users can use the timer directly after understanding its design idea, or they can do some modification design by themselves.

The soft timer is especially suitable for adding timer tasks in low-power applications, so that the timer can be woken up to complete the timer tasks even in the sleep state. The soft timer can also be used in non-low-power applications.

The source code is in vendor/common/blt_soft_timer.c and blt_soft_timer.h. If you want to use it, change the following macros to 1 first:

```
#define BLT_SOFTWARE_TIMER_ENABLE 0 //enable or disable
```

The blt soft timer is a query timer designed based on system tick, its accuracy cannot be as accurate as hardware timer, and it needs to be queried all the time in main_loop.

We have scheduled: blt soft timer is used when the timing time is more than 5ms, and the requirement of time error is not particularly high. The most important feature of blt soft timer is that it is not only queried in main_loop, but also ensures that the timer can be woken up and executed in time after entering suspend, which is based on the "application layer wake-up timer" introduced in the section of low-power wake-up. Currently, the design supports up to 4 timers running at the same time, actually users can modify the following macros to realize more or less timers.

```
#define MAX_TIMER_NUM 4 //timer max number
```

6.12.2 Soft Timer Initialization

Call the following API for initialization:

void blt_soft_timer_init(void):

It can be seen that the initialization on the source code registers blt_soft_timer_process as a callback function for the application layer to wake up early.

```
void blt_soft_timer_init(void){

    bls_pm_registerAppWakeupLowPowerCb(blt_soft_timer_process);

}
```

6.12.3 Query Processing for Soft Timer

The query processing of the blt soft timer is implemented using the blt_soft_timer_process function:

void blt_soft_timer_process(int type):

The type of blt_soft_timer_process parameter has the following two cases: 0 means querying the function in main_loop, and 1 means the function is accessed when an early timer wakeup occurs.

```
#define MAIN_LOOP_ENTRY 0
#define CALLBACK_ENTRY 1
```

6.12.4 Task Configuration of Soft Timer

If the user wants to use a timer to realize certain functions, he can use the following API to add a timer task, using the method ():

(1) Define your own soft timer function: this function's function is to send ALL ON commands periodically (just an example).

```
int soft_timer_switch_send_all_on(void)
{
    access_cmd_onoff(ADR_ALL_NODES, 0, G_ON, CMD_NO_ACK, 0);
    LOG_USER_MSG_INFO(0, 0, "%s", __func__);
    return 0;
}
```

(2) Adding timer task

Use the following API to add.

int blt_soft_timer_add(blt_timer_callback_t func, u32 interval_us):

func is a task function to be executed periodically; interval_us is the timing time in us.

The int return value of the timed task func is handled in three ways:

- If the return value is less than 0, the task is automatically deleted after execution. You can use this feature to control the number of times the timer is executed.
- Returns 0, the previous interval_us is always used for timing.
- If the return value is greater than 0, the return value is used as the new timer period in us.

6.12.5 Task Deletion of Soft Timer

In addition to using the above return value less than 0 to automatically delete a timer task, you can also use the following API to specify the timer task to be deleted.

int blt_soft_timer_delete(blt_timer_callback_t func):

6.12.6 Example of Soft_timer Cycle Send Command

The following example implementation is based on the 8258_mesh_switch project.

(1) Turn on BLT_SOFTWARE_TIMER_ENABLE.

(2) In the execution of blt_soft_timer_init(); after adding blt_soft_timer_add() can be. The sample code is as follows:

The following code starts a soft timer task by pressing the key RC_KEY_R for the first time. This task is to call soft_timer_switch_send_all_on() every 500ms to send a command. When the time is not up, the node is in sleep state.

Pressing key RC_KEY_R again closes this task and stops sending commands.

```
void mesh_proc_keyboard ()
{
    ......

    else if (kb_event.keycode[0] == RC_KEY_R){// will enter here once when RC_KEY_R is pressed
     ↪  and release.
        static u32 press_cnt = 0;
        press_cnt++;
        if(press_cnt & 0x01){
            blt_soft_timer_add(soft_timer_switch_send_all_on, 500 * 1000); //
        }else{
            blt_soft_timer_delete(soft_timer_switch_send_all_on);
        }
    }
}
```

For the definition of the return value of soft_timer_switch_send_all_on(), please refer to the parameter "func" of blt_soft_timer_add function.

6.13 Use of the Long Sleep Interface

The Long sleep interface is not recommended for any sleep within 230 seconds. This is because the timing method needs to be modified after waking up and the timing accuracy is reduced a bit. Also the SUSPEND_MODE mode of long sleep should not be used because of the high power consumption.

Using the Long Sleep interface, you can set the sleep time to a maximum of 37 hours.

6.13.1 Function Name

```
/**
 * @brief      This function servers to wake up the cpu from sleep mode.
 * @param[in]  sleep_mode - sleep mode type select.
 * @param[in]  wakeup_src - wake up source select.
 * @param[in]  wakeup_tick - the time of sleep.unit is 31.25us,1ms = 32.
 * @return     indicate whether the cpu is wake up successful.
 */
int cpu_long_sleep_wakeup(SleepMode_TypeDef sleep_mode,  SleepWakeupSrc_TypeDef wakeup_src,
 ↪  unsigned int  wakeup_tick);
```

6.13.2 Use Methods

- Enable MESH_LONG_SLEEP_WAKEUP_EN.
- If there is a need for deep retetion mode, PM_DEEPSLEEP_RETENTION_ENABLE needs to be enabled.

- Call cpu_long_sleep_wakeup.

Test Example:

(1) Call cpu_long_sleep_wakeup(SUSPEND_MODE, PM_WAKEUP_TIMER, 40 * 32 * 1000) in the main loop; means to set up a wakeup after 20s of sleep, and add a log to record the current time before running the long sleep function.

```
    LOG_USER_MSG_INFO(0, 0, "system_time_s: %d, system_time_100ms: %d, system_time_ms: %d!",
↪   system_time_s, system_time_100ms, system_time_ms);
    cpu_long_sleep_wakeup(DEEPSLEEP_MODE_RET_SRAM_LOW32K, PM_WAKEUP_TIMER, 40 * 32 * 1000);
```

(2) Observe if the time result printed by the log is the same as the sleep time.

(3) Observe if system_time_100ms, system_time_s are accurate.

```
[14:40:51.484 (Rx)]
[USER]:(USER)Start user init...
[14:40:51.541 (Rx)]
[USER]:(USER)system_time_s: 0, system_time_100ms: 0, system_time_ms: 5!
[14:41:31.540 (Rx)]
[LIB]:(sdk)mesh tx NoAck,op:0x4e82(LIGHTNESS_STATUS),src:0x19f0,dst:0xffff,sno:0x000001 par_len:5 par:00 00 ff ff 0a
[14:41:31.558 (Rx)]
[USER]:(USER)system_time_s: 40, system_time_100ms: 400, system_time_ms: 40011!
[14:42:11.537 (Rx)]
[USER]:(USER)system_time_s: 80, system_time_100ms: 800, system_time_ms: 80026!
```

Figure 6.11: Long sleep 40s test

(4) Modify the 40 seconds in step 1 to 600 seconds, and repeat steps 2 and 3.

6.14 Wakeup Source Identification Interface

The wakeup source identification interface can get the current wakeup source, there are four wakeup sources in total.

```
enum{
    CPU_POWER_RESET,      // Power-on reset wakeup
    CPU_WATCHDOG_RESET,   // Watchdog reset wakeup
    CPU_PAD_WAKEUP,       // Wake up with a button
    CPU_TIMER_WAKEUP,     // Timed wakeup
};
```

6.14.1 API Function Name

```
/**
 * @brief       This function server to get cpu wakeup source
```

```
 * @return      CPU_WATCHDOG_RESET: watchdog reset.
 *              CPU_PAD_WAKEUP: gpio wakeup.
 *              CPU_TIMER_WAKEUP: timer wakeup.
 *              CPU_POWER_RESET: power reset.
 * @note        function called must be after "cpu_wakeup init()" and before wakeup io
 ↪  setting(if exist).
 */
int get_cpu_wakeup_source()
{
    if(read_reg8(0x72) & BIT(0)){
        write_reg8(0x72, BIT(0)); // manual clear watchdog reset flag after read.
        return CPU_WATCHDOG_RESET;
    }

    u8 val = analog_read(0x44);
    if((val & WAKEUP_STATUS_TIMER_PAD ) == WAKEUP_STATUS_PAD){
        return CPU_PAD_WAKEUP;
    }
    else if((val & WAKEUP_STATUS_TIMER_CORE ) == WAKEUP_STATUS_TIMER_CORE){
        return CPU_TIMER_WAKEUP;
    }

    return CPU_POWER_RESET;
}
```

6.14.2 Use Methods

(1) Call int get_cpu_wakeup_source() where you need to get the wakeup source, and get the wakeup source based on the return value.

(2) After calling the wake-up source test interface, clear bit(0) of digital register 0x72 to zero, otherwise the next call to the function will default to recognizing the wake-up source as a watchdog wake-up.

Note:

> The wakeup source recognition interface should not be called until after cpu_wakeup init(), because the MCU is not yet able to perform analog_read() to read the analog registers before cpu_wakeup init() is executed.

6.15 Key Scanning

The demo sdk turns on UI_KEYBOARD_ENABLE to enable the key scanning function, which detects the input of matrix keyboard keys or buttons.

The keypad detection is described in detail in the "Keystroke Scanning" section of this document, AN-21112301-C_Telink B85m BLE Single Connection SDK Developer Handbook.pdf. The download link is:

Chinese version:

AN-21112301-C_Telink B85m BLE Single Connection SDK Developer Handbook.pdf

English version:

AN-21112300-E_Telink B85m BLE Single Connection SDK Developer Handbook.pdf

The sdk has already been adapted for the 8258 dongle, 8258 development board and 8258 Switch PCBA by default. If you want to redo the PCBA, you can focus on the following aspects:

6.15.1 Matrix Keyboard Mode

(1) KB_LINE_MODE set to 0.

(2) KB_LINE_HIGH_VALID valid level setting, 0 for low validity, 1 for high validity.

(3) KB_DRIVE_PINS, KB_SCAN_PINS modified per new PCBA.

(4) Configure the FUNCTION, INPUT, and pull-down attributes of the GPIOs corresponding to KB_DRIVE_PINS and KB_SCAN_PINS.

(5) If there is a need to modify the KEY map, just modify KB_MAP_NORMAL.

6.15.2 Button Mode

(1) KB_LINE_MODE set to 1.

(2) KB_LINE_HIGH_VALID valid level setting, 0 for low validity, 1 for high validity.

(3) KB_DRIVE_PINS has no real meaning, just set it to 0 or the first GPIO of the scan pin, KB_SCAN_PINS is modified according to the new PCBA.

(4) Configure the FUNCTION, INPUT, and pull-down attributes of the GPIO corresponding to KB_SCAN_PINS.

(5) If there is a need to modify the KEY map, just modify KB_MAP_NORMAL.

7 Vendor Model Introduction

7.1 Adding vendor model

Normally, it is not necessary for users to add model, because currently SIG model is completed, vendor model can use the already added vendor model: VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, only need to add op code.

If user want to publish multiple status for current vendor model, new model need to be added. It not recommended to do so. Please contact us, or refer to MD_SCENE_EN to add new model.

7.2 Adding vendor command register reference

7.2.1 vendor_opcode

Command and opcode are refer to the same item in this article, op code (1BYTE) + vendor id(2BYTE).

Vendor model have 64 op codes in total. Note, it's not each product type has 64, it's all the product with the same vendor id in the whole mesh network has 64 in total. When MESH_USER_DEFINE_MODE chooses MESH_NORMAL_MODE, 32 must be reserved for Telink, i.e., 0xC0—0xDF, and 0xE0—0xFF is for users. It it recommended to use them by sub-commands way. Some Telink self-defined function will be disabled if user use more than 32 op codes. Please contact us in this case.

The maximum length of vendor command parameters is 377byte, but SIG mesh bottom layer will automatically de-pack packets longer than 8 byte, and the efficiency will be reduced. Therefore, it is recommended that keep frequently used control commands not longer than 8 byte.

7.2.2 Steps of Adding Vendor Opcode

VENDOR_OP_MODE_SEL is set to the default VENDOR_OP_MODE_DEFAULT.

In order to facilitate the user to quickly add the vendor opcode, the user can directly use the opcode demo defined by VENDOR_OP_USER_DEMO_EN, and no need to add a new opcode. the user can directly use these four opcodes and only need to change the corresponding callback functions cb_vd_user_demo_set(), cb_vd_user_demo_get() to the expected function, then it can be used quickly. In addition, users adding new vendor opcode can also refer to the opcode demo defined by VENDOR_OP_USER_DEMO_EN to add more vendor opcode.

```
#if (VENDOR_OP_USER_DEMO_EN)
    CMD_NO_STR(VD_MESH_USER_DEMO_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
↪   cb_vd_user_demo_set, VD_MESH_USER_DEMO_STATUS),
    CMD_NO_STR(VD_MESH_USER_DEMO_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
↪   cb_vd_user_demo_get, VD_MESH_USER_DEMO_STATUS),
    CMD_NO_STR(VD_MESH_USER_DEMO_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S,
↪   cb_vd_user_demo_set, STATUS_NONE),
```

```
    CMD_NO_STR(VD_MESH_USER_DEMO_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C,
↪   cb_vd_user_demo_status, STATUS_NONE),
#endif
```

Additionally, if there is data to be stored, call mesh_common_store() inside cb_vd_user_demo_set().

```
mesh_common_store(FLASH_ADR_MD_VD_LIGHT);
```

Then the parameter sno_vd_user_demo added to model_vd_light_t will be stored in this sector of flash FLASH_ADR_MD_VD_LIGHT. After re-powering up, sdk has implemented mesh_flash_retrieve() to read the contents of the sector to the corresponding global variable. Note that the structure and size of model_vd_light_t must not change before or after OTA, otherwise the data will be read abnormally after OTA.

This document takes the group of commands VD_GROUP_G_SET / VD_GROUP_G_GET / VD_GROUP_G_SET_NOACK / VD_GROUP_G_STATUS as an example to introduce the following.

7.2.2.1 Add Definition of Vendor Opcode

```
// op cmd 11xxxxxx yyyyyyyy yyyyyyyy (vendor)
// --------------------------------from 0xC0 to 0xFF
#if (VENDOR_OP_MODE_SEL == VENDOR_OP_MODE_SPIRIT)
    ......
#elif(VENDOR_OP_MODE_SEL == VENDOR_OP_MODE_DEFAULT)
// ------ 0xC0 to 0xDF for telink used
    ......
#define VD_GROUP_G_GET                  0xC1
#define VD_GROUP_G_SET                  0xC2
#define VD_GROUP_G_SET_NOACK            0xC3
#define VD_GROUP_G_STATUS               0xC4
    ......
#endif
```

7.2.2.2 Add Registration of Vendor Opcode

Vendor model registration reference code vendor_model.c implementation of mesh_cmd_sig_func_t const mesh_cmd_vd_func[] = {...}.

```
const mesh_cmd_sig_func_t mesh_cmd_vd_func[] = {
#if (VENDOR_OP_MODE_SEL == VENDOR_OP_MODE_SPIRIT)
    ......
#elif(VENDOR_OP_MODE_SEL == VENDOR_OP_MODE_DEFAULT)
    ......
```

```
    CMD_NO_STR(VD_GROUP_G_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_group_g_set,
↪   VD_GROUP_G_STATUS),
    CMD_NO_STR(VD_GROUP_G_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_group_g_get,
↪   VD_GROUP_G_STATUS),
    CMD_NO_STR(VD_GROUP_G_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_group_g_set,
↪   STATUS_NONE),
    CMD_NO_STR(VD_GROUP_G_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C, cb_vd_group_g_status,
↪   STATUS_NONE),
#endif
    ......
};
```

Note:

> mesh_cmd_vd_func[] is a const array, so this is a read-only array, thus can save RAM space.

7.2.2.3 mesh_cmd_sig_func_t introduction

```
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: new-added command's opcode, no matter it is SIG command or vendor command, is expressed as u16, vendor command do not need to fill the vendor id bytes, library bottom layer will add automatically.
- status_cmd: If the opcode is "status command" corresponding to certain "acknowledge request command", e.g. VD_LIGHT_ON/OFF_STATUS, the "status_cmd" should be set as 1; otherwise it should be set as 0. When model_id_rx is client model, "status_cmd" should be set as 1. This status_cmd flag is used in Library.
- model_id_tx: Corresponding model ID sending this command. E.g., when publish status, first, check mesh_cmd_vd_func[] according to op to get model_id_tx, then get the corresponding global veriables, such as model_sig_g_on/off_level.on/off_srv, then get the model_sig_g_on/off_level.on/off_srv->com. pub_adr and its publish parameter, finally publish status.
- model_id_rx: Corresponding model ID receiving this command. If the node does not have corresponding model id in composition data, this opcode won't be processed.

When supports a specific model, the model parameters should be checked to determine if the model has bond corresponding app key, when the destination address is a group address, if the model has follow the corresponding group.

- When this command is received, callback processing function mesh_rc_data_layer_access_cb()->p_res->cb() is invoked, users can process their own app in this callback function.
- op_rsp：If this opcode is "acknowledge request command", the "op_rsp" should be set as corresponding ack command; otherwise it should be set as "STATUS_NONE". The sending end will use this to determine if it has received the corresponding status response after it send the command.

7.2.2.4 Adding Command Callbacks

Adding a callback function for the VD_GROUP_G_SET command

```
int cb_vd_group_g_set(u8 *par, int par_len, mesh_cb_fun_par_t *cb_par)
{
    int err = -1;
    int pub_flag = 0;
    //model_g_light_s_t *p_model = (model_g_light_s_t *)cb_par->model;
    vd_group_g_set_t *p_set = (vd_group_g_set_t *)par;
    u8 sub_op = p_set->sub_op;

    if(!cb_par->retransaction){
        cb_vd_group_g_sub_set p_cb_set = (cb_vd_group_g_sub_set)search_vd_group_g_func(sub_op, SEARCH_VD_GROUP_G_FUNC_
        if(p_cb_set){
            pub_flag = p_cb_set(par, par_len, cb_par);
        }else{
            return -1;
        }
    }

#if DEBUG_CFG_CMD_GROUP_AK_EN
    if(par_len >= 3){
        if(par[2] && (par[2]<=128) && BIT_IS_POW2(par[2])){
            blt_rxfifo.num = par[2];
        }
    }
#endif

    if(VD_GROUP_G_SET_NOACK != cb_par->op){
        err = cb_vd_group_g_get(par, par_len, cb_par);
    }else{
        err = 0;
    }

    if(!err && pub_flag){
        if(is_vd_onoff_op(sub_op)){ // only onoff need publish now
            model_pub_check_set(ST_G_LEVEL_SET_PUB_NOW, cb_par->model, 1);
        }
    }

    return err;
}
```

Figure 7.1: command callback

7.2.2.5 Add TID Registration

In general, it is not necessary to add a TID; see this section for a description of the use of TIDs.：Example of Adding a Knowledge-command.

If the added command code requires a TID field (), it also needs to be additionally registered inside is_cmd_with_tid_vendor(), as detailed in the example section of Example of Adding a Knowledge-command.

7.2.3 Example of Adding a Knowledge-command

acknowledge-command, i.e. request command with status response。

Take VD_GROUP_G_SET for example.

1) Add the content below in the "mesh_cmd_vd_func[]":

```
(VD_GROUP_G_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_group_g_set, VD_GROUP_G_STATUS)
```

2) This command needs the "TID (Transmit ID)" field. Therefore, it's needed to add corresponding branch in the "is_cmd_with_tid_vendor()", and mark the location of the TID field in the access payload.

In the library, there will be a global variable mesh_tid that manages TID uniformly. When a command is sent, it will be automatically increased by one and copied to the TID field of the command parameter area. Please refer to Vendor model format for detail.

TID purpose: If repeated TID is received within a specific duration (currently it's set as 6s by default), the corresponding action won't be implemented, but it will respond with response. This recognition action is implemented in the library, while the upper APP can directly judge the flag "cb_par->re-transaction".

TID normally is for light status control commands. TID is to prevent acting repeatedly when receive retry in a specific duration, thus cause wrong delay time, for example, when the node receive OFF command, the corresponding delay time is 1s, and when the delay time passes for 0.8 s, receives retry command, if this command is executed, the delay time will re-count for 1 s, so the phenomena is that the node will be off after 1.8s. This will also cause light flash, e.g., 2 app control the same node at the same time, one act generic on, the other generic off, and both have retry action (message have different sequence numbers but same TID), for this case, what we expected is, light only execute 1 on and 1 off, the final status is determined by the last received command. This can be done if there is TID identification, without TID, the light may execute multiple on/off actions, thus causes flash.

In SIG standard commands, only light control command, such as on/off set, level set, lightness set, CT set, use TID.

Commands like lightness get, and config set/get in config model do not use TID.

Do not add vendor command if not necessary, do not waste any byte of the limited effective bytes.

3) Edit the function "the cb_vd_group_g_set()", invoke the "light_on/off_idx()" to execute light on/off action.

4) Since this command is a command that requires an ack reply, write the corresponding ack function vd_light_tx_cmd_onoff_st(), which calls mesh_tx_cmd(VD_GROUP_G_STATUS,……) for ack reply inside the function.

Note:

- To reply to status after receiving a command, you need to call mesh_tx_cmd_rsp() to reply, not mesh_tx_cmd2normal(), because in a network with multiple network keys, app keys, when receiving a packet, the packet is decrypted with whatever key is used, and when replying to status, the packet must be encrypted with the corresponding key for encryption. So use mesh_tx_cmd_rsp().
- mesh_tx_cmd2normal_primary() use the first key by default to send, normally is used for send command, when reply status, do not use this function.

5) Assemble the interface "vd_cmd_on/off()" sending the "VD_GROUP_G_SET" command.

"rsp_max" indicates the number of nodes that need response.

- When the "adr_dst" is unicast, the "rsp_max" can be set as 1 (recommended) or 0.
- When the "adr_dst" is group, the "rsp_max" should be set as the number of elements owned by group according to the record in APP database.

7.2.4 Add Unacknowledged command

Take "VD_GROUP_G_SET_NOACK" as an example.

1) Add the content below in the "mesh_cmd_vd_func[]":

```
(VD_GROUP_G_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_group_g_set, STATUS_NONE)
```

2) This command needs the "TID (Transmit ID)" field. Please refer to the method of adding acknowledge command.
3) Compile the function "cb_vd_group_g_set()" (shared with "VD_GROUP_G_SET"). Please refer to the method of adding acknowledge command.
4) This command does not need ack response.
5) Assemble the interface "vd_cmd_on/off()" sending the "VD_GROUP_G_SET_NOACK" command.

Please refer to the method of adding acknowledge command.

7.2.5 Publish function registration

The vendor model's publish functionality is generally not needed.

If necessary, the publish parameter of the light node model is set by the CFG_MODEL_PUB_SET command, the model has the publish function. When the model status changes, it will automatically publish a status message to the publish address configured by the publish parameter. In addition, the publish parameter can also be configured to send periodically, please check publish command parameter definition spec in [4.3.2.16 Config Model Publication Set] for detail.

In order to implement the above automatic publish function, you need to register the publish function for the model to send status messages, as shown below:

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

Figure 7.2: publish function

When the status changes, or after the publish cycle time expires, the mesh stack will call back this function and send a status message.

7.3 Add the Vendor Opcode Subcommand

"vendor opcode introduction" This section describes how to use Vendor subcommands. Customers can choose whether to use the subcommands or not according to their needs. The following describes how to use the vendor opcode subcommand.

7.3.1 Vendor Subcommand Range

The sub opcode of the Vendor sub-command occupies one byte, and there are only 256 values in total. 0x00 to 0x7f is reserved for Telink, and the range for user is 0x80 to 0xff, as detailed in the comment of sdk's vd_group_g_func[].

7.3.2 Steps of Adding Vendor Subcommand

In order to facilitate the user to quickly add the vendor subcommand, the user can directly use the demo bracketed by VENDOR_SUB_OP_USER_DEMO_EN without adding a new vendor subcommand, the user can directly use this subcommand and only need to change the corresponding callback functions vd_rx_group_g_sub_op_user_demo_set(), vd_rx_group_g_sub_op_user_demo_st() to quickly use the subcommand. Also, user added subcommands can refer to VENDOR_SUB_OP_USER_DEMO_EN bracketed demo to add more subcommands.

```
#if VENDOR_SUB_OP_USER_DEMO_EN
    {VD_GROUP_G_SUB_OP_USER_DEMO, vd_rx_group_g_sub_op_user_demo_set,
↪   vd_rx_group_g_sub_op_user_demo_st}
#endif
```

In addition, if you need to store data, you can call mesh_common_store() in vd_rx_group_g_sub_op_user_demo_set().

```
mesh_common_store(FLASH_ADR_MD_VD_LIGHT);
```

and then the parameter sno_vd_sno_sub_op_user_demouser_demo added to model_vd_light_t is stored in this sector of flash FLASH_ADR_MD_VD_LIGHT. after re-powering up, the sdk has already implemented the ability to retrieve the contents of this sector to the corresponding global variable through mesh_flash_retrieve() to read the contents of this sector into the corresponding global variable. Note that the structure and size of model_vd_light_t must not change before or after OTA, otherwise the data will be read abnormally after OTA.

The following describes an example of adding subcommands to the VD_GROUP_G_SET / VD_GROUP_G_GET / VD_GROUP_G_SET_NOACK / VD_GROUP_G_STATUS group of commands.

7.3.2.1 Add the Definition of the Vendor Subcommand

The demo SDK defines vendor's on and off as two separate commands, mainly for compatibility with old version. If you want to add this kind of command, you only need to add a sub-command, and then use another byte in the parameter area to indicate on or off. So the following is about VD_GROUP_G_ON only.

```
enum{/*vendor generic group, op code include C1-C4*/
    ......
    VD_GROUP_G_ON                       = 1,    // compatible with legacy
    ......
};
```

7.3.2.2 Add Registration of the Vendor Subcommand

Vendor subcommand registration reference code vendor_model.c vd_group_g_func_t vd_group_g_func = {……} implementation.

```
vd_group_g_func_t vd_group_g_func[] = {
    /* telink use sub op from 0x00 to 0x7f*/
    ......
    {VD_GROUP_G_ON, vd_group_g_light_onoff, vd_light_tx_cmd_onoff_st},
    ......
};
```

Note:

> vd_group_g_func is a const type array, so it is read-only and cannot be rewritten. This saves RAM space. In addition, if the added command requires a TID field, you need to register it in is_cmd_with_tid_vendor(), please refer to the "Adding an acknowledge-command" section for more details.

7.3.2.3 vd_group_g_func_t Introduction

```
typedef struct{
    u32 sub_op;
    cb_vd_group_g_sub_set cb_set;
    cb_vd_group_g_sub_tx_st cb_tx_st;
    //cb_vd_group_g_sub_rx_status cb_rx_status; // TBD, only client may use.
}vd_group_g_func_t;
```

- sub_op: subcommand.
- cb_set: Sets the handler function to be invoked when the command is received.
- cb_tx_st: Set response function with sending status.
- cb_vd_group_g_sub_rx_status: The gateway device receives the processing function of the STATUS command for this subcommand, which is currently not enabled.

7.3.2.4 Adding Subcommands Callback Functions

Adding a callback function for the VD_GROUP_G_ON subcommand

```
int vd_group_g_light_onoff(u8 *par, int par_len, mesh_cb_fun_par_t *cb_par)
{
    int pub_flag = 0;
    vd_light_onoff_set_t *p_set = (vd_light_onoff_set_t *)par;
    int light_idx = cb_par->model_idx;
    int on = !!p_set->sub_op;   // make sure bool
    light_onoff_all(on);
    if(vd_onoff_state[light_idx] != on){
        vd_onoff_state[light_idx] = on;
        pub_flag = 1;
    }else{
    }

    return pub_flag;
}
```

7.3.3 Adding Acknowledge Type Subcommand

The acknowledge-command is a request command with a status response, indicating that the command requires a status response.

Take VD_GROUP_G_ON as an example.

Two conditions need to be met for a subcommand to reply: the larger command needs to be VD_GROUP_G_SET or VD_GROUP_G_GET, and the member variable cb_tx_st inside vd_group_g_func[] is not NULL.

(1) Add vd_light_tx_cmd_onoff_st in vd_group_g_func[] at the corresponding location.

```
vd_group_g_func_t vd_group_g_func[] = {
    /* telink use sub op from 0x00 to 0x7f*/
    ......
    {VD_GROUP_G_ON, vd_group_g_light_onoff, vd_light_tx_cmd_onoff_st},
    ......
};
```

(2) Write vd_light_tx_cmd_onoff_st function

```
int vd_light_tx_cmd_onoff_st(u8 light_idx, u8 sub_op, u16 ele_adr, u16 dst_adr, u8 *uuid,
 ↪  model_common_t *pub_md)
{
    vd_light_onoff_st_t rsp;
    rsp.sub_op = !!vd_onoff_state[light_idx];
    return mesh_tx_cmd_rsp(VD_GROUP_G_STATUS, (u8 *)&rsp, sizeof(rsp), ele_adr, dst_adr, uuid,
     ↪  pub_md);
}
```

7.3.4 Add Subcommands of Type Unacknowledge

Take VD_GROUP_G_ON as an example.

Subcommands do not require a reply, one of the following conditions is satisfied: the larger command uses VD_GROUP_G_SET_NOACK or the member variable cb_tx_st inside vd_group_g_func[] is NULL.

7.3.5 Write API for Sending VD_GROUP_G_ON Command

vd_cmd_onoff()

7.3.6 Example of Adding an Empty Vendor Subcommand

The empty vendor subcommand example is an example where the callback function is empty. So to add the vendor subcommand empty example the user only needs to perform two steps.

(1) Adding the vendor subcommand definition

```
enum{/*vendor generic group, op code include C1-C4*/
    ......
    VD_GROUP_LOOP_ON                        = 1,    // compatible with legacy
    ......
};
```

(2) Adding registration of vendor subcommands

Vendor subcommand registration reference code vendor_model.c vd_group_g_func_t vd_group_g_func = {……} implementation.

```
vd_group_g_func_t vd_group_g_func[] = {
    /* telink use sub op from 0x00 to 0x7f*/
    ......
    {VD_GROUP_LOOP_ON, NULL, NULL},
    ......
};
```

8 Global Configuration File Introduction

8.1 mesh_config.h

PROXY_HCI_SEL:

For debugging, developer can choose PROXY_HCI_GATT by default.

DEBUG_VENDOR_CMD_EN:

Enable/disable vendor model debug command. Enabled by default.

FAST_PROVISION_ENABLE:

This is a private mode, can provision with multiple nodes at the same time, supports group action and relay network. Disabled by default.

MESH_USER_DEFINE_MODE:

Define authentication mode during provision, MESH_NORMAL_MODE: no OOB mode; others are static OOB mode, please refer to Connect with a platform.

SUBSCRIPTION_BOUND_STATE_SHARE_EN:

The purpose is to add the group number to the models listed in sub_share_model_sig[] and sub_share_model_vendor[] automatically after receiving the command to set the group number for onoff model. This is because the group number information is shared between models with state binding, such as Onoff model and lightness model.

Additionally, in private mode, models that do not have a state binding relationship can be configured to share group number information with each other by adding the corresponding group number to sub_share_model_sig[] and sub_share_model_vendor[].

PROVISION_FLOW_SIMPLE_EN:

Same as the standardized provision, provision nodes one by one, i.e., only one node is configuring network at the same time. When node receives app key add, automatically binds key to every model. Provisioner does not need to send key bind command. Simplify provision process and reducing provision time.

AIS_ENABLE / MI_API_ENABLE:

Please refer to Connect with a platform.

LIGHT_TYPE_SEL:

Select light type. Currently supported light types are mutually exclusive. Please refer to section 1.3 LIGHT_TYPE_SEL Introduction.

The following are model on/off control macro, e.g., MD_LIGHTNESS_EN, when it is enables, whether it enables client or server model, or both, is determined by MD_SERVER_EN, MD_CLIENT_EN and MD_CLIENT_VENDOR_EN. Check introductions of these 3 macros below.

LIGHT_TYPE_CT_EN:

Enable / Disable CT light related model, includes Light CTL Server, Light CTL Setup Server, Light CTL Temperature Server, Light CTL Client.

LIGHT_TYPE_HSL_EN:

Enable / Disable HSL light related model, includes Light HSL Server, Light HSL Hue Server, Light HSL Saturation Server, Light HSL Setup Server, Light HSL Client.

MD_LIGHT_CONTROL_EN:

Enable / Disable (Default) Lighting Control related model on/off control, includes Light LC Server, Light LC Setup Server, Light LC Client.

MD_LIGHTNESS_EN:

Enable / Disable Lightness related model, includes Light Lightness Server, Light Lightness Setup Server, Light Lightness Client.

MD_LEVEL_EN:

Enable (Default) / Disable Generic Level Model. Each status can have a corresponding level model.

MD_MESH_OTA_EN:

Enable / Disable (Default) Mesh_OTA_Model interface.

MD_ONOFF_EN:

Enable (Default) / Disable Generic On/off Model.

MD_DEF_TRANSIT_TIME_EN:

Enable (Default) / Disable Generic Default Transition Time Model.

MD_POWER_ONOFF_EN:

Enable (Default) / Disable Generic Power On/ off Model. Enable / Disable at the same time with MD_DEF_TRANSIT_TIME_EN, because the parameters of these 2 models are save in the same flash sector.

MD_TIME_EN:

Disable (Default) / Enable Time Model.

MD_SCENE_EN:

Disable (Default) / Enable Scene Model.

MD_SCHEDULE_EN:

Disable (Default) / Enable Schedule Model. Disable / Enable at the same time with MD_TIME_EN, because schedule depends on time.

MD_PROPERTY_EN:

Enable / Disable Property model, includes Generic User Property Server，Generic Admin Property Server, Generic Manufacturer Property Server, Generic Client Property Server, Generic Property Client.

MD_LOCATION_EN:

Enable/Disable Location model, includes Generic Location Server, Generic Location Setup Server, Generic Location Client.

MD_SENSOR_EN:

Enable/Disable Sensor model, includes Sensor Server, Sensor Setup Server, Sensor Client.

MD_BATTERY_EN:

Enable/Disable Battery model, includes Generic Battery Server, Generic Battery Client

MD_SERVER_EN:

Enable the SIG and vendor models of the enabled server models when the value is set to 1, e.g., lightness server and VENDOR_MD_LIGHT_S.

MD_REMOTE_PROV:

Enable/Disable Remote provision model. Default disable.

MD_CLIENT_EN:

Enable the client SIG model when the value is set to 1, e.g. lightness client.

MD_CLIENT_VENDOR_EN:

Enable client vendor model: VENDOR_MD_LIGHT_C when the value is set to 1.

MD_VENDOR_2ND_EN:

Enable the second vendor server model VENDOR_MD_LIGHT_S2 when the value is set to 1. Normally vendor model needs only 1.

Note:

> Generally nodes do not need Client Model, so Client Model is disabled for light side by default so as to save RAM. Client model is controlled by MD_CLIENT_EN and MD_CLIENT_VENDOR_EN, some light node need to enable vendor client model but not SIG client model.

FACTORY_TEST_MODE_ENABLE:

Enable (Default) / Disable factory test mode. For the convenience of factory test, in the case of no provision, default key can be used to implement simple operations such as turning on/off node, adjusting luminance.

MANUAL_FACTORY_RESET_TX_STATUS_EN:

Set whether to send NODE_RESET_STATUS to notify gateway or app after 5 times of booting/reset.

KEEP_ONOFF_STATE_AFTER_OTA:

Set whether to keep the on/off status of the light before reset.

ELE_CNT_EVERY_LIGHT:

Element no. of each light. E.g., a CT light need 2 elements, most model will put it in the first element, only Light CTL Temperature Server and corresponding level model are in the second element.

Since lightness and CTL Temperature can both be controlled by level model commands, if there is only 1 element address, when receiving level set command, it is impossible to determine whether it is to control lightness or Temperature.

Note the difference between LIGHT_CNT and ELE_CNT.

LIGHT_CNT is how many same lights in the BLE module, e.g., 2 CT lights.

ELE_CNT = ELE_CNT_EVERY_LIGHT * LIGHT_CNT is how many elements in this node. It is also the element address no. when provision.

FEATURE_FRIEND_EN:

Set whether to support Friend Feature

FEATURE_LOWPOWER_EN:

Set whether to support Low Power Feature.

FEATURE_PROV_EN:

Provision switch, need to enable.

FEATURE_RELAY_EN:

Set whether to support Relay Feature.

FEATURE_PROXY_EN:

Set whether to support Proxy Feature.

MAX_LPN_NUM:

Set the number of low power nodes supported by one friend node. Currently it's set as 2, it is recommended to limit this value less than 10(the maximum verified number) if the user need to modify this number. Too big value will cause higher possibility of packets conflict when friend reply respond to multiple LPN, and thus cause time delay and higher power consumption of LPN node, the RAM consuming will also increase.

USER_DEFINE_SET_CCC_ENABLE:

Must enable. Set whether App controlled node report notify/indication.

SEND_STATUS_WHEN_POWER_ON:

Set whether to send luminance state packet when power on, default sending address is 0xffff.

8.2 mesh_node.h

SUB_LIST_MAX:

The maximum number of subscribed addresses (i.e., group number) supported by each model.Versions before V3.3.0 (not included) cannot be modified by the user, because the macro is used in the library, and can be modified later than that version. When the modified number is greater than 8, in order to save RAM and flash parameter storage area, the default subscription of virtual address is turned off, i.e., VIRTUAL_ADDR_ENABLE is equal to 0. Generally speaking, you cannot use the virtual address, if you need to turn it on, just set VIRTUAL_ADDR_ENABLE to 1.

BIND_KEY_MAX:

Maximum supporting bind key number, cannot be modified, because this macro is used in library.

SCENE_CNT_MAX:

Maximum configurable scene number, can be modified.

8.3 app_mesh.h

8.3.1 Macro introduction

TRANSMIT_CNT_DEF:

Set message default transmit cnt, i.e., retry time of each command

Retry time = TRANSMIT_CNT_DEF + 1.

TRANSMIT_INVL_STEPS_DEF:

Set message default transmit interval, i.e., retry interval.

Retry interval = (TRANSMIT_INVL_STEPS_DEF + 1) * 10ms + (0~10) ms.

TRANSMIT_CNT_LPN_ACCESS_CMD:

For LPN node, control commands' transmit cnt is defined by TRANSMIT_CNT_DEF, e.g., friend request, friend poll, other message is defined by TRANSMIT_CNT_LPN_ACCESS_CMD, e.g., on/off status.

TRANSMIT_CNT_DEF_RELAY, TRANSMIT_INVL_STEPS_DEF_RELAY:

Relay's transmit count and transmit interval.

MESH_ADV_CMD_BUF_CNT:

Set message transmitting buffer size, excludes relay message.

MESH_ADV_BUF_RELAY_CNT:

Set relay message of relay message.

SEC_NW_BC_INV_DEF_100MS:

Set security beacon's transmitting interval when provision, unit is 100ms.

8.3.2 Function introduction

mesh_tx_cmd(material_tx_cmd_t *p)

This is a common function to send command.

1) Parameters:

```
type typedef struct{
union{  //point to parameter address
u8 *par;
u8 *p_ac;
};
union{  //parameter length
u32 par_len;
u32 len_ac;
};
u16 adr_src; //source address
u16 adr_dst;//destination address
u8* uuid; //point to virtual address
model_common_t *pub_md; // point to model parameter
u32 rsp_max; //number of nodes that need response
u16 op; // command code
u16 nk_array_idx; // network_key index
```

```
u16 ak_array_idx; // app_key index
u8 retry_cnt; // number of retry times
}material_tx_cmd_t;
```

Parameter values are determined by the parameters of the invoking function "mesh_tx_cmd2normal_primary(u16 op, u8 *par, u32 par_len, u16 adr_dst, int rsp_max)".

2) Return value

If the return value is 0, it indicates successful command execution.

If the return value is not zero, it indicates transmission failure, e.g. currently there's a command being sent, new command cannot be accepted (busy state), certain parameter is illegal, and etc.

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

This function serves to fix "adr_src" as "ele_adr_primary", and then assemble "mesh_tx_cmd()".

8.4 app_provision.c

u8 is_provision_success():

Get the status if the node is provisioned successfully.

u8 is_provision_working():

Get the status if the node is in provision process.

8.5 mesh_node.c

is_own_ele():

Determine if node's adr is the element address of its own.

8.6 mesh_common.c file introduction

HCI fifo:

hci_tx_fifo and hci_rx_fifo are fifos to define and transmit data by peripherals, e.g., gateway nodes and gateway firmware USB communication.

mesh_get_proxy_hci_type():

Define proxy type, PROXY_HCI_GATT by default. PROXY_HCI_USB is debug mode, not open to user.

mesh_tid_save():

Function to save TID. E.g. Commands such as generic on/off need to use tid. If deep sleep mode is not executed, it's not needed to save the tid (just initialize it as 0 after power on). If deep mode is executed, e.g. switch, each key press will initialize all variables including tid, in this case, the tid should be saved.

adv_filter_proc():

IRQ RX will filter the received verified correct packets with this function, e.g., abandon connectable packet. See code for detail.

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, };
```

Refer to SUBSCRIPTION_SHARE_EN introduction in mesh_config.h.

These Models are bonded by default, i.e. when setting subscribing address (assign group), these models will take effect at the same time.

How to call:

Receive 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():

The SDK will callback this function after OTA start command is received.

ota_condition_enable():

Condition to allow GATT OTA. When GATT connection is successful, and "set proxy filter" is received, "pair_login_ok" will be set as 1. (Note: set proxy filter need to be encrypted/decrypted with network key when receiving/transmitting.)

proc_telink_mesh_to_sig_mesh():

It serves to detect whether firmware type before OTA is SIG mesh or other SDK, e.g. Telink mesh. If it is not SIG mesh, product switch and parameter initialization will be executed.

mesh_ota_reboot_proc():

After mesh OTA is finished, delay for 1.5s and then reboot.

How to call: main_loop() -> mesh_loop_process() -> mesh_ota_reboot_proc()

mesh_ble_connect_cb:

Callback this function when GATT connect successfully.

mesh_ble_disconnect_cb:

Callback this function after GATT disconnect.

update_para_change_MTU():

It serves to request for BLE connection parameter update as needed, and prevent starting parameter update during discovery and provision.

gatt_adv_prepare_handler():

1) relay_adv_prepare_handler()

Relay buffer is independent, use different fifo with TX command, relay buffer can transmit packet during TX command transmit interval.

Priority: TX command -> relay -> connectable packet.

2) Others are GATT packets.

app_advertise_prepare_handler ():

When BLE stack bottom layer allows to send adv packet, it will callback this function. If there's adv packet (including connectable adv packet, beacon packet) to be sent in current task, it's only needed to set the parameter "p" as the pointer of the structure.

Priority: message Friend Node send to LPN after it receive LPN poll > TX command > relay > connectable adv packet.

1) get_adv_cmd():

The return value is pointer of mesh message packet to be sent. If it's non-zero value, it indicates there's packet to be sent, including MESH_ADV_TYPE_MESSAGE, MESH_ADV_TYPE_BEACON of SECURE_BEACON type.

2) mesh_adv_cmd_set()

Copy packet to be sent to BLE stack.

3) p_bear -> trans_par_val:

It includes transmit count and transmit interval.

4) mesh_rsp_random_delay_step

When the node receive a group address as destination address, it need to add Random delay for response. Check mesh_rc_data_layer_access_cb() for detail.

5) adv_retry_flag

Serves for cancelling network transmit interval, continuous transmission and etc., e.g., poll sent by LPN after build friendship.

app_l2cap_packet_receive ():

When BLE stack receives packet with payload, it will callback this function, and then invoke the function "blc_l2cap_packet_receive()" to analyze the data. During debugging, developer can print out the data for the convenience of analysis.

chn_conn_update_dispatch():

Negligible currently.

sim_tx_cmd_node2node():

It serves to send unreliable light ON/OFF command with the interval of three seconds for demo demonstration.

usb_id_init():

It's used to configure USB ID. When multiple dongles are connected with one PC simultaneously, they must be configured with different IDs so as to be recognized as different devices by PC.

ble_mac_init():

When parameter location of MAC is illegal value, it will randomly generate a MAC and save it.

mesh_scan_rsp_init():

It serves to fill in the fixed field of "scan rsp" during initialization, e.g. mac address. Mac needs to be used when building database, and iOS system cannot directly obtain it from the AdvA field of adv packet data. Therefore, it should be marked in the content of scan response.

mesh_scan_rsp_update_adr_primary():

It serves to fill in the fixed field of "scan rsp" during initialization, e.g. mac address. Mac needs to be used when building database, and iOS system cannot directly obtain it from the AdvA field of adv packet data. Therefore, it should be marked in the content of scan response.

publish_when_powerup():

Boot, send corresponding status after a random interval (publish_powerup_random_ms), notify app or gateway note to get online.

mesh_vd_init():

Common processing part related to mesh of multiple projects. It's invoked in "mesh_init_all()".

mesh_global_var_init():

Initialization function of global structure variable, executed before reading related parameters stored in flash. It serves to set default value of compiling, and if there are related parameters in flash, the values in flash will be used.

model_sig_cfg_s_cps:

I.e. composition data. For related definitions, please refer to the structure definition of model_sig_cfg_s_cps and spec.

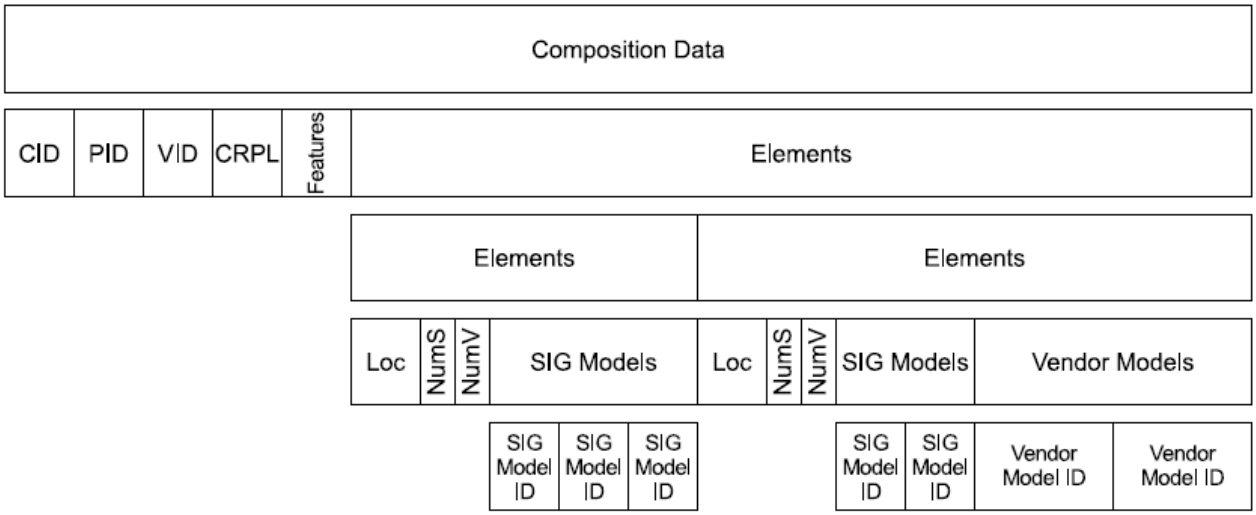


Figure 8.1: Composition data

set_unprov_beacon_para():

- p_uuid：pointer to set uuid. The length is 16 bytes.
- p_info：pointer to set oob_info. The length is 2 bytes.
- p_hash：pointer to set hash value of URI. The length is 4 bytes.
- uri_para：pointer of uri connection. The maximum length is 40 bytes.
- uri_len：Length of actually used data in uri part. The maximum length does not exceed 40.

The parameters above serve to configure beacon packet of unprovisioned node.

set_provision_adv_data():

- p_uuid：pointer to set uuid. The length is 16 bytes.
- oob_info：pointer to set oob_info. The length is 2 bytes.

The parameters above serve to configure parameters of adv packet part when provision is not finished.

set_proxy_adv_pkt():

- p_hash：pointer to set hash. The length is 8 bytes.
- p_random：pointer to set random. The length is 8 bytes.
- node_identity：It indicates adv packet type.

If "node_identity" is 0, it indicates adv type is "advertising with Network ID", in this case "p_hash" and "p_random" won't take effect.

If "node_identity" is 1, it indicates adv type is "advertising with Node Identity", in this case "p_hash" and "p_random" will take effect.

The parameters above serve to configure adv packet to send proxy connection after provision is finished.

uart_drv_init()/usb_bulk_drv_init():

Serial port and USB initialization, select serial port or USB via the macro "HCI_ACCCESS". Use "blc_register_hci_handler" to register callback function. User can invoke "my_fifo_push_hci_tx_fifo" to push data to be reported into "hci_tx_fifo".

set_material_tx_cmd():

Set transmission parameter：

1) op：vendor op code，input 1 byte, no need to input vendor id, it will be fulfilled automatically.
2) rsp_max：only effect to status replied command, serves to detect whether receives enough status. When destination address is unicast, the value is 0 or 1(0 and 1 are the same, the detection is done when receiving the status), when destination address is group, the value is the node number of the group.
3) retry_cnt：only effect to status replied command, when the command is sent for a while, and no specified rsp_max status is received, then trigger retry flow, the retry time is determined by retry_cnt.
4) uuid：when the sending destination address is virtual address, need to input uuid, otherwise it is 0.
5) nk_array_idx：mesh supports multiple netkey, this is to set the array index no. in netkey array, note, it is not the provision global netkey index.

6) ak_array_idx：mesh supports multiple appkey, this is to set the array index no. in appkey array, note, it is not the provision global appkey index.

7) pub_md：when execute publish status, it need to be set as pointer point to model, because when sending massage, mesh_model_pub_par_t parameter in pub_md-> pub_par need to be used, e.g., ttl，network transmit，network count，appkey index，but not use default values. Set pub_md to 0 when not execute publish status.

mesh_tx_cmd2normal_primary()：

Node actively send command API, see set_material_tx_cmd() for parameter detail.

SendOpParaDebug_vendor()：

In WIN32 mode，analysis of gateway，app，par_tmp[2:3]，see ini format analysis.

is_need_response_to_self()：

Set if need reply status when receive command sending by the function itself and need to answer with status.

mesh_rc_data_layer_access_cb()：

When node receives a command sent to itself (condition：model supports, destination address matches) will call this function.

1) Vendor Op code range in VD_OP_RESERVE_FOR_TELINK_START and VD_OP_RESERVE_FOR_TELINK_END is opcode reserved for Telink, not open to users.

2) mesh_need_random_delay：when node receives group address as destination address, need to add a Random delay to avoid multiple nodes respond at the same time when response.

3) p_res->cb：this is the corresponding callback function for each op code, mesh_cmd_sig_func[]->cb and mesh_cmd_vd_func[]->cb.

mesh_rsp_handle_cb()：

Reports status status message gateway received to firmware, via USB or UART.

hci_send_data_user()：

Buffer data of hci tx fifo, the first 2 byte is len, the third is data type to tell data type. Here is HCI_RSP_USER, other please refer to hci_type_t.

mesh_tx_reliable_stop_report()：

Callback function when gateway send reliable command, and the stop condition is fulfilled.

app_hci_cmd_from_usb()：

In blt_sdk_main_loop() function, callback app_hci_cmd_from_usb() to handle commands sent by firmware, analyze and execute the commands via app_hci_cmd_from_usb_handle().

app_hci_cmd_from_usb_handle ()：

The corresponding data is in ini format, check ini chapter for detail.

8.7 cmd_interface.h file introduction

access_cmd_get_level():

This function serves to obtain level value of element by setting "opcode" as "G_LEVEL_GET" and then assembling "mesh_tx_cmd()".

access_cmd_set_level():

This function serves to set level value of element by assembling "mesh_tx_cmd()". When ack is 1, set opcode as "G_LEVEL_GET". When ack is 0, set opcode as "G_LEVEL_SET_NOACK". The SDK will manage and implement tid parameters used in this command, so these parameters are negligible for upper development.

access_set_lum():

This function serves to set level value by inputting "lum" (range is 0~100) and assembling "access_cmd_set_level ()".

access_cmd_onoff():

This function serves to set on/off value of element by assembling "mesh_tx_cmd()". When ack is 1, set opcode as "G_ON/OFF_GET". When ack is 0, set opcode as "G_ ON/OFF _SET_NOACK". The SDK will manage and implement tid parameters used in this command, so these parameters are negligible for upper development.

8.8 vendor_model.c file introduction

This file mainly introduces transmission of opcode corresponding to vendor model, as well as corresponding callback function to be executed after this opcode is received.

Note: non-provisioner nodes use only 1 vendor id, and will show this id in composition data, Vendor model has 64 op code in total. Please be noted, this is not 64 for a product, it's 64 for all products. So please use it wisely, use as many sub-commands as possible.

Telink also uses some vendor op code for self-define features, so currently 0xC0—0xDF is reserved for Telink, and 0xE0—0xFF is for other users.

Register of vendor opcode

See Chapter 3.2.

mesh_search_model_id_by_op_vendor():

This function serves to search for related resources in the array "mesh_cmd_vd_func[]" via opcode. User does not need to modify this function.

vd_cmd_key_report():

This function serves to report key press event and it's used in the project "8258_mesh_switch".

It can be considered that "int SendOpParaDebug(u16 adr_dst, u8 rsp_max, u16 op, u8 *par, int len);" is equivalent to "mesh_tx_cmd_primary()".

is_cmd_with_tid_vendor():

This function serves to check whether this opcode needs to carry tid and return value accordingly. If this opcode needs to carry tid, it will return 1, and return the location of tid in parameter area via "tid_pos_out"; otherwise, it will return 0.

8.9 mesh_test_cmd.c file introduction

This file serves to save implementation of test commands.

9 8258 MESH Project Introduction

9.1 app_config_8258.h

PCBA_8258_SEL:

Select PCBA, default is 48pin dongle board, the other 2 are reference board.

FLASH_1M_ENABLE:

Enable this macro when internal flash is 1M because the flash map is different, and for the initial configuration, MAC is 0Xff000 while 512K MAC is 0x76000.

HCI_ACCESS:

Set HCI interface.

- HCI_USE_USB: use USB
- HCI_USE_UART: use UART

HCI is not needed for data transmission by default.

UART_GPIO_SEL:

Set UART IO.

HCI_LOG_FW_EN:

Firmware disables this function by default, user can enable this if needed, refer to log output chapter for detail.

ADC_ENABLE:

Set whether to enable ADC or not.

ONLINE_STATUS_EN:

Private mesh SDK online status function. Send real time status data, optimize real-time function of publish.

DUAL_MODE_ADAPT_EN:

SIG mesh + ZigBee dual modes.

DUAL_MODE_WITH_TLK_MESH_EN:

Enable SIG mesh + private mesh SDK dual modes.

TRANSITION_TIME_DEFAULT_VAL:

Default transition time is used when power on, and receiving a command supporting transition variable, e.g., generic on/off, but there is no transition variable in this command, the light will act according to TRANSITION_TIME_DEFAULT_VAL. The default transition time is 1s, to disable transition, set TRANSITION_TIME_DEFAULT_VAL to 0. Refer to trans_time_t for detail.

SW1_GPIO/SW2_GPIO:

Two buttons on dongle board, used for debugging, disabled by default.

To enable, first verify IO, then modify corresponding PULL_WAKEUP_SRC_XXX and XXX_INPUT_ENABLE.

PWM_R/ PWM_G/ PWM_B/ PWM_W:

Set IO corresponding to PWM.

GPIO_LED:

Set led light IO, e.g., when the provision is completed, reset to factory configuration, the definition of light flashing.

9.2 app.c file introduction

9.2.1 Customization of Adv packet and Adv response packet

Advertising packet

Connectable adv packet: Currently SIG MESH spec has already defined all fields for connectable adv packet format. Please refer to the structure "PB_GATT_ADV_DAT" or spec for details.

Advertising response packet

User can customize adv response packet by modifying the array "u8 tbl_scanRsp [] = {}" via "mesh_scan_rsp_init()". The maximum length of adv response packet can reach 31 bytes, only a part of which is used currently. User can configure the "rsv" field as needed in "mesh_scan_rsp_init()".

```
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 Configuration of FIFO part

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

The two functions serve to configure packet Rx buffer and Tx buffer in BLE stack bottom layer.Generally it's not recommended to modify them unless RAM size is not large enough.

9.2.3 app_event_handler ()

Callback processing function: When BLE stack receives adv (include connectable adv packet, beacon packet, etc), connect request packet, BLE connection parameter update packet, BLE connection termination, and etc, this callback function will be invoked after the event to process correspondingly.

Currently all beacons used for SIG MESH communication are processed in the branch "(subcode == HCI_SUB_EVT_LE_ADVERTISING_REPORT)".

For processing of other events, please refer to corresponding code. Currently only simple LED indicating light processing is contained, and related functions can be added if needed.

HCI_SUB_EVT_LE_ADVERTISING_REPORT:

Processing branch after receiving adv packet. User can add corresponding event and processing function under this branch.

HCI_SUB_EVT_LE_CONNECTION_COMPLETE:

Event callback generated after BT connection is established. BLE stack bottom layer will callback to this branch after BT connection is established.

HCI_CMD_DISCONNECTION_COMPLETE:

After BT connection is terminated, BLE stack bottom layer will callback to this branch.

9.2.4 main_loop ()

- mesh_loop_proc_prior(): function with high priority for real time features
- blt_SDK_main_loop (): main_loop function of BLE stack.
- proc_led(): LED indicating light event processing function.
- factory_reset_cnt_check(): factory reset processing function. Support reset method of five power on operations. Please refer to factory reset section.
- mesh_loop_process():SIG mesh related loop function, including retry mechanism of reliable command, segment ack timeout response, TID timeout detect mechanism, and etc.
- sim_tx_cmd_node2node(): Demo demonstration interface of ON/OFF command timed transmission.

9.2.5 user_init()

- proc_telink_mesh_to_sig_mesh(): To implement OTA between sig mesh and telink mesh with incompatible parameter format, it's needed to initialize parameters. In current SDK, when mesh type change is detected, parameter area to be used by new mesh will be cleared.
- bls_ll_setAdvParam(): Define parameters including adv packet interval, currently not recommended to modify.
- blc_ll_setAdvCustomedChannel(): Customize adv channel. Sig mesh requires to use standard channel 37/38/39. However, during test process, for the convenience of debugging, the channel can be changed.
- bls_ll_setAdvEnable(1): Enable transmission of adv packet.
- rf_set_power_level_index (MY_RF_POWER_INDEX): The default setting of transmit power is 3dbm, if you need to modify the transmit power, just modify the macro MY_RF_POWER_INDEX. If there is a dynamic modification in the middle of the process, need to call rf_set_power_level_index (my_rf_power_index) when restoring.
- mesh_init_all(): sig mesh related initialization.

9.2.6 void proc_ui()

This function mainly implements UI related processing, e.g. button detect function, as well as corresponding test code.

9.3 app_att.c file introduction

pb_gatt_provision_out_ccc_cb():

Enable transmission of "provision out" part. Only when "provision_Out_ccc" is set as "01 00", can mesh node normally return command.

pb_gatt_Write ():

Callback function corresponding to uuid of "my_pb_gatt_in_UUID" and used to process provision command.

proxy_gatt_Write():

Callback function to process proxy command. The command head of proxy command include three types: MSG_PROXY_CONFIG, MSG_MESH_BEACON, MSG_NETWORK_PDU.

- MSG_PROXY_CONFIG: It's used to configure white list and black list for proxy communication.
- MSG_MESH_BEACON: It's used to control reception of beacon command (notify).
- MSG_NETWORK_PDU: It's used to control ON/OFF command.

attribute_t my_Attributes[]:

Service list in SIG_mesh containing basic att, as well as att contents related to SIG_mesh part.

9.4 light.c file introduction

Modify IO pins

It's only needed to modify IO pins corresponding to PWM_R / PWM_G / PWM_B / PWM_W.

```
#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 invert feature
    u8 func;    // PWM first function or second function
    u8 rsv[1];
}light_res_hw_t;
light_res_hw_t light_res_hw[LIGHT_CNT][4];
```

light_res_hw defines PWM IO features.

func: GPIO_PC1, GPIO_PC4, GPIO_PD5 have 2 PWM output functions, here defines whether to use function 1 or function 2.

In PWM macro definition, all 4 leds, i.e., RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B/ RES_HW_PWM_W, on dongle board/reference board are listed, but for a specific type of light, not all of them are needed, light_res_hw is to define this, e.g.:

LIGHT_TYPE_CT: select RES_HW_PWM_R and RES_HW_PWM_G, red is for warm light bead, green is for cold light bead.

LIGHT_TYPE_HSL: RES_HW_PWM_R, RES_HW_PWM_G, RES_HW_PWM_B are corresponding to RGB beads respectively, when modify light, change HSL to RGB in dim_refresh(), to drive LED.

LIGHT_TYPE_LPN_ONOFF_LEVEL: select RES_HW_PWM_R, currently can only control onoff because the low power consumption of retention.

LIGHT_TYPE_PANEL: default value is 3, occupying 3 element addresses, with 3 onoff server models, and the corresponding beads of these 3 onoff models are RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B.

Set PWM Frequency

Just modify PWM_FREQ.

Note: PWM tick overflow will cause STATIC_ASSERT(PWM_MAX_TICK < 0x10000) error when compiling. PWM tick is 16 bits, and the default PWM clock is PLL clock, when PWM_FREQ is too small, PWM tick will overflow, in this case, user can set PWM frequency division, i.e., PWM_CLK_DIV_LIGHT. Generally it is not needed.

ct_flag:

Used only in LIGHT_TYPE_CT_HSL mode. There are CT bead and HSL bead in this mode, but only 1 will light up at the same time. ct_flag is 1, indicates this is CT bead and 0 indicates HSL bead.

light_res_sw_save:

This variable includes all light status related parameters need to be saved, e.g., lightness, CT and etc. Note, all data are transfer to generic level format (range from -32768 ~ 32767) before saved.

The reason why save data in level format:

(1) all status value can transfer to level,

(2) level is the most accurate

(3) save only 1 parameter for the same status, e.g., for CT value, you can't save both CT value and the transferred level value, because these 2 values may not synchronize.

In general, all status are saved in level format, otherwise may lose accuracy.

Nonlinear correspondence of luminance and PWM value

Developer can modify the array "rgb_lumen_map[]" according to actual light characteristic.

// 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():

The initial value when first booting is the default compiling value, when corresponding parameter is saved to flash, then use the saved value.

light_res_sw_load():

Status parameter for load light from flash.

light_pwm_init():

Call this function after read light status. Set to OFF after initializing PWM register, then check if need to enable and if it need transition parameter.

light_par_save_proc():

The light status will save to flash 3s after it changed to avoid writing flash too often.

light_dim_set_hw():

Set PWM output.It will be executed after node receives "G_LEVEL_SET"/"G_LEVEL_SET_NOACK" command.

Idx and idx2：light_res_hw[idx][idx2].

idx：light count index, e.g., when a BLE module has 2 CT lights

Idx2：light bead index.

light_dim_refresh ():

When light status changes, call this function to refresh PWM output value. In this function, users can get lightness, CT value and etc., users can calculate PWM value based on this value according to their own dimming algorithm.

The default algorithm is, change the CT value of standard lightness to 0—100 scale, then check rgb_lumen_map[101] to find the corresponding PWM output value.

get_light_pub_list():

Check all status need to be published when light status changes.

temp_to_temp100():

Change 800—20000 CT value to 0—100 scale.

temp100_to_temp():

Change 0—100 scale to 800—20000 CT value.

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;
```

Set transition parameter when receive command like level set/lightness set and transition is needed. "1p32768" in st_transition_t means dividing one level scale in 32768 units, i.e, the unit of this value is 1/32768, thus can avoid floating calculation and enhance calculation efficiency.

step_1p32768: the changing value for each LIGHT_ADJUST_INTERVAL during transition.

remain_t_ms: remain value in the command changes to ms.

delay_ms: delay_ms value in the command changes to ms.

present: real time value of level during transition.

present_1p32768: remaining part of present, unit is 1/32768 level scale.

target: target level.

light_transition_proc():

Transition polling processing function. When the transition finishes, i.e., level reach target level, if this transition is triggered by scene load, call scene_target_complete_check(i); to label that the scene is valid.

When the transition finishes, check and transmit publish status.

led_onoff_gpio():

In deep retention sleep or deep sleep mode, the output during sleep is done by setting pull-up/pull-down. In this case, PWM stops working, as well as gpio function，gpio output enable，gpio output registers, only analog registers setting pull-up/pull-down works.

proc_led():

LED indicating light polling processing function.

rf_link_light_event_callback ():

It's LED indicating light register event. By using this method, light blinking is executed in main_loop and it won't influence processing of other events.

In LPN mode, proc_led() can not be polled all the time because the SDK enters deep mode, thus the light flashes slowly, not as we expected. Current solution is to set faster flashing parameter when led indicating light is needed, then keep polling proc_led(), process other function after the flash ends. Normal LPN products need no LED, only for development.

Flashing scene introduction:

LGT_CMD_SET_MESH_INFO(LGT_CMD_PROV_SUC_EVE): light flashing when provision succeeds.

LGT_CMD_FRIEND_SHIP_OK:scene generated by LPN when LPN builds friendship with friend successfully.

LGT_CMD_SET_SUBSCRIPTION: receiving message to modify subscription address

LGT_CMD_BLE_ADV: BLE disconnecting scene, disable by default, only for debug mode.

LGT_CMD_BLE_CONN: BLE connecting scene, disable by default, only for debug mode.

LGT_CMD_SWITCH_POWERON: flash once when power switches to on.

LGT_CMD_SWITCH_PROVISION: switches to PROVISION mode.

LGT_CMD_SWITCH_CMD: switch sends press button command.

PROV_START_LED_CMD: gateway starts provision flow to a node.

PROV_END_LED_CMD：provision flow finishes.

LGT_CMD_DUAL_MODE_MESH：switch modes in dual mode status.

show_ota_result()：

It's processing function of light blinking indication after OTA is finished.

show_factory_reset()：

It's processing function of light blinking indication after implementing factory reset operation.

How to Introduce Customized Dimming Algorithm：

Default dimming algorithm：refer to light_dim_refresh().

Users can modify dimming algorithm by changing light_dim_refresh(), i.e, get standard value with this function, then call their own dimming algorithm：

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:
```

Refer to light_dim_refresh(), get HSL.h/HSL.s/HSL.l or RGB.r/ RGB.g/ RGB.b based on dimming algorithm needs.

On/off：defined by lightness.

10 Provisioner (Gateway) Project Introduction

10.1 Provisioner Function Introduction

10.1.1 adv-bearer and gatt-bearer

Provisioning is to add an unallocated device into mesh network via Provisioner, so that the device can become a node in the mesh network. The provision process mainly allocates network key and IV index (key parameters to determine whether it's the same network), as well as unicast adr (address allocated in the network). The Provisioner can use network parameters and unicast adr to access and control corresponding node, e.g. turn on/off light, adjust luminance.

Provision supports two types of link channels:

- Implement communication in adv-bearer channel via adv packet. This section mainly introduces the adv-bearer part.
- Implement communication via BLE connection and gatt-bearer. For the implementation of gatt-bearer, please refer to section 7.2, and Android/iOS APP corresponding to SIG_mesh can implement corresponding functions.

10.2 Provisioner Principle

10.2.1 Command Interaction of Provisioner

The provisioner uses "adv-bearer" to add unprovisioned device (unpaired node) into network, and adopts BT channel 37, 38 and 39 to communicate with unprovisioned device. By transferring parameters (e.g. random, key) between the provisioner and unprovisioned device, network parameters and address allocation are exchanged to finally add the unprovisioned device into the network. Please refer to section 5.3 in sig_mesh document "Mesh_v1.0" for details.

10.2.2 Timing Sequence Chart of adv Provisioner

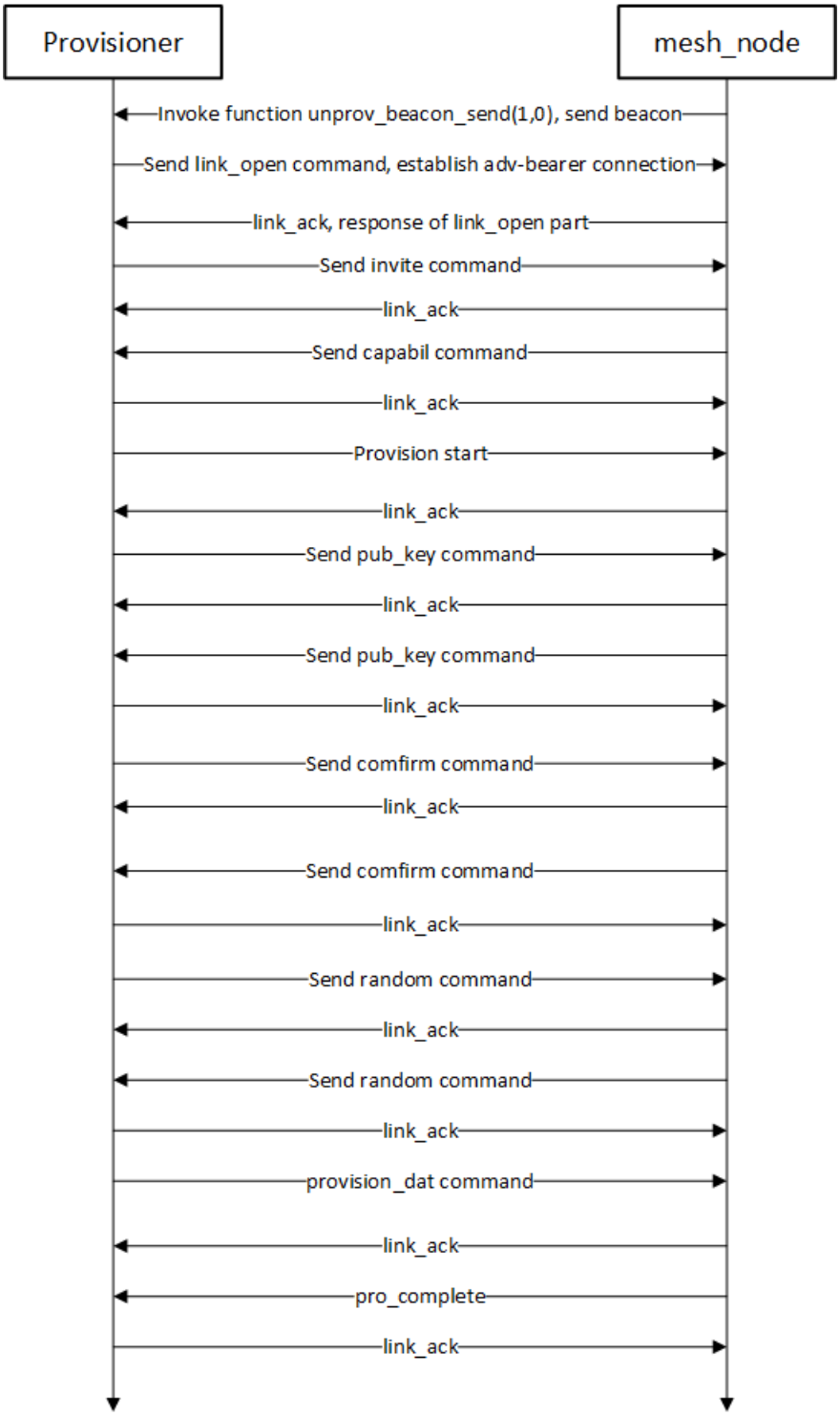


Figure 10.1: adv Provisioner Timing Sequence Chart

The "provision_dat" command contains three network parameters including network key, IV index and unicast adr, and implements the function of network formation.

Function invoking relationship chart for the packet Tx part of adv-provision:

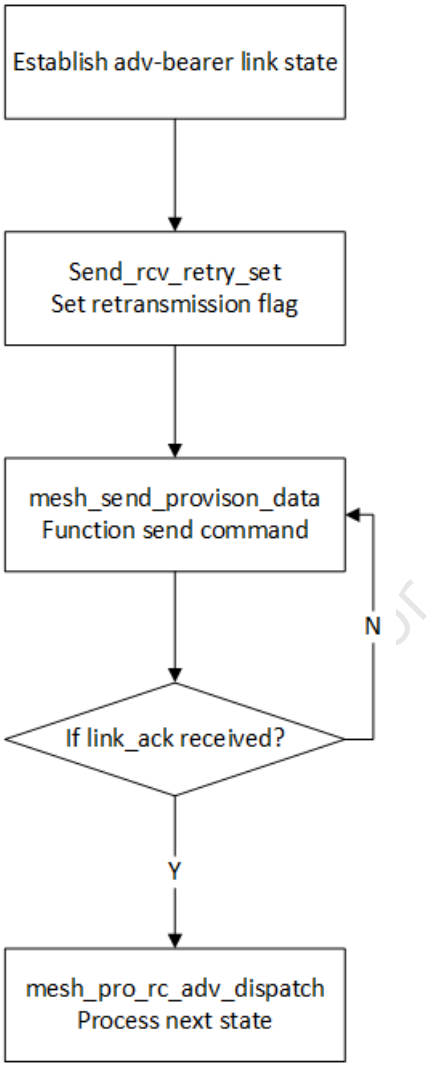


Figure 10.2: Function Invoking Relationship Chart for the packet Tx Part of Adv-provision

Function invoking relationship chart for the packet Rx part of adv-provision:

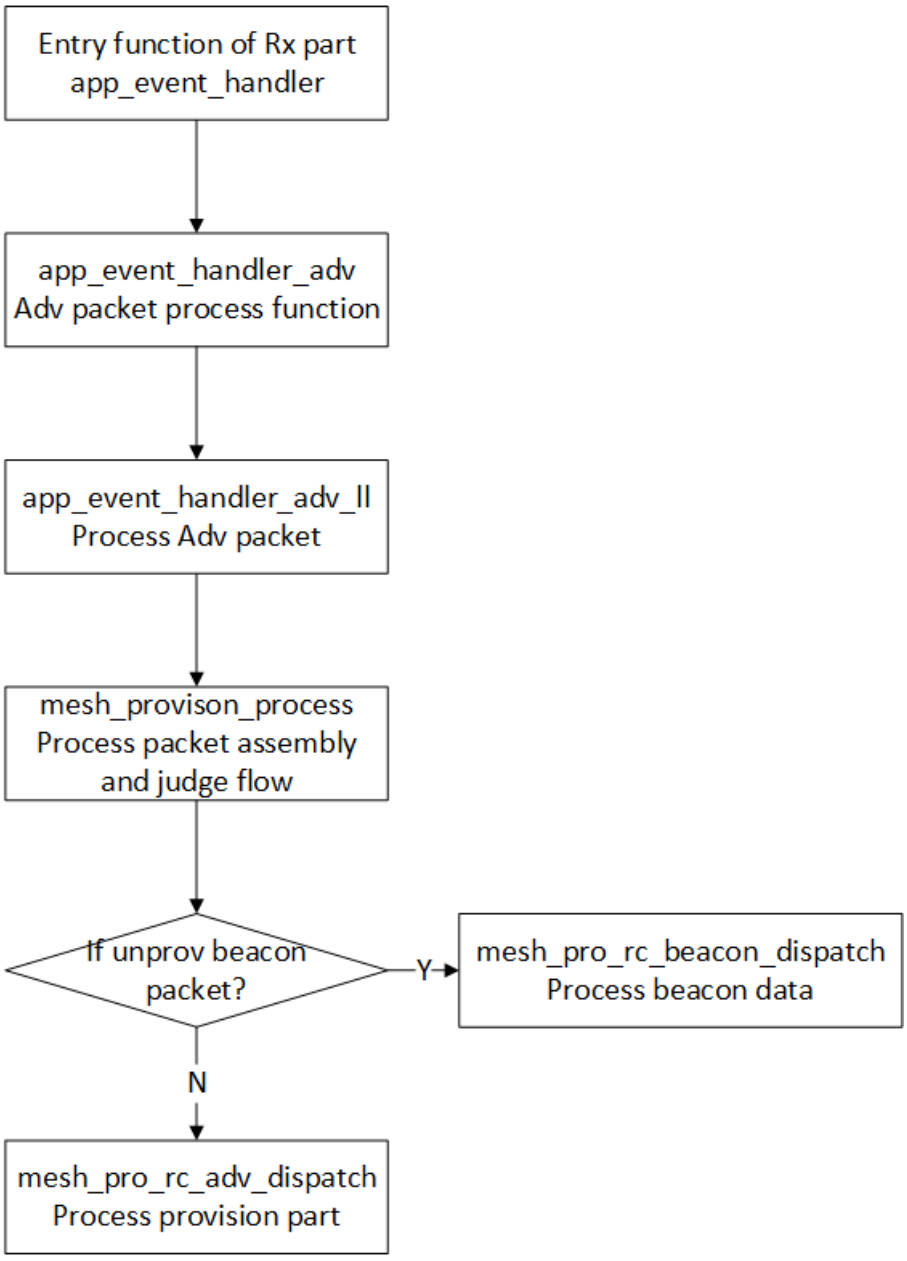


Figure 10.3: Function Invoking Relationship Chart for the Packet Rx Part of Adv-provision

10.2.3 Timing Sequence Chart of GATT Provisioner

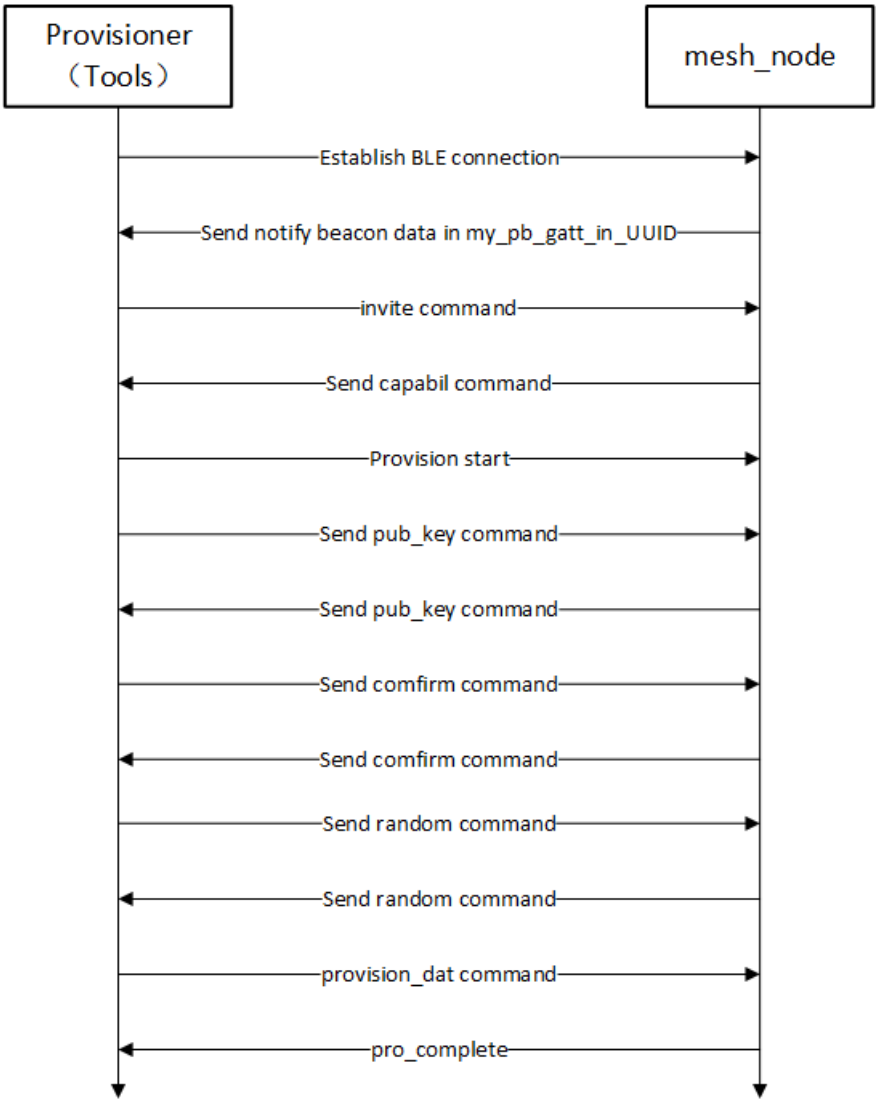


Figure 10.4: gatt provisioner Timing Sequence

By using the method of gatt-provision, the function of provision can be implemented more quickly. The "int pb_gatt_Write (void *p)" is the entry function of "gatt_provision" part.

Packet Tx function entry of gatt_provision:

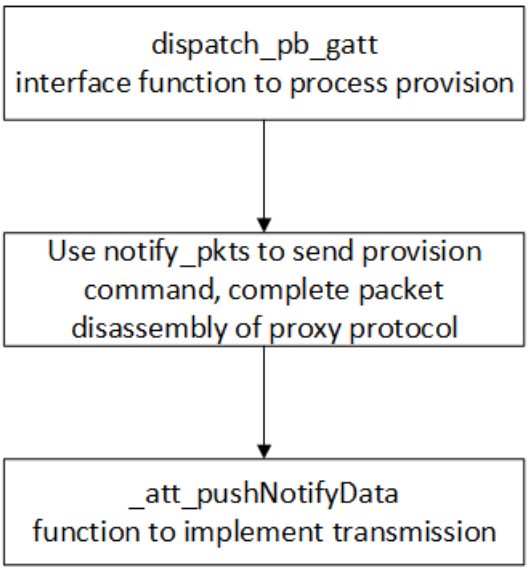


Figure 10.5: Packet Tx Function Entry of gatt_provision

Packet Rx function entry of gatt_provision:

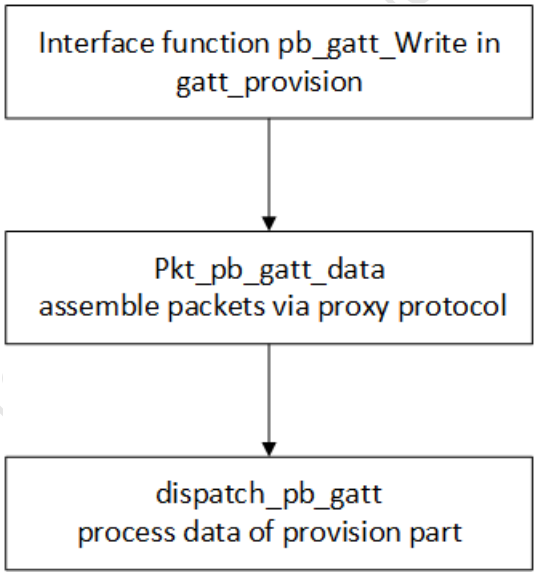


Figure 10.6: Packet Rx Function Entry of gatt_provision

10.3 app.c file introduction

In the provisioner project of current SDK, only the "app.c" file needs customized modifications.

Customization of Adv packet and Adv response packet

Please refer to Section 9.2.1.

Configuration of fifo part

Please refer to Section 9.2.2.

HCI (USB/UART) Report Data

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

p：point to address of the data to be sent

n：data length

head：head of the data

head_len：length of the head (0 if not specified)

Call my_fifo_push_hci_tx_fifo (u8 p, u16 n, u8 head, u8 head_len) to send data to hci_tx_fifo, the hci_tx_fifo data will be sent in the callback function. Call back function is registered in user_init.

- 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 ()：

Refer to section 9.2.3.

main_loop ()：

Refer to section 9.2.4.

user_init()：

Refer to section 9.2.5.

proc_ui()：

The "proc_ui" function configures IO pin scanning with the interval of 40ms to detect IO change. The interface function "access_cmd_onoff" is finally used to send ON/OFF command. By pressing SW1/SW0, an ON/OFF command will be sent with the interval of 100ms.

10.4 Provisioner operation and APIs

Mesh stack runs in gateway dogle. Node information are saved in provisioner flash with the address of FLASH_ADR_VC_NODE_INFO(0x3f000), 1 sect or is 4K, so the maximum saved node number is 200. If more than 200 nodes can not continue to add the corresponding nodes to the network correctly, you need to manually delete some offline nodes, or expand the storage area, to expand the method, please refer to "Method to Modify the Maximum Number of Nodes in a Mesh Network".

10.4.1 Format of SIG_MESH_TOOL ini file

Gateway operation will use "SIG_MESH_TOOL". User can click control buttons on the interface, or send out command via the left "command list" window (double click cmd line) or the bottom "edit control" window (compile cmd and press Enter). The provisioner will handle this in the corresponding branch of app_hci_cmd_from_usb_handle after it receives the command.

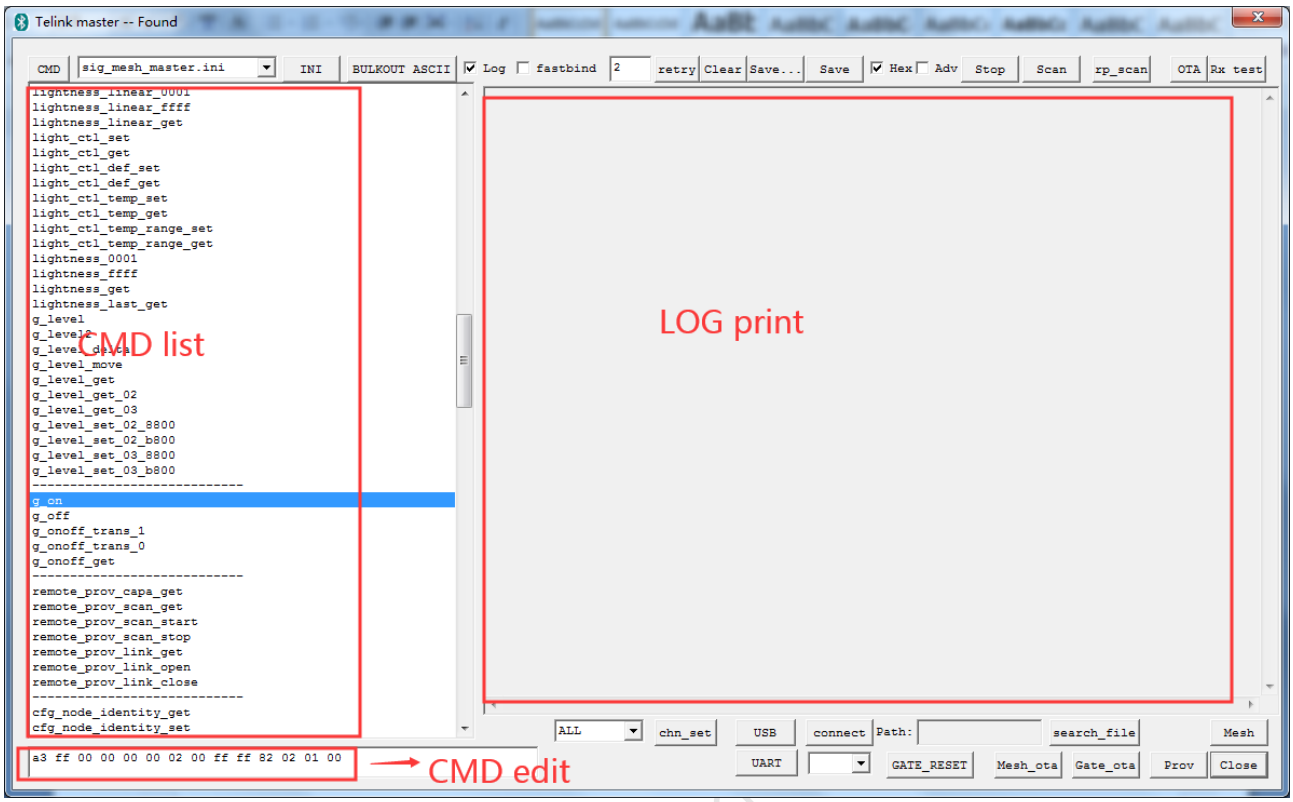


Figure 10.7: SIG_MESH_TOOL

The command has 2 formats in ini file, SIG model and vendor model.

10.4.2 SIG model format taking g_all_on as an example

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

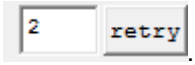
Figure 10.8: g_all_on

The first 2 bytes are identifier, defined by Telink, used to identify communication packet head, for gateway it is 0xE8FF, for app(including mobile app and kma dongle firmware) is 0xA3FF.

Parameter structure as below:

```
typedef struct{
        u16 nk_idx;     //netkey index
        u16 ak_idx;     //app_key index
```

```
        u8 retry_cnt;   // time of app layer, when rsp_max is not received, app layer will retry
        u8 rsp_max;     // expected answer number
        u16 adr_dst;    // destination address
        u8 op;          // first byte of op_code
        u8 par[MESH_CMD_ACCESS_LEN_MAX]; //rest byte of op_code and parameter
}mesh_bulk_cmd_par_t;
```

Among them, retry_cnt is in VC tool, if set to 0, it means to use the value of "retry" control, the default is 2,  . VC tool will automatically change the value of retry_cnt in the INI data to 2. If it is set to 0xFF, it means that no retry is required, that is, the VC tool will automatically change the value of retry_cnt in the INI data to 0.

Refer to set_material_tx_cmd() for detail of nk_idx，ak_idx，rsp_max.

Note:

> If TID is 0, then it will be maintained and managed by protocol stack, if TID is not 0, then use this as TID, so that users can maintain TID on their own.

10.4.3 Vendor Model Format

Take CMD-vendor_on as example

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]
e8	ff	0000	0000	02	00	ff ff	C2 1102	c4	2	1	0

Figure 10.9: CMD vender on

Different with SIG model, there are 2 more parameters, op_rsp and tid_pos, these 2 parameters are pseudo parameters, and will not be sent to light node.

op_rsp: set corresponding response opcode(vendor id is not compulsory)

tid_pos: set tid position (0 means no tid bytes, 1 means tid is in para[0], 2 means para[1]...), op_rsp and tid_pos 's configuration should be unified with that of firmware.

The purpose of these 2 parameter, is that provisioner need to support more vendor id's op code, and vendor op may be added at any time, we cannot compile all these information in the program, so we add these information via ini.

10.4.4 Burn Nodes

Burn 2 8258 dongles: 1 8258 provisioner node(8258_mesh_gw.bin), 1 dongle node(8258_mesh.bin).

Refer to Debugging Tool Instructions for burning steps.

10.4.5 Add Light via Provisioner

The gateway cooperates with the lighting process of the SIG_mesh_tool tool and the corresponding command format (hexadecimal representation).

1) Plug the provisioner dongle to the USB port. Start the "SIG_MESH_TOOL", and select "tl_node_gateway.ini". The top side of the tool will show "Found", which indicates the gateway device (provisioner) is found.

The tool will automatically obtain the uuid and mac address of the gateway. The command format is:

HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_GET_UUID_MAC

i.e., e9 ff + 10

The gateway will report uuid and mac after receiving it, the format is:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_UUID+uuid(16 bytes) +mac(6 bytes), i.e., 91+99+uuid(16 bytes)+mac(6 bytes).

2) Power on the 8258 dongle, and then click the "Scan" button on the tool to start scanning for devices.

The corresponding command of the scan control is HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_START: e9 ff + 00

The command corresponding to the stop control is HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_STOP: e9 ff + 01

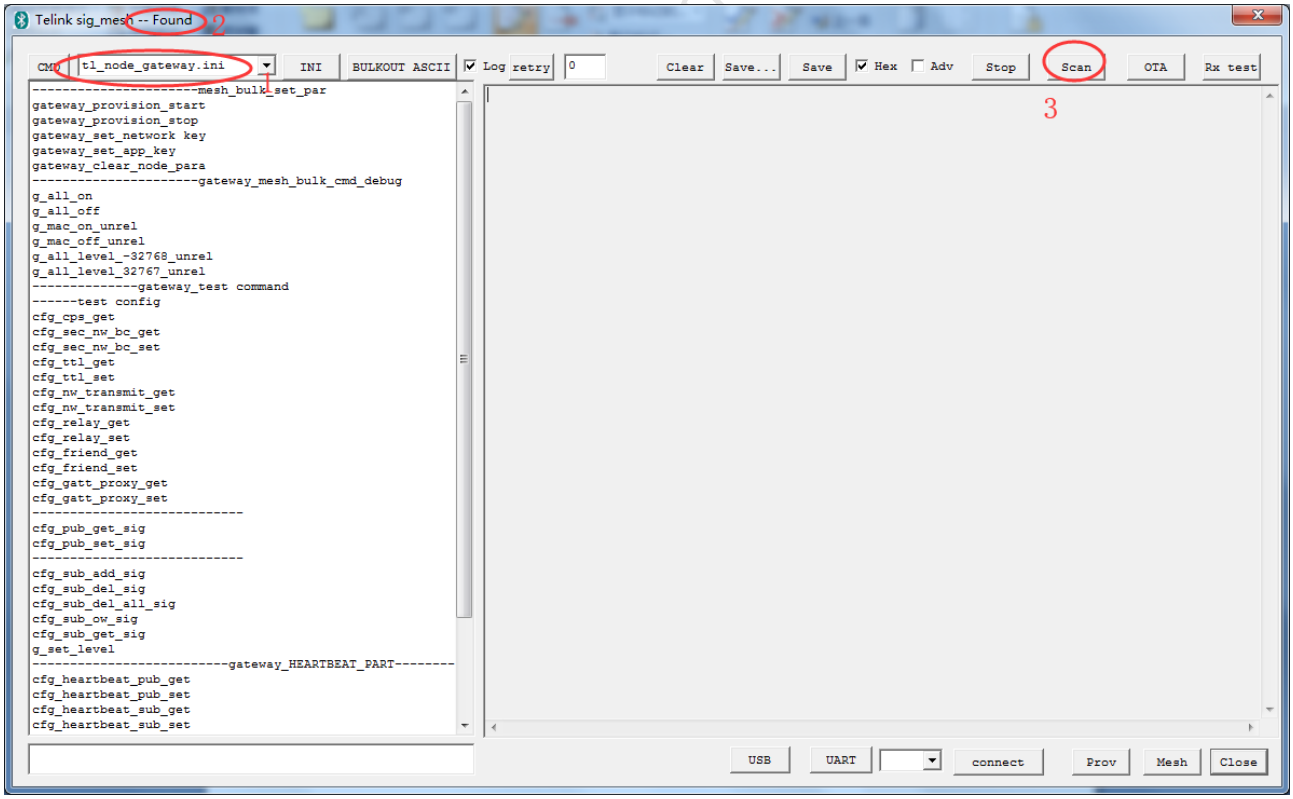


Figure 10.10: Add light via provisioner

3) After clicking the scan control button, the gateway will report the received unprovision beacon in the following format:

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

The scanned devices will be shown in the device list. Double click the target device which needs provision, the corresponding command is:

HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_ADV_FILTER+6 byte mac address

i.e.:e9 ff + 08 + mac(6 bytes).

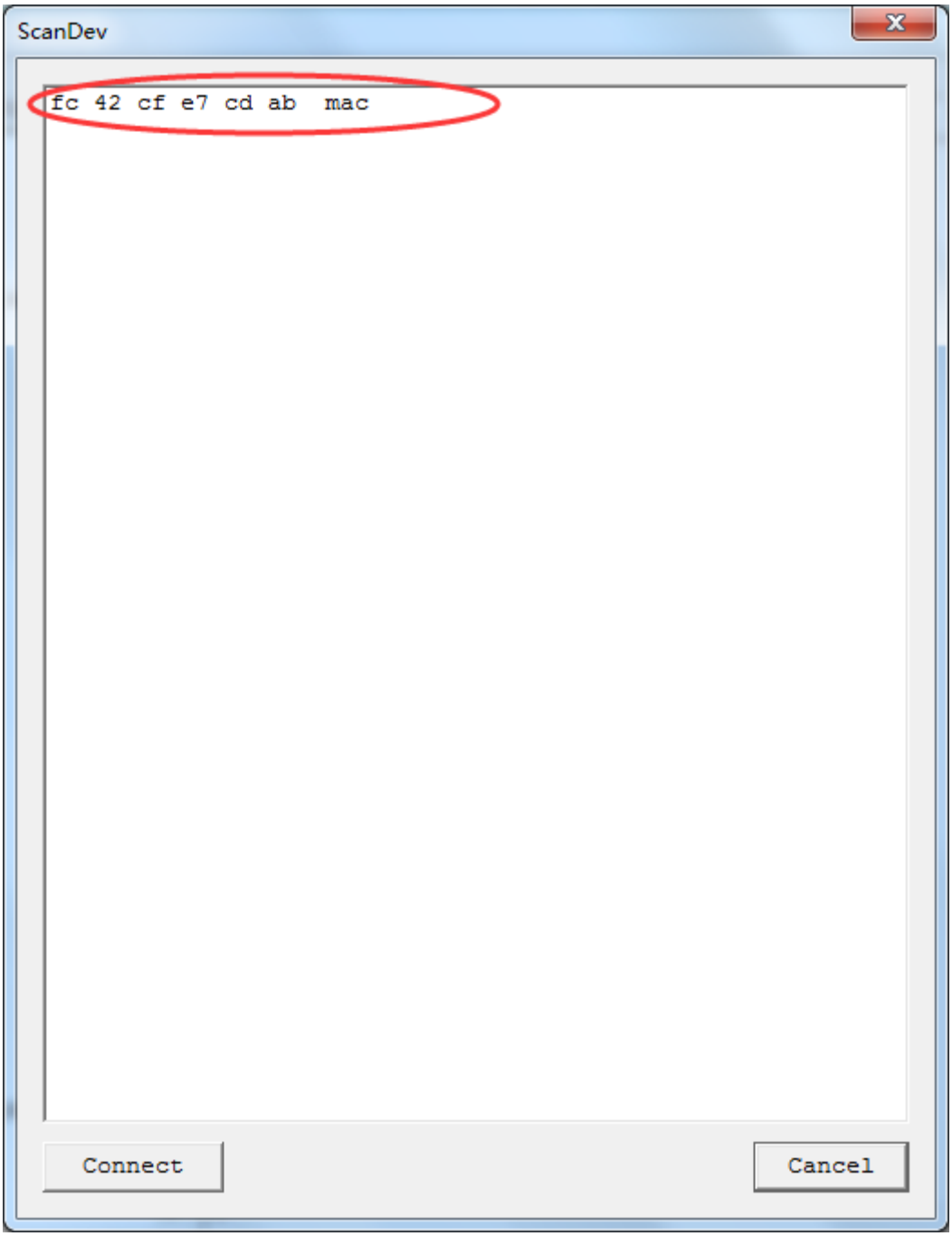


Figure 10.11: unprovision beacon

4) Click "Prov" to enter provision interface.

a. The corresponding command of the Provision control is: HCI_GATEWAY_CMD_GET_PRO_SELF_STS.

That is: e9 ff 0c.

b. After receiving the command, the gateway will return whether there is configuration information and the number of elements of the gateway. The corresponding command format is: TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_PRO_STS_RSP + provision_flag + pro_net_info.

That is: 91 8b + provision_flag + pro_net_info. pro_net_info is 25 bytes of provision data. The format is as follows:

```
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;
}provision_net_info_str;
```

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_ELE_CNT+total element: i.e., 91+8c+ total element.

c. If the provision flag is 0, it means that the gateway has no configuration information. The SetPro Internal control is enabled. Fill in the provision interface with relevant parameters of pro_net_info and click SetPro_interval to set the gateway configuration information. Then two commands will be issued automatically:

- HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_SET_PRO_PARA + pro_net_info (the first command)

i.e.,: e9 ff + 09 + pro_net_info

- HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_SET_DEV_KEY + unicast address + device key (the second command)

i.e.,: e9 ff + 0d +gateway address+device key

If the "SetPro Internal" button is disabled, it indicates the gateway has configured network parameters. Just skip parameter setting step.

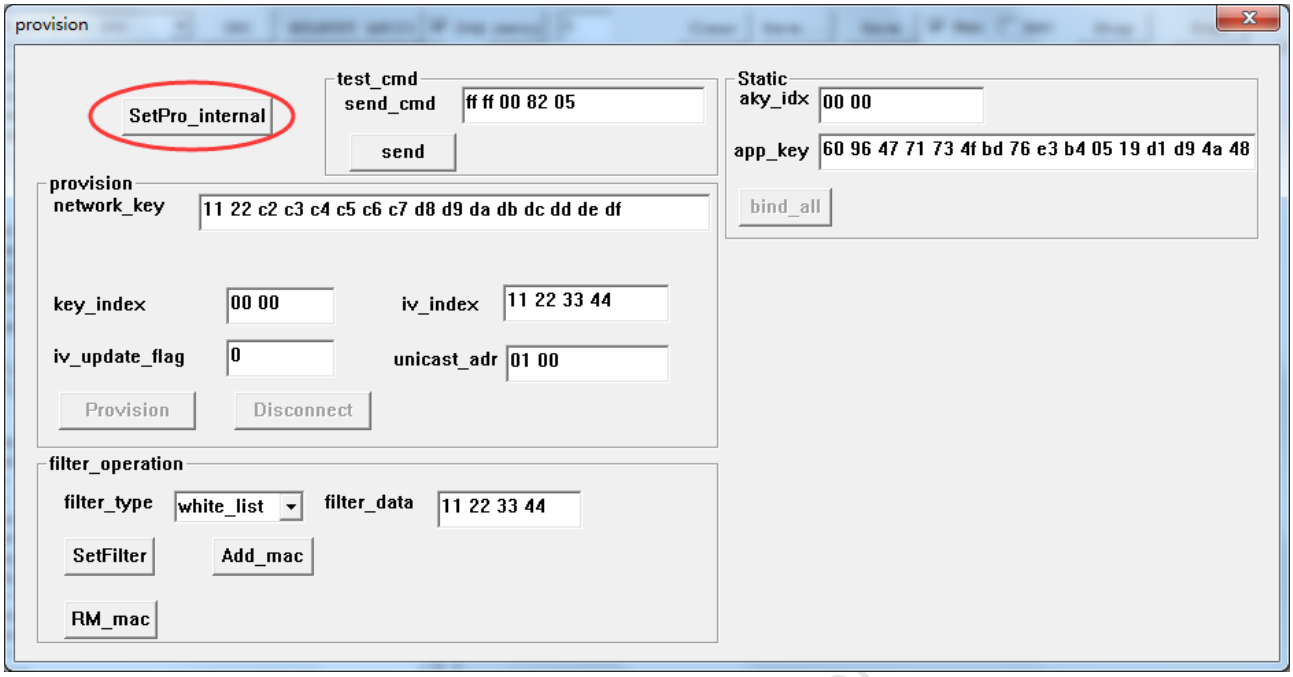


Figure 10.12: SetPro Internal

5) Click the "Provision" button to implement provision. During provision process, related log will be printed out and shown on the main interface. After successful provision, the "bind_all" button becomes enabled.

a. The corresponding command of the Provision control is: HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_NODE_PARA+ pro_net_info

i.e. e9 ff+0a+ ro_net_info.

b. During the Provision process, the allocated addresses are reported in the following format:

TSCRIPT_GATEWAY_DIR_RSP +HCI_GATEWAY_RSP_UNICAST+unicast addr,

i.e. 91+80+unicast address.

c. The node information will be reported after the provision is completed in the following format:

TSCRIPT_GATEWAY_DIR_RSP+ HCI_GATEWAY_CMD_SEND_NODE_INFO+ VC_node_info_t

i.e. 91+8d+ VC_node_info_t.

VC_node_info_t is defined as following:

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

d. The status of the provision will be reported after the provision is completed in the following format:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_PROVISION_EVT+ gateway_prov_event_t

i.e.: 91 + 89 + gateway_prov_event_t.

gateway_prov_event_t is defined as following:

```
typedef struct{
    u8 eve;//1 means success
    u16 adr;
    u8 mac[6];
    u8 uuid[16];
}gateway_prov_event_t;
```

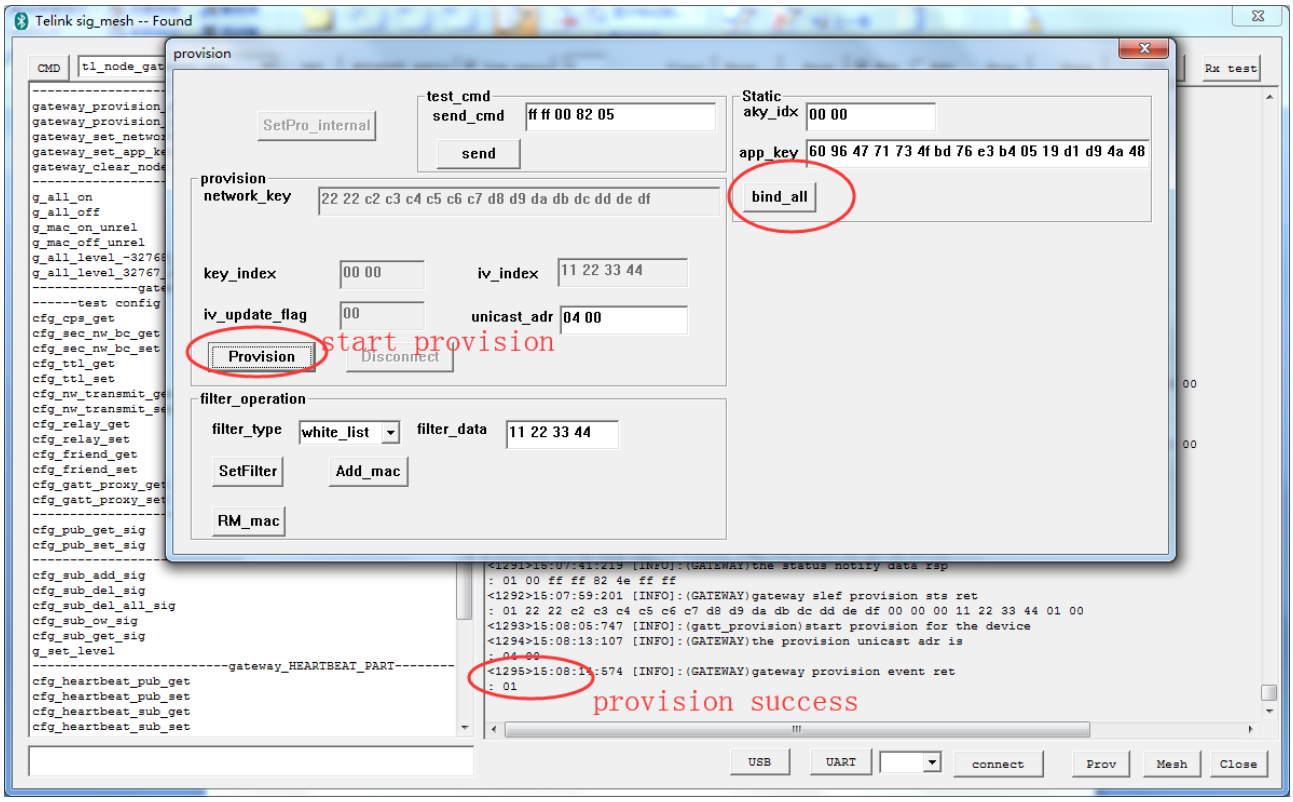


Figure 10.13: Provision

10.4.6 app_key binding

After provision is finished, it's also needed to bind the app_key for model by clicking the "bind_all" button.

a. The command corresponding to bind_all is: HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_START_KEYBIND + fast_bind ++app_key index(2 byte)+app_key(16 bytes).

i.e. e9 ff + 0b + fast_bind + app_key index(2 byte)+app_key(16 bytes).

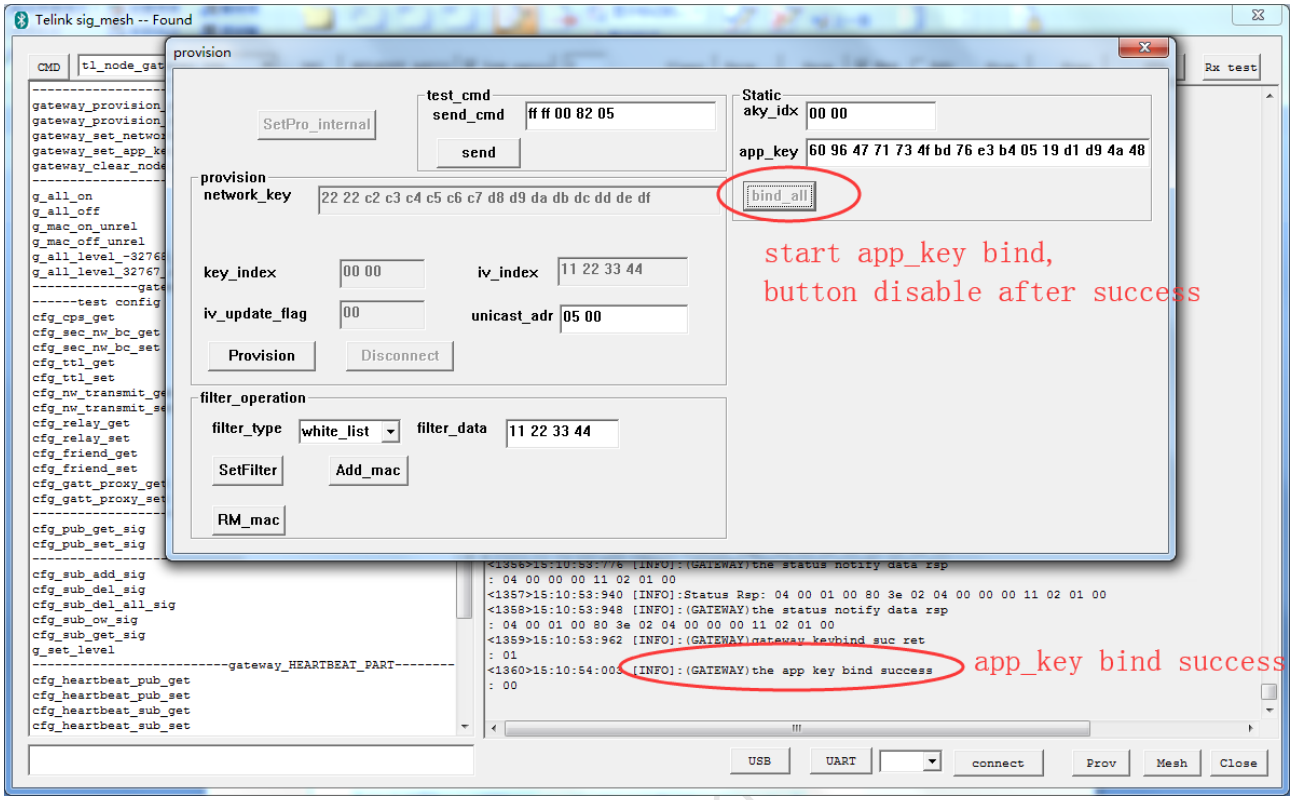


Figure 10.14: bind_all

When fast_bind is 1: the gateway will only send appkey add command. The provisioned device needs to enable the default binding function (PROVISION_FLOW_SIMPLE_EN is set to 1).

When fast_bind is 0: the gateway binds all model ids by default. To save time, users can choose the model ids to be bound. The gateway opens the macro MD_BIND_WHITE_LIST_EN. For the model ids to be bound, refer to the master_filter_list [] in the Mesh_common.c file. Users can modify it as needed.

b. During the App_key bind process, the gateway will call u8 gateway_model_cmd_rsp (u8 * para, u8 len) to return the status information of the bound model in the format: TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + parameter.

i.e.: 91 + 81 + appkey bind status

c. App_key bind will return HCI_GATEWAY_CMD_KEY_BIND_EVT after completion, indicating success or time_out. The format is:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_KEY_BIND_EVT +result

i.e.: 91 + 8a + result. (1:success 2:time_out)

10.4.7 Light on/off Control

After app_key binding, click "Mesh" to enter mesh interface, or directly double click the INI command "g_all_on/g_all_off" in the left command window of the main interface to implement on/off control.

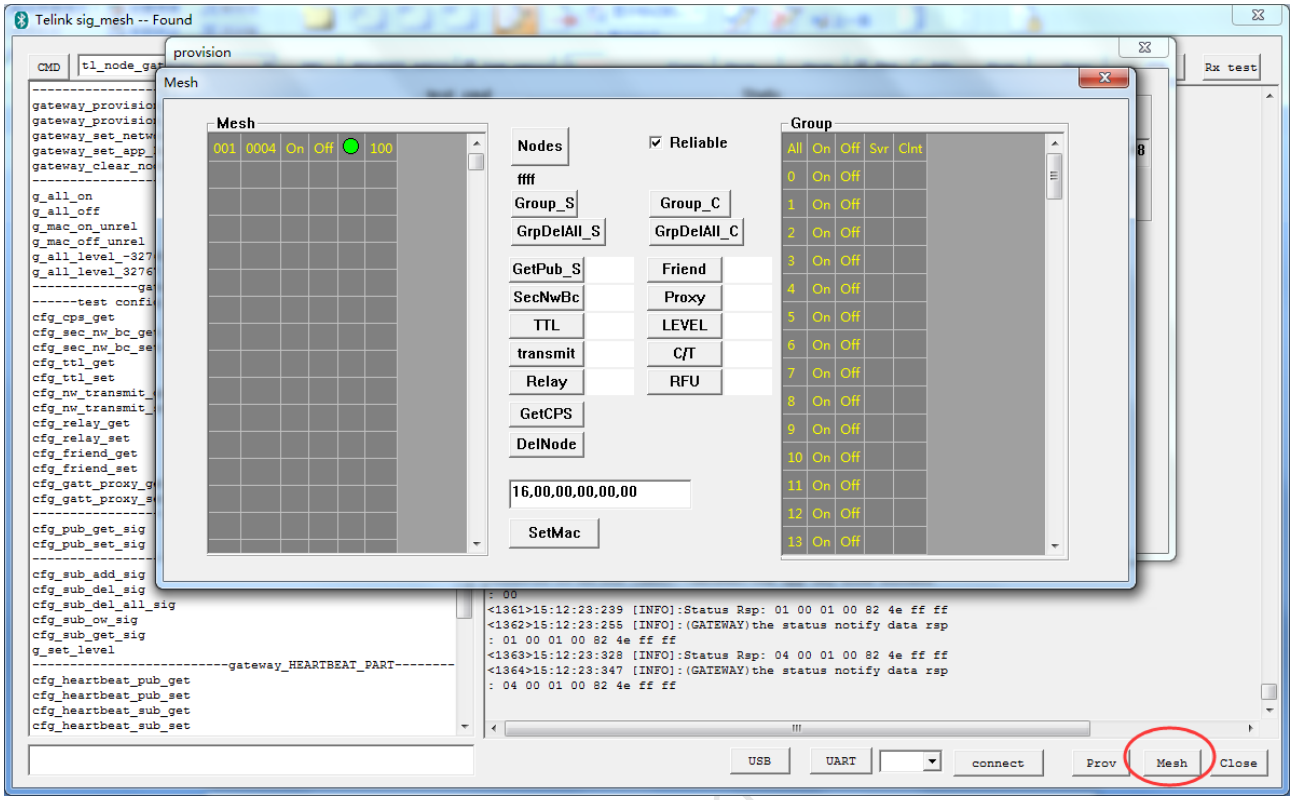


Figure 10.15: Light on/off control

After sending the onoff command, the corresponding node reports the status in the format:

```
Status Rsp_____________: 04 00 01 00 82 04 00 01 0a
```

The corresponding structure is:

```
typedef struct{
    u16 len; // length
    u16 src; // source address
    u16 dst; // destination address
    u8 data[ACCESS_WITH_MIC_LEN_MAX];   // access layer(op code, parameters)
}mesh_rc_rsp_t;
```

10.4.8 Provisioner Control Flow Chart

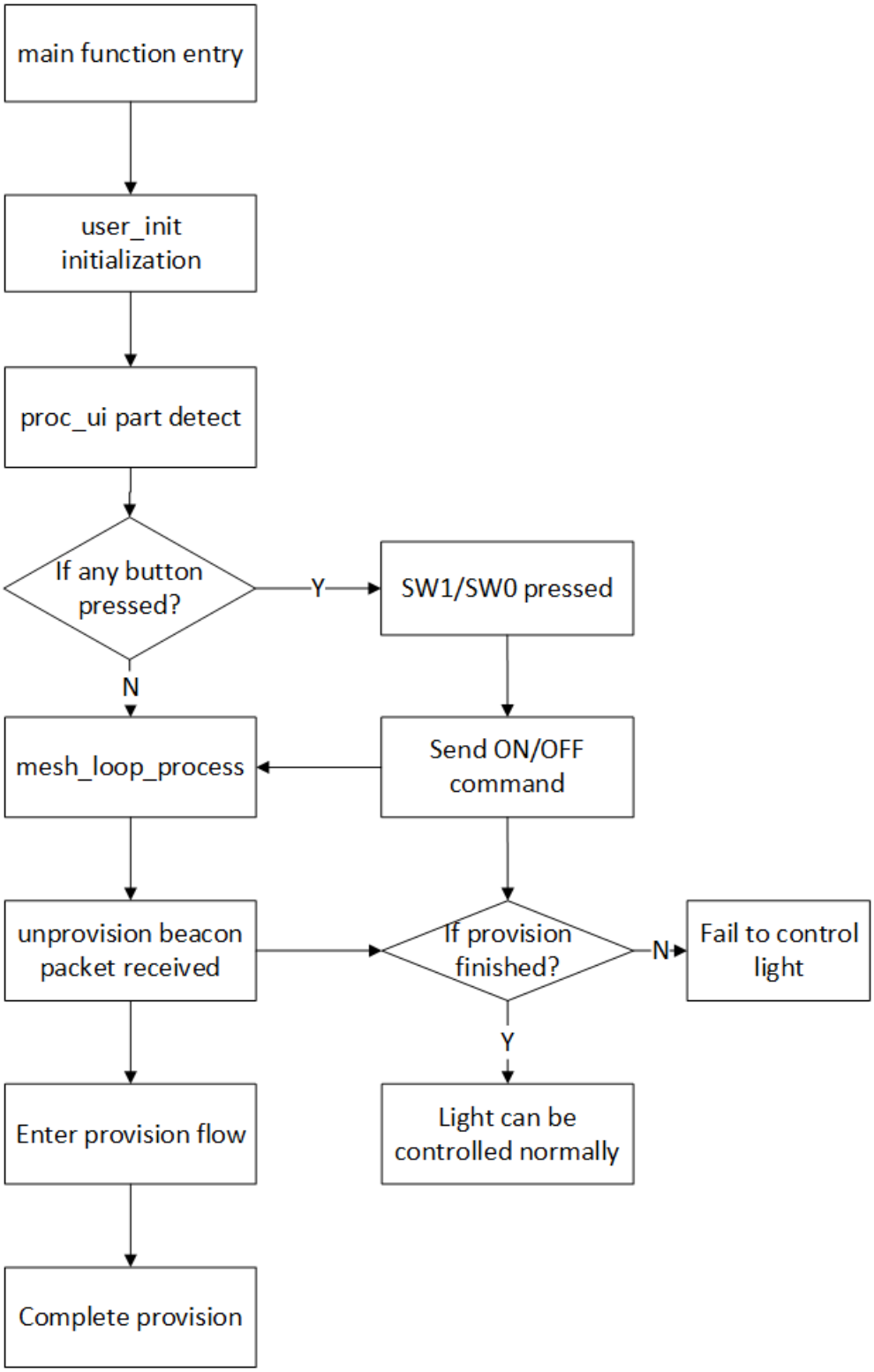


Figure 10.16: Provisioner Control Flow Chart

10.4.9 Smart Provision

One Click networking corresponds to Smart Provision inside the code.

The one-click networking function is based on the gateway project, and then remove sig_mesh_tool.exe tool on PC, only used the gateway dongle for networking. It is applicable to simple network scenarios, the later also do not need through the UI interface to the network nodes for too much configuration, mainly to use the commonly used control functions.

10.4.9.1 Difference between Smart Provision and Normal Networking

- Gateway normal networking mode: It requires a host computer or an app to perform networking, and after the networking is completed, various configurations can be performed on the network nodes through the host computer.
- Smart Provision mode: You can network without the host computer, need gateway dongle only.

In addition, in order to simplify the networking process and save networking time, the node side needs to open the PROVISION_FLOW_SIMPLE_EN, so that after the gateway dongle sends the App key add during networking, it ends the networking process and no longer sends the app key bind command, and the node to be networked automatically triggers the app key bind on itself after receiving the App key add.

If PROVISION_FLOW_SIMPLE_EN is not turned on at the node side, then it needs:

(1) Modify prov_uuid_fastbind_mode() to return 1 directly inside the function.

(2) Change GATEWAY_APPKEY_ADD_HEAD from

```
{(u8)HCI_CMD_GATEWAY_CTL, HCI_CMD_GATEWAY_CTL>>8, HCI_GATEWAY_CMD_START_KEYBIND, 1}
```

to

```
{(u8)HCI_CMD_GATEWAY_CTL, HCI_CMD_GATEWAY_CTL>>8, HCI_GATEWAY_CMD_START_KEYBIND, 0}
```

10.4.9.2 Principle Decription

The distribution process of one-key networking uses the standard distribution method of sig, only that the command interaction process of the host computer distribution is moved to the application layer of the dongle firmware at gateway side. After the provision start key is pressed, the main process mesh_smart_provision_proc() handles the provisioning status and simulates the host computer to push commands into the hci rx fifo. In the interface function gateway_common_cmd_rsp(), which is reported to the host computer, mesh_smart_provision_rsp_handle() is called to handle the message processing.

The initial iv index of the gateway after network allocation is SMART_IV_INDEX. Network key and app key are randomized values. See smart_gateway_provision_data_set() for details.

When networking, if you only want to add nodes that meet certain conditions, you can modify the filtering rules inside the function prov_uuid_fastbind_mode() in the HCI_GATEWAY_CMD_UPDATE_MAC branch of the mesh_smart_provision_rsp_handle() function.

10.4.9.3 Function Decription

The process of one-key distribution will be the same as normal networking, the gateway will send invite, start and other commands. For example: mesh_adv_prov_send_invite() sends invite command, mesh_adv_prov_send_start_cmd() sends start command.

10.4.9.4 Testing Process

(1) SDK settings

Gateway: need to turn on SMART_PROVISION_ENABLE

Node: need to turn on PROVISION_FLOW_SIMPLE_EN

If you do not want PROVISION_FLOW_SIMPLE_EN to be turned on at the node side, please refer to the introduction inside "Difference between Smart Provision and Normal Networking" to configure it.

(2) Initial networking

Gateway one-key network function, enable this function to burn 8258_mesh_gw.bin file to 8258 dongle and then press the key SW2, the gateway will automatically add the unallocated nodes within one-hop range to the network, within 30 seconds the unallocated nodes cannot be searched for to exit the process of network allocation, press SW1 to control the switching of nodes in the network.

(3) The steps for adding node again after the network has been established and running for a period of time.

Press the networking key SW2 again.

11 Mesh LPN Project Introduction

11.1 LPN Node and Implementation Method

Note: All figures in this section are derived from Sig Mesh spec.

11.1.1 LPN and friend

Low-Power feature: Rx side can run with obviously low duty cycle in mesh network. By enabling radio receiver only when necessary, the duty cycle is minimized to decrease node's power consumption. This is implemented by establishing friendship between LPN (Low Power Node) and FN (Friend Node).

A LPN can only establish friendship with a single FN, while a FN can establish friendship with multiple LPNs.

When a friendship is established, if the LPN node has previously established a friendship with another FN node, the LPN will inform the new FN node through the friend request command, and the FN node will call friend_cmd_send_clear() to send the clear command to notify the old FN node to clear the friendship with the LPN counterpart. The low-power node will poll (Poll) the friend node with a longer period, say 2 seconds (FRI_POLL_INTERVAL_MS) or longer. Check to see if there is a new message, and if so, get the message. After establishing the friendship, the LPN node reports the current subscription list (i.e., all group number information) to the FN node. Then if the FN receives a message whose destination address matches these group numbers or the LPN's element address, it caches the message, and then sends the cached message to the LPN when it receives a Poll command from the LPN. the FN and the LPN interact with each other with the Poll and update commands, which have iv index information in them, and can perform the iv update flow.

Friend feature: To help LPN running, the FN will store the information to be sent to the LPN, and only initiate transmission when there's obvious request from the LPN.

11.1.2 Friendship Parameters

LPN needs to find FN and initiate a "Friendship Establish" process to establish friendship with it. Following shows some key parameters which are configured during the "Friendship Establish" process and serve to manage LPN behavior.

1) ReceiveDelay is the time that elapses between when the LPN sends a request to the buddy node and when it starts listening to the response. This gives the friend node time to prepare the response and send it back. Specified by the LPN through this macro FRI_REC_DELAY_MS and communicated to the FN through the friend request command.
2) ReceiveWindow is the timing used by the LPN to listen for responses. Specified by the FN through this macro FRI_REC_WIN_MS and communicated to the LPN through the friend offer command. the following figure depicts the timing involving ReceiveDelay and ReceiveWindow.

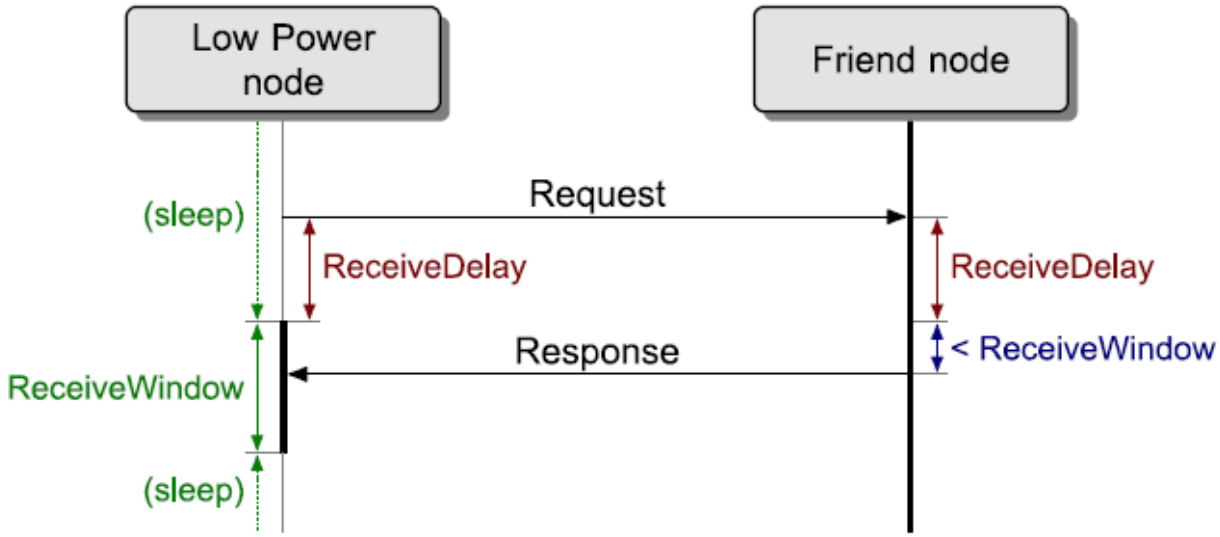


Figure 11.1: Timing Sequence of ReceiveDelay and ReceiveWindow

3) PollTimeout sets the maximum time that may elapse between two consecutive requests sent by the LPN to its friend node. It is specified by the LPN through this macro LPN_POLL_TIMEOUT_100MS and is communicated to the FN through the friend request command. If the friend node fails to receive a request from the LPN before the PollTimeout timer expires, the friendship relationship will be terminated.

11.1.3 Establish Friendship

The process to establish friendship in BT mesh network is shown as below:

Step 1 LPN issues a "Friend Request" message which does not support relaying. Only the FN within the direct radio range will process this message, and other nodes without "friend" features will discard this message. The "Friend Request" message contains parameters of LPN, including "ReceiveDelay", "ReceiveWindow" and "PollTimeout".

Step 2 If a FN nearby supports specific requirement in the "Friend Request" message, it will prepare a "Friend Offer" message and send it back to the LPN. This message contains various parameters, including supported ReceiveWindow size, available message queue size, available subscription list size, and RSSI value measured by the FN.

Step 3 When the LPN receives the "Friend Offer" message, it will adopt a specific algorithm to select suitable FN. This accurate algorithm may take various cases into consideration: Some device may give priority to the ReceiveWindow size, so as to minimize power consumption; some device may pay more attention to the RSSI value, so as to ensure high-quality link with FN. It depends on product developer.

Step 4 The LPN will send a "Friend Poll" message to the selected FN.

Step 5 After the "Friend Poll" message from the LPN is received, the FN will respond with a "Friend Update" message to finish "Friendship Establish" process and supply security parameters.

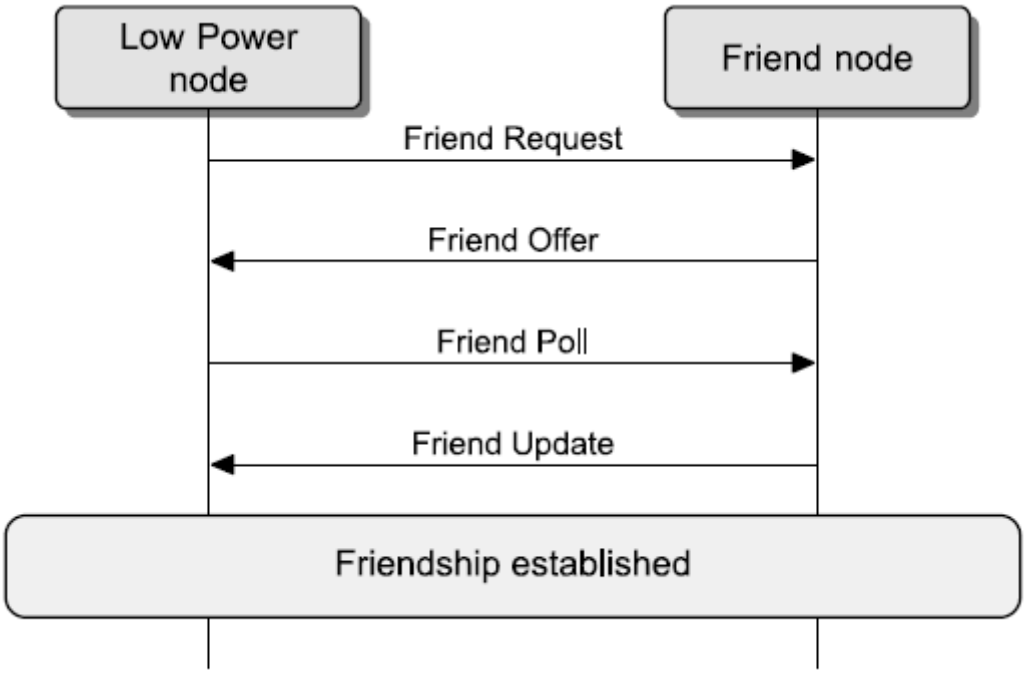


Figure 11.2: Establish friendship

11.1.4 Friendship Message Exchange

After friendship is established, the FN will store all messages of the LPN in the "Friend Queue". These messages are so-called "stored message". The figure below shows message exchange between the FN and the associated LPN.

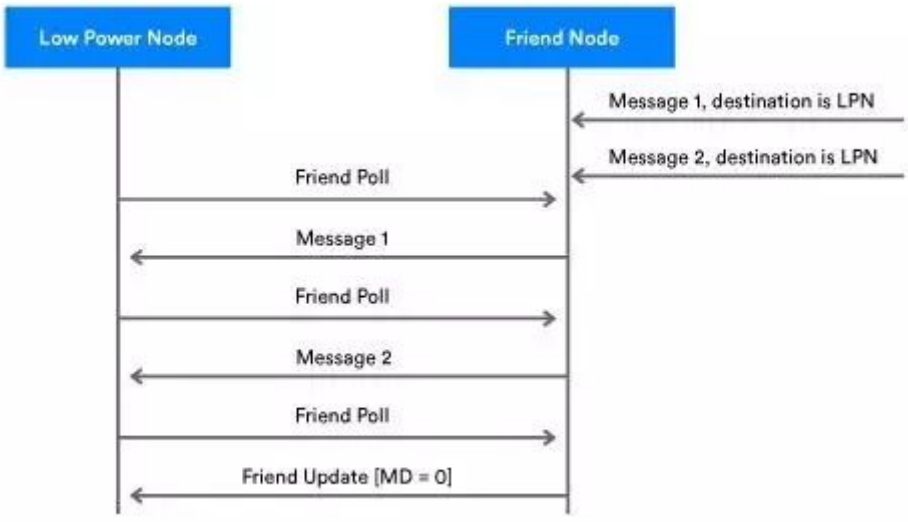


Figure 11.3: Friendship Message Exchange

When the FN receives a message from the LPN addressing to this node, the FN will buffer this message by

storing it in the "Friend Queue" area. As shown in the figure above, the FN stores "Message 1" and "Message 2" for the LPN.

The LPN will periodically enable its transceiver, and send "Friend Poll" message to the FN so as to check whether there's any stored message buffered for the LPN.

The FN will first send a stored message to the LPN as the response to the "Friend Poll".

After each reception of message from the FN, the LPN will continue to send "Friend Poll" message until it receives a "Friend Update" message with the "MD (More Data)" field set as "0". "MD=0" means there's no more message buffered in the FN for the LPN. Then the LPN stops the polling to the FN.

11.1.5 Security

Master Security Material: It's derived from network key (NetKey), and it can be used by other nodes within the same network. Message encrypted by using "Master Security Material" can be decoded by any node within the same network.

Friend Security Material: It's derived from network key (NetKey), as well as extra counter number generated by the LPN and FN. Message encrypted by using "Friend Security Material" can only be decoded by the LPN and the FN processing this message.

Friend messages encrypted by using "Friend Security Material" include: "Friend Poll", "Friend Update", and "Friend Subscription List".

Friend messages encrypted by using "Master Security Material" include: "Friend Clear" and "Friend Clear Confirm".

Any other non-control message from the LPN to the FN will set the "credential_flag" in corresponding model publish parameter as needed, so as to determine whether the encryption method is "Master Security Material" or "Friend Security Material". The default value of the "credential_flag" is 0, corresponding to "Master Security Material" encryption.

11.1.6 Friendship Termination

If the FN fails to receive a "Friend Poll", "Friend Subscription List Add" or "Friend Subscription List Delete" message before the "PollTimeout" expires, the friendship between the FN and the LPN is terminated.

The LPN can initiate friendship termination program by sending a "Friend Clear" message to the FN, so that the FN will terminate their friendship.

11.2 Friendship Sleep and Working Mechanism

11.2.1 FN Receive Packet Processing Interface

```
void mesh_friend_ship_proc_FN(u8 *bear)
```

- Where bear is not empty, it indicates that a friendship-related command was received.

- When bear is empty, it means it is a polling call inside main_loop() to detect and handle timing events and timeout events.

```
void mesh_friend_ship_proc_FN(u8 *bear)
{
    foreach(i,g_max_lpn_num){ // a friend node may establish friendship with many LPN, so check all LPN.
        mesh_fri_ship_proc_fn_t *proc_fn = &fri_ship_proc_fn[i];
        if(!bear){
            if(proc_fn->status){ // (FRI_ST_IDLE != proc_fn->status)
                if(FRI_ST_OFFER == proc_fn->status){
                    if(clock_time_exceed(proc_fn->offer_tick, proc_fn->offer_delay*1000)){
                        ......
                        // send friend offer and set to state of receiving friend poll after received friend request.
                        friend_cmd_send_fn(i, CMD_CTL_OFFER);
                        ......
                        mesh_friend_ship_set_st_fn(i, FRI_ST_POLL);
                    }
                }else if(FRI_ST_POLL == proc_fn->status){
                    // add 500ms, because handling response of POLL was delay some ten ms.
                    if(clock_time_exceed(proc_fn->offer_tick,
                     (500+FRI_ESTABLISH_OFFER_MS)*1000)){
                        // timeout to receive friend poll from LPN after send friend offer to LPN,
                        // means that LPN did not receive offer, or LPN did not select current FN as friend node,
                        // or FN did not receive the friend poll from LPN.
                        mesh_friend_ship_proc_init_fn(i);
                    }
                }else if(FRI_ST_TIMEOUT_CHECK == proc_fn->status){
                    if(clock_time_exceed_100ms(proc_fn->poll_tick, (u32)
                     (fn_req[i].PollTimeout))){
                        // timeout to receive friend poll from LPN, then will disconnect this friendship.
                        friend_ship_disconnect_fn(i, FS_DISCONNECT_TYPE_POLL_TIMEOUT);
                    }
                }
            }

            if(proc_fn->clear_poll){    // clear by other FN
                if(clock_time_exceed_100ms(proc_fn->clear_start_tick, (u32)
                 (fn_req[i].PollTimeout)*2)){
                    // when the timeout expires, even if the clear response has not been received yet, the clear command will stop being sent.
                    mesh_stop_clear_cmd(i);
                }else{
```

```
                    if(clock_time_exceed_100ms(proc_fn->clear_cmd_tick,
                     ↪ proc_fn->clear_int_100ms)){
                        ......
                        // Gradually reduce the frequency of sending clear commands.
                        // please refer to mesh V1.1 spec "Figure 3.24: Friend Clear procedure
                         ↪ example" of "3.6.6.3.1 Friend establishment".
                        friend_cmd_send_fn(i, CMD_CTL_CLEAR);
                    }
                }
            }

            if(proc_fn->clear_by_lpn_tick && clock_time_exceed(proc_fn->clear_by_lpn_tick,
             ↪ 5*1000*1000)){
                // when received friend clear, should not clear at once, and need to delay some
                 ↪ time to clear Friendship.
                // because LPN may retry sending friend clear command when not receive clear
                 ↪ confirm.
                friend_ship_disconnect_fn(i, FS_DISCONNECT_TYPE_CLEAR);
            }
        }else{
            ...... // to process packet received
        }
    }
}
```

11.2.2 Processing Interface for Packets Sent by FN to LPN

```
mesh_friend_response_delay_proc_fn()
```

When FN needs to send packet to LPN, for example, send friend update, when poll delay reaches the end, it needs to send the packet as soon as possible, instead of waiting for the adv interval (10ms) set by bls_ll_setAdvParam() to reach like other ordinary network PDUs, then send the packet. So we poll the tick with mesh_friend_response_delay_proc_fn(), and when the time is up, call fn_quick_send_adv() to send the packet immediately. In addition, since the message sent to the LPN needs to be checked again to see if the message needs to be updated, such as whether the segment block ack needs to be updated, etc. (refer to the processing of get_cache_buf_for_poll()), it needs to wait until the update is done, and then perform the encryption of the network layer before sending. See the handling of mesh_friend_response_delay_proc_fn() below for details:

```
void mesh_friend_response_delay_proc_fn(u8 lpn_idx)
{
    fn_ctl_rsp_delay_t *p_delay = &fn_ctl_rsp_delay[lpn_idx];
    int print_cache_flag = 0;
```

```
    if(p_delay->delay_type && clock_time_exceed(p_delay->tick, fn_req[lpn_idx].RecDelay * 1000 -
    ↪ 1800)){    // 1800us: encryption pkt time
        if(DELAY_POLL == p_delay->delay_type){
            if(p_delay->poll_rsp){
                if(fn_other_par[lpn_idx].cache_overwrite){
                    p_delay->poll_rsp = get_cache_buf_for_poll(lpn_idx, 1, 1); //
↪   cache_overwrite will be clear inside.
                }

                ......
                if(bear_tx_len <= MESH_BEAR_SIZE){
                    ......
                    // no encryption before, because need to check buffer in
                     ↪  mesh_fri_cmd2cache_(), then to set cache_overwrite or not.
                    mesh_sec_msg_enc_nw_rf_buf((u8 *)(&bear_temp->nw),
↪   mesh_lt_len_get_by_bear(bear_temp), FRIENDSHIP,
↪   lpn_idx,0,fn_other_par[lpn_idx].nk_sel_dec_fn, 0);
                    ......
                    mesh_tx_cmd_add_packet_fn2lpn((u8 *)bear_temp);
                }
                ......
            }

            mesh_fri_ship_proc_fn_t *proc_fn = &fri_ship_proc_fn[lpn_idx];
            if(proc_fn->clear_delay_cnt){
                proc_fn->clear_delay_cnt--;
                if(0 == proc_fn->clear_delay_cnt){ // make sure establish friendship success
                    friend_cmd_send_fn(lpn_idx, CMD_CTL_CLEAR); // use normal fifo, not
↪   mesh_adv_fifo_fn2lpn_
                    ......
                }
            }
        }else if(DELAY_SUBSC_LIST == p_delay->delay_type){
            friend_cmd_send_subsc_conf(p_delay->adr_dst, (u8)p_delay->par_val);
        }else if(DELAY_CLEAR_CONF == p_delay->delay_type){
            ......
            friend_cmd_send_clear_conf(clear.LPNAdr, (u8 *)&clear,
↪   sizeof(mesh_ctl_fri_clear_t));
        }

        p_delay->delay_type = 0;
    }

    if(my_fifo_data_cnt_get(&mesh_adv_fifo_fn2lpn)){
        fn_quick_send_adv();    // "poll rsp" may be delay when in BLE_S window, so quickly send
↪   here again. and also "send_subsc_conf /send_clear_conf" need quick send.
```

```
    }
}
```

11.2.3 LPN Packet Processing Interface

```
void mesh_friend_ship_proc_LPN(u8 *bear)
```

- Where bear is not empty, it indicates that a friendship-related command was received.
- When bear is empty, it means it is a polling call inside main_loop() to detect timing events.

```
void mesh_friend_ship_proc_LPN(u8 *bear)
{
    ......
    if(!bear && is_mesh_adv_cmd_fifo_empty()){
        if(fri_ship_proc_lpn.poll_retry && clock_time_exceed(fri_ship_proc_lpn.poll_tick,
        ↪ poll_retry_interval_ms*1000)){
            fri_ship_proc_lpn.poll_retry--;
            if(0 == fri_ship_proc_lpn.poll_retry){
                ...... // Logic for handling FN replies that are not received after the poll has
↪   been sent and the time limit has expired
            }
        }
        else if(subsc_list_retry.retry_cnt && clock_time_exceed(subsc_list_retry.tick,
        ↪ timeout_ms)){
            subsc_list_retry.tick = clock_time();   // also refresh when send_subsc
            subsc_list_retry.retry_cnt--;
            ...... // Logic for handling when the LPN reports the subscription list after the
↪   friendship has just been successfully established and no response is received from the FN
↪   after the timeout period.
        }
    }

    mesh_cmd_bear_t *p_bear = (mesh_cmd_bear_t *)bear;
    //mesh_cmd_nw_t *p_nw = &p_bear->nw;
    mesh_cmd_lt_ctl_unseg_t *p_lt_ctl_unseg = &p_bear->lt_ctl_unseg;
    u8 op = -1;
    if(bear){
        op = p_lt_ctl_unseg->opcode;
    }

    if(0 == fri_ship_proc_lpn.status){ // LPN Processing branch after a friendship has been
    ↪ successfully established, or before a friend request has been sent.
        if(bear){
            if(CMD_CTL_SUBS_LIST_CONF == op){
```

```
                ..... // After sending Friend Subscription List Add, the processing branch of
↪  Friend Subscription List Confirm is received.
                      // See 3.6.5.7 Friend Subscription List Add for details.
            }else if(CMD_CTL_UPDATE == op){
                .....
                // Receives the Friend update processing branch. Includes processing of the iv
                ↪  index, etc.
                iv_update_key_refresh_rx_handle(&p_update->flag, p_update->IVIndex);
            }
        }else{
            if(is_friend_ship_link_ok_lpn() && is_mesh_adv_cmd_fifo_empty() &&
            ↪  clock_time_exceed(fri_ship_proc_lpn.poll_tick, get_lpn_poll_interval_ms() *
            ↪  1000)){
                // When the LPN doesn't need to sleep at a certain time, then it can't execute
                ↪  the event of sending a friend poll periodically via
                ↪  mesh_friend_ship_start_poll() inside user_init_deepRetn(). So here we add
                ↪  the handling of checking again if we need to send a friend poll or not. If
                ↪  there is a retention wakeup, then the processing here is not executed.
                mesh_friend_ship_start_poll();
            }
        }
    }else{
        switch(fri_ship_proc_lpn.status){   // Be true only during establishing friendship.
            case FRI_ST_REQUEST:
                if(is_mesh_adv_cmd_fifo_empty() && clock_time_exceed(fri_ship_proc_lpn.req_tick,
                ↪  FRI_REQ_TIMEOUT_MS * 1000)){
                    ......
                    friend_cmd_send_request();
                    ......
                    mesh_friend_ship_set_st_lpn(FRI_ST_OFFER); // After sending the request, it
↪  enters the state of waiting to receive the offer.
                }
                break;
            case FRI_ST_OFFER:
                if(bear){
                    if(CMD_CTL_OFFER == p_lt_ctl_unseg->opcode){
                        if(0 != lpn_rx_offer_handle(bear)){ // Includes a comparison to select
                        ↪  an optimal FN
                            break;
                        }
                    }
                }else{
                    if(clock_time_exceed(fri_ship_proc_lpn.req_tick,
                    ↪  FRI_ESTABLISH_PERIOD_MS*1000)){
                        if(mesh_lpn_par.FriAdr){
                            mesh_lpn_par.link_ok = 1;
```

```
                        mesh_friend_key_update_all_nk(0, 0); // After 1 second, determine
↪ the best FN and then update the corresponding friend key
                        ......
                    }
                    mesh_friend_ship_set_st_lpn(FRI_ST_POLL);// Enter the state of sending
↪ Friend Poll
                }
            }
            break;
        case FRI_ST_POLL:
            if(is_friend_ship_link_ok_lpn()){
                if(is_mesh_adv_cmd_fifo_empty()){
                    mesh_lpn_par.poll.FSN = 0;   // init
                    // send poll
                    fri_ship_proc_lpn.poll_retry = FRI_GET_UPDATE_RETRY_MAX + 1;
↪
                    friend_cmd_send_poll(); // Press the Friend poll into the send packet
↪ fifo, checking at the top of mesh_friend_ship_proc_LPN() when the time is up before sending
↪ the packet
                    t_rec_delay_and_win = mesh_lpn_par.req.RecDelay +
↪ mesh_lpn_par.offer.RecWin;
                    mesh_friend_ship_set_st_lpn(FRI_ST_UPDATE);// Go to Waiting to receive
↪ friend update
                }
            }else{
                lpn_no_offer_handle(); // Check that if no offer is received during the
↪ Waiting to Receive Offers phase, the friendship creation fails and a resend of the Friend
↪ request is initiated
            }
            break;
        case FRI_ST_UPDATE:
            if(bear){   // current state is establishing friendship
                if(CMD_CTL_UPDATE == p_lt_ctl_unseg->opcode){
                    // Friend update received, Friendship creation complete.
                    //friendship establish done
                    mesh_lpn_par.req.PreAdr = mesh_lpn_par.FriAdr;
                    iv_update_key_refresh_rx_handle(&p_update->flag, p_update->IVIndex);
                    mesh_friend_ship_proc_init_lpn();

                    friend_ship_establish_ok_cb_lpn();
                }
            }else{
                if(clock_time_exceed(fri_ship_proc_lpn.poll_tick,
                 ↪ t_rec_delay_and_win*1000)){
                    // If no Friend update is received after the timeout period, return to
                     ↪ the FRI_ST_POLL phase and resend the Friend Poll.
```

```
                    mesh_friend_ship_retry();
                }
            }
            break;
        default:
            break;
    }
  }
}
```

11.2.4 FriendShip Sleep Mechanism

Timed events, including timed wake-up packets, are based on the soft timer mechanism. For soft timer related content, please refer to this section "Application of Soft Timer".

The mesh_lpn_adv_interval_update() refreshes the broadcast (wake-up) interval of the LPN according to the different states of the LPN, thus changing the interval of the friend request/poll commands.

11.2.5 Friendship Working Mechanism

The LPN node enables the low power management mechanism of BLE by turning on BLE_REMOTE_PM_ENABLE. The details of this mechanism can be found in the BLE handbook, such as "AN-21112301-C_Telink B85m BLE Single Connection SDK", "Developer Handbook.pdf" in the "Low Power Management (PM)" section. In short, the mechanism is realized by soft timer:

- Inside the user init, the sleep management module is registered with blc_ll_initPowerManagement_module(), including ll_module_pm_cb, etc.
- In ADV state, ADV interval is defined, and soft timer realizes to send broadcast packet once per interval, and then main loop executes to sleep management unit ll_module_pm_cb() in blt_sdk_main_loop(), then soft timer sets the next wakeup time point according to ADV interval, and then enters into sleep. Then when the time is up, MCU wakes up, executes user_init_deepRetn(), and sends the next broadcast packet ……
- In the GATT connected state, the interval becomes the connected interval; the other mechanisms are the same.

The working mechanism of LPN is as follows:

(1) At the beginning, it is in un-networked state, user_init() –> user_init_peripheral –> mesh_lpn_adv_interval_update() will wake up and send the connectable broadcast packet periodically with the interval of the connectable broadcast packet as the soft timer event. By default, PB-ADV and PB-GATT are supported, so inside user_init_peripheral(), judge and call bls_pm_setSuspendMask (SUSPEND_DISABLE) to turn off the Sleep mechanism if it is in un-networked state.

(2) After the lpn node allocates the netkey and other information in the network, the provisioner starts the key bind process. Since the time of the key bind process is uncertain, the LPN is judged by the mesh_lpn_state_proc(), and when no key bind command is received for 3 seconds (LPN_START_REQUEST_AFTER_BIND_MS), the entire provisioning process is considered to have been

completed. Then it calls the mesh_friend_ship_set_st_lpn(FRI_ST_REQUEST) interface to enter the FRI_ST_REQUESTt state, and call mesh_friend_ship_set_st_lpn() inside mesh_friend_ship_set_st_lpn() to set the interval of the soft timer periodic event to FRI_REQ_TIMEOUT_MS.

```
void mesh_lpn_state_proc()
{
......
    if(lpn_provision_ok){
        ......
    }else{
        if(!is_provision_success()){
            ......
        }else{
            if((!lpn_provision_ok) && node_binding_tick && clock_time_exceed(node_binding_tick,
             ↪  LPN_START_REQUEST_AFTER_BIND_MS*1000)){
                lpn_provision_ok = 1;// provison and key bind finish
                gatt_adv_send_flag = GATT_LPN_EN;
                mesh_friend_ship_set_st_lpn(FRI_ST_REQUEST);
                if(BLS_LINK_STATE_CONN == blt_state){
                    bls_ll_terminateConnection(0x13); // disconnect to establish friendship
                }
            }
        }
    }
......
}
```

Or, after powering down and re-powering up, inside proc_ui() call mesh_friend_ship_set_st_lpn(FRI_ST_REQUEST).

(3) After that, send Friend Request in mesh_friend_ship_proc_LPN() to enter the friendship creation process.

Note: For every mesh message sent by LPN, it will call mesh_lpn_sleep_prepare(u16 op) function to set the PM and update the callback function and time point for the next task via soft timer.

The function friend_cmd_send_request() sends the Friend Request by executing mesh_lpn_sleep_prepare() to set the next wakeup point after FRI_ESTABLISH_REC_DELAY_MS and then lpn_quick_tx() to send the packet immediately.

(4) After sending friend request, it will wait for FRI_ESTABLISH_PERIOD_MS (default is 1.1 seconds), within 1.1 seconds, if it doesn't receive any friend offer, the MCU will set the next wakeup time point according to the soft timer in blt_sdk_main_loop(), and then go to sleep in the sleep management unit ll_module_pm_cb(). The MCU will set the next wakeup time according to the soft timer, and then go to sleep. When the time is up, it will wake up and continue to send friend request.

(5) If a friend offer is received, the process of packet receipt processing and establishing a Friendship is performed, as described in lpn packet processing interface.

(6) When the Friendship is established, mesh_lpn_adv_interval_update() is executed to update the base wakeup period to the poll interval, in addition to this base wakeup period, there are also timer events that are added by the call to mesh_lpn_sleep_prepare() for each packet sending event, and so on.

(7) Sending a friend poll is triggered by mesh_friend_ship_start_poll(). It is currently called in three places:

- user_init_deepRetn()->mesh_friend_ship_start_poll() This is a normal send, i.e., every time the poll interval wakes up, it will be called again to send the poll periodically.
- mesh_lpn_poll_md_wakeup()->mesh_friend_ship_start_poll() This is called when it is detected that the Friend Node's cache still has data to be fetched.
- mesh_friend_ship_proc_LPN()->mesh_friend_ship_start_poll() This is only triggered in special cases. I.e., if you don't enter sleep at a certain time, there is no way to trigger sending a poll via user_init_deepRetn()->mesh_friend_ship_start_poll(), so it is triggered here.

11.2.6 Mechanism for LPN to Receive a Destination Address as a Group Number

- Each time a friendship is created, the LPN sends the subscription list add command (CMD_CTL_SUBS_LIST_ADD).
- The FN node stores the group number list when it receives it, see the processing of friend_subsc_list_add_adr() for more details
- Subsequently, when the FN receives commands from other nodes with a destination address that matches the group number in the group number list, it helps the LPN to cache the information and sends it to the LPN for processing when it receives the poll command from the LPN. When testing, configure a group number for the LPN. Then, every time the LPN creates a friendship, the LPN will automatically issue CMD_CTL_SUBS_LIST_ADD.

11.3 Common Parameter Configuration for LPN

FN stands for Friend Node and LPN stands for Low Power Node for the following contents.

11.3.1 Friend Node

- FN_CACHE_SIZE_LOG: The maximum number of messages to be cached for LPN is FN_CACHE_SIZE_LOG times 2.
- FRI_REC_WIN_MS: The minimum reception window required by FN for LPN, default is 20ms. It indicates the time to listen to the broadcast packet after LPN sends Poll, if timeout occurs, it means that Friend node's reply is not received. Then the Poll command will be retransmitted.FRI_REC_WIN_MS cannot be set too small because there are 3 channels for broadcast packet sending and the possibility that the FN is dealing with something else with higher priority, resulting in the timing of the FN's reply to the LPN not being as precise as it should be.

11.3.2 Low Power Node

- FRI_REQ_TIMEOUT_MS: Configure the interval for sending friend request. The default is 2 seconds. If the product definition requires lower power consumption, it can be increased according to the actual situation.

- FRI_ESTABLISH_WIN_MS：the maximum time to wait for receiving Friend offer after sending Friend offer. The spec specifies that the time is 1 second, because we want to receive offers from as many FNs as possible, and then choose the best FN. Generally it is not recommended to change this value. However, if the product requires very low power consumption, and only modifying FRI_REQ_TIMEOUT_MS can not meet the demand, then we can consider changing FRI_ESTABLISH_WIN_MS to a smaller value.
- FRI_POLL_INTERVAL_MS：interval of friend poll. The default is 2 seconds, which is a short time, mainly because it is used for single fire switch low power devices, and the command response time can not be too long. If the product definition requires low power consumption, it can be changed according to the actual situation.
- FRI_POLL_RETRY_MAX： LPN does not receive any reply from FN after sending Poll command, when the number of times exceeds this value, LPN will flip the value of FSN in the poll once, and then send the Poll again, if it still doesn't receive any reply from FN for the consecutive FRI_POLL_RETRY_MAX times, LPN will consider that FN is offline. At this point, LPN will disconnect the current friendship and start to send friend request to try to establish friendship with other friend node.
- LPN_SCAN_PROVISION_START_TIMEOUT_MS：It means that after LPN sends a friend request, no offer has been received from FN node, if the time exceeds this time, LPN will go to sleep in order to save power, and need to wake up by pressing the key to start sending friend request again. The default time is 60 seconds.

11.4 LPN Demonstration

11.4.1 Hardware

This demo is based on the GATT master dongle mode. The operation steps of the APP and gateway modes are similar to the GATT master dongle mode. Note that in the gateway mode, the gateway node itself also supports the friend function.

One 8269 GATT master dongle and two 8258 mesh dongle (one burns 8258_mesh.bin, supports Friend function by default. The other burns 8258_mesh_LPN.bin, which is the LPN node).

Note：

- LPN supports generic ONOFF by default, generic Level,but can not support lightness and light CT.
- LPN does not receive 0xffff destination addresses. It only receives unicast addresses and subscribed group numbers. Because there are too many 0xffff commands in the air, if the LPN polling interval is long, the commands in the friend cache will be flushed easily.

11.4.2 Test method

The time-related macros mentioned below can be modified by customers according to their actual needs.

Step 1 Mesh friend node (FN) is powered on and provisioned with SIG_MESH_TOOL.

Step 2 Powered on unprovisioned LPN node, at this time, the LPN is in awake state.

After the LPN node is powered on, the red LED will be in the ON mode. In the unprovision state, do not enter the sleep mode, the purpose is to support GATT provision and ADV provision. In this state, if the provisioning process has not started after 1 minute (LPN_SCAN_PROVISION_START_TIMEOUT_MS), then the system will enter the deep sleep mode, and ADV will not be sent, LED will be turned off, the purpose is to save power

consumption and avoid working in high power consumption mode for too long. If LPN have entered deep sleep, you need to press SW1 or SW2 defined in mesh_lpn_key_map [] to wake up. After wakeup, LPN will start sending ADV again and waiting for the provision flow.

Step 3 Provision and bind key process for the unconfigured LPN in the awake mode.

When Bind key is successful. After 3 seconds (LPN_START_REQUEST_AFTER_BIND_MS), LPN will automatically reboot, and then set lpn_provision_ok to 1, and enter LPN mode, starting to send a friend request command every 2 seconds (FRI_REQ_TIMEOUT_MS).

When the provision is successful, in order to reduce the processing of invalid network messages and reduce power consumption, LPN only receives messages sent from FN through friendship. If you want to receive ordinary network messages, initialize mesh_lpn_rx_master_key to 1.

Step 4 When there is FN, it will automatically establish a friendship. Only when the establishment is successful (red light flashes 3 times), LPN can receive message.

After receiving the friend request, FN will automatically reply to the friend offer, and then establish the friendship. If the establishment is successful, the friend_ship_establish_ok_cb_lpn () will be called back and the red light will flash 3 times (LGT_CMD_FRIEND_SHIP_OK). Then it starts sending friend POLL in a 2 second period (FRI_POLL_INTERVAL_MS). After the FN receives the POLL, if there is a cache message that needs to be sent to the LPN, it will send the message to the LPN. The default maximum number of Cache message (network PDU) is 4 (2 ^ FN_CACHE_SIZE_LOG).

If there is no FN responding to Friend Request, LPN will keep sending friend request in 2 second cycle.

Step 5 The "mesh" window displays the LPN node and ONOFF operation.

First open the "mesh" window, click the "LPN_get_level" INI command, a3 ff 00 00 00 00 00 00 00 04 00 82 05 appears in the lower left corner of the figure below, where 04 00 is the unicast address of LPN (if it is incorrect, it needs to be modified), press Press the Enter key to send the command. After receiving the LPN level status reply, the LPN node will be displayed in the UI, and then you can initiate the ONOFF operation on the LPN, as shown below.

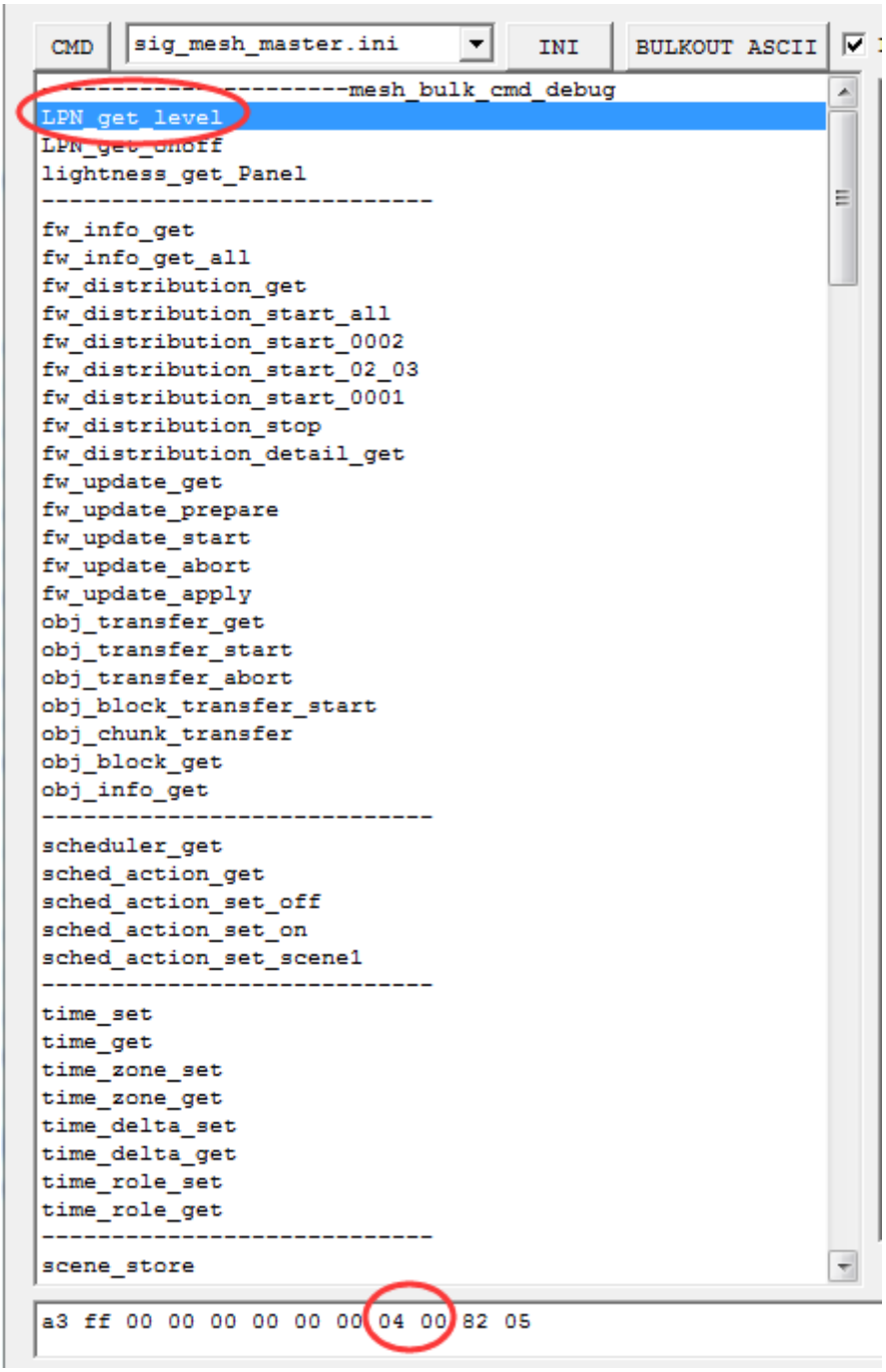


Figure 11.4: LPN_get_level

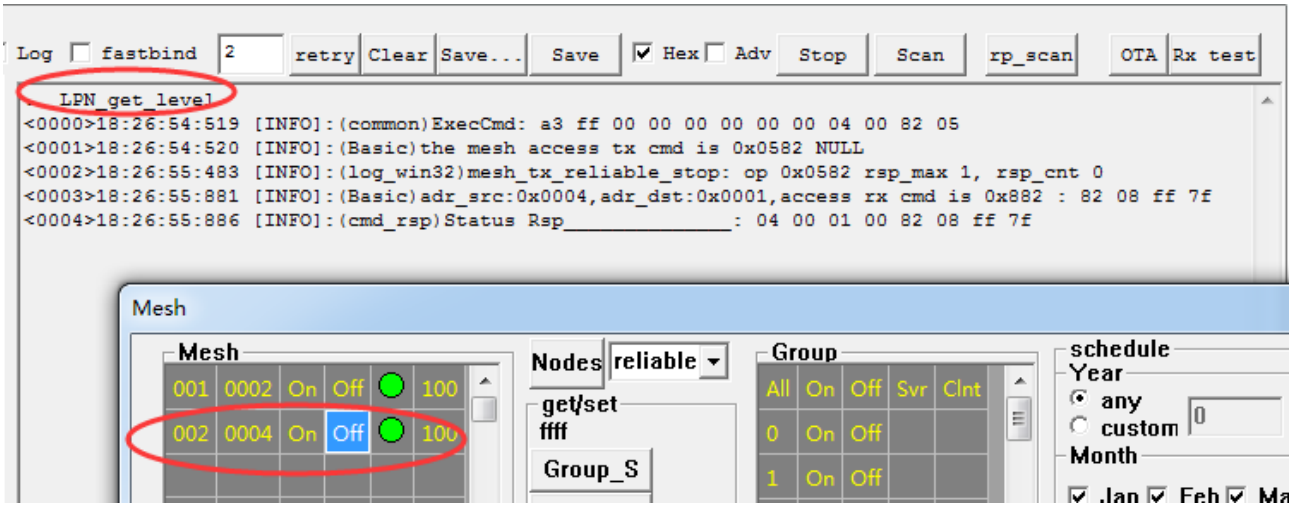


Figure 11.5: ONOFF operation on LPN

Because LPN does not receive the message whose destination address is 0xffff, it needs to send the command in unicast mode. If you have configured a group number for LPN, you can also send commands in group mode.

Also note that after clicking the "Nodes" button or reopening the "Mesh" window, the VC tool will put all the nodes offline, and then send the lightness get all (destination address is 0xffff) command to regain the node status, but There is no separate send command to the LPN node by unicast destination address, so you need to manually click the "LPN_get_level" command or click the ON / OFF command in the mesh window to display the online status, otherwise it is offline.

Step 6 group operation is the same as normal node operation, please refer to "4.5.2 Group Control (ie Subscription Function Demo)"

Step 7 The LPN detects that the FN is powered off and automatically searches for a new FN.

When the FN is powered off, LPN retry 8 times (FRI_POLL_RETRY_MAX) POLL command, where the POLL interval is 170ms (FRI_REC_DELAY_MS + FRI_REC_WIN_MS), if the LPN still does not receive a reply from the FN, it is considered that the FN has been powered off, it will disconnect the friendship and callback friend_ship_disconnect_cb_lpn (), if you need to perform led flashing operation, please add it in the callback function, then resend the friend request to find a new friend node.

Step 8 For now, one friend node of demo SDK establishes a friendship with two LPNs at the same time by default. If you need to modify it, just set MAX_LPN_NUM. The maximum value is 16.

When the LPN is powered off, the FN will detect for 10 seconds (LPN_POLL_TIMEOUT_100MS). If the POLL command has not been received, the node is considered to be powered off. At this time, the FN will clear the LPN information.

Step 9 Press the key to send the ALL ON / OFF command.

When the LPN is in the retention sleep mode, press SW2 (MESH_LPN_CMD_KEY) to wake up the LPN, and then detect the key through suspend_handle_next_poll_interval ()-> mesh_lpn_wakeup_key_io_get (), and then execute the test_cmd_wakeup_lpn () function to alternately send ALL ON / OFF commands. LPN spontaneously sends the access layer command to use master security credentials to encryption by default.

Step 10 Reset to Factory Setting

Long press the button SW1 (MESH_LPN_FACTORY_RESET_KEY) for 3 seconds (LONG_PRESS_TRIGGER_MS) to trigger the factory reset.

11.5 app.c file introduction

Customization of Adv Packet and Adv Response Packet

Please refer to Section 9.

Configuration of fifo part

Please refer to Section 9.

app_event_handler ():

Please refer to Section 9.

main_loop ():

Please refer to Section 9.

user_init():

Please refer to Section 9.

proc_ui():

This function mainly does some UI processing, such as button detection function, and the corresponding test code. When LPN is in non-GATT ota mode, it will send friend request to establish friend relationship. Press key SW2 (KEY_), it will send ON/OFF command alternately; Long press key SW1 (KEY_RESET) for 3 seconds (LONG_PRESS_TRIGGER_MS) to trigger factory reset.

test_cmd_wakeup_lpn():

When pressing the corresponding command button (SW2 in current demo dongle) to wake up the program, the function "test_cmd_wakeup_lpn()" will be executed. This function will send ON/OFF command. After sending the command, it will enter sleep. This function is only used for demonstration.

mesh_lpn_state_proc():

This function focuses on the processing of the working state of the LPN node:

(1) Setting the LED flash when in LPN_MODE_NORMAL mode

(2) LPN has been allocated and has not entered PM for 60 consecutive seconds, return to FRI_ST_REQUEST state.

(3) The LPN is not configured and has not been networked within 60 seconds (LPN_SCAN_PROVISION_START_TIMEOUT of power-up and goes to sleep.

(4) The LPN binds the appkey for 3 seconds and then enters the FRI_ST_REQUEST state.

mesh_lpn_pm_proc():

This function mainly manages the function of LPN node, user can handle some PM states in this function. For example, when the LPN node is networked and in connected state, always enable ENABLE_SUSPEND_MASK to save power consumption. When the user presses the key, it will not enter the PM demo for 4 seconds.

11.6 mesh_lpn.c file introduction

mesh_lpn_sleep_prepare ():

This function handles the sleep processing function of LPN. lpn_sleep.op indicates what command or event needs to go to sleep, and handles the subsequent actions of the event after waking up.

For example, when lpn_sleep.op is equal to CMD_CTL_POLL, it means that the POLL message has just been sent, and then you need to enter the retention sleep time of receive delay, and then wake up to enter the receive window, as shown below:

```
void mesh_lpn_sleep_prepare(u16 op)
{
    if(is_lpn_support_and_en && (BLS_LINK_STATE_CONN != blt_state)){
        if(CMD_ST_NORMAL_TX != op){
            ENABLE_SUSPEND_MASK;
            rf_set_tx_rx_off();// disable tx rx in manual mode,must
            CLEAR_ALL_RFIRQ_STATUS;
            blt_soft_timer_delete(&mesh_lpn_poll_receive_timeout);
        }

        if(CMD_CTL_REQUEST == op){
            blt_soft_timer_update(&mesh_lpn_rcv_delay_wakeup, FRI_ESTABLISH_REC_DELAY_MS*1000);
        }
        else if((CMD_CTL_POLL == op) || (CMD_CTL_SUBS_LIST_REMOVE == op) || (CMD_CTL_SUBS_LIST_ADD == op)){
            blt_soft_timer_update(&mesh_lpn_rcv_delay_wakeup, mesh_lpn_par.req.RecDelay*1000);
        }
        else if(CMD_ST_NORMAL_TX == op){
            blt_soft_timer_update(&mesh_lpn_poll_md_wakeup, get_mesh_adv_interval());
        }
        else if(CMD_ST_POLL_MD == op){
            mesh_lpn_poll_md_pending = 1;
            u32 sleep_ms = FRI_POLL_DELAY_FOR_MD_MS;
            #if MD_MESH_OTA_EN
            if(is_blob_chunk_transfer_ready()){
                sleep_ms = 10;  // waiting for chunk message from FN retransmit completed
            }
            #endif
            blt_soft_timer_update(&mesh_lpn_poll_md_wakeup, sleep_ms * 1000);
            #if !WIN32
            blt_rxfifo.rptr = blt_rxfifo.wptr - 1;// clear buf, blt_rxfifo.rptr will ++ in lib
            #endif
        }
        else if(CMD_CTL_UPDATE == op){
            #if !WIN32
            blt_rxfifo.rptr = blt_rxfifo.wptr - 1;// clear buf, blt_rxfifo.rptr will ++ in lib
            #endif
            blt_soft_timer_delete(&mesh_lpn_poll_receive_timeout);
        }
        else{//CMD_ST_SLEEP

        }
    }
    else if(CMD_ST_NORMAL_TX == op){
        blt_soft_timer_update(&mesh_lpn_poll_md_wakeup, get_mesh_adv_interval());
    }
}
```

Figure 11.6: mesh_lpn_sleep_prepare

Other customized events are:

CMD_ST_SLEEP: After the interaction cycle of a friendship is completed, it enters the retention sleep mode of 2 seconds (friend request interval or poll interval), and then wakes up to enter the interaction of the next cycle.

CMD_ST_NORMAL_TX: Sets the time to next enter mesh_lpn_poll_md_wakeup after an unsolicited mesh message.

CMD_ST_POLL_MD：After sending the POLL, the FN reply MD (more data) is 1, then sleep for 100ms (FRI_POLL_DELAY_FOR_MD_MS), wake up, and continue to send POLL to receive the remaining message.

mesh_feature_set_lpn()：

Initialization of some configurable parameters of LPN. Mainly configure LPN_POLL_TIMEOUT_100MS, the default value is 10 seconds.

12 Switch Project Introduction

12.1 Switch function introduction

The Switch mainly serves to add the function of remote control. The provisioner needs to add the switch node into the network, so that the buttons on the switch can be used to control nodes in the mesh network.

12.2 Switch principle

As a low power remote control node to control mesh, the switch must trigger pairing mode to implement provision. After the provisioner adds the switch to the mesh network, the switch can control nodes in the network.

12.3 app.c file introduction

Customization of Adv packet and Adv response packet

Please refer to Section 9.

Configuration of fifo part

Please refer to Section 9.

app_event_handler ():

Please refer to Section 9.

main_loop ():

Please refer to Section 9.

user_init():

Please refer to Section 9.

proc_ui ():

The "proc_ui" function configures key scan with the interval of 4ms. "mesh_proc_keyboard" is the interface function for key processing.

- When "keycode" is "RC_KEY_A_ON", the switch will send the all_on command to turn on all lights in the network.
- When "keycode" is "RC_KEY_A_OFF", the switch will send the all_off command to turn off all lights in the network.

proc_led():

First the configuration function "cfg_led_event" is used to configure LED blinking frequency and time. E.g. "cfg_led_event(LED_EVENT_FLASH_1HZ_4S)": configure LED to blink for 4s with the frequency of 1Hz. Then the function "proc_led" serves to control the processing of LED blinking part.

mesh_switch_init():

The "mesh_switch_init" contains setting of two parts:

- Code setting of wakeup IO of switch part, as well as enabling of the wakeup enable flag bit.
- IO setting of LED part. By default, LED pin is configured as GPIO mode with 100kohm pull-down resistor, and LED will blink four times after power on.

proc_rc_ui_suspend():

The processing function for sleep function part is "proc_rc_ui_suspend()".

The processing of sleep part in current SDK is set as below: In advertising state, if MCU directly enters deep state without sending packets, after wakeup by key press, MCU will continue to enter deep state when packet transmission is finished. After pairing mode is triggered, MCU will enter deep state 30s later, and it won't enter deep state temporarily in link state.

For the processing flow of sleep part, please refer to section 12.7.

kb_scan_key ():

"kb_scan_key" is the interface of matrix keyboard scan part. In current SDK, by default "numlock_status" is set as 0 to indicate the numlock in full keyboard, while "read_key" is the read key value.

12.4 Key Event Detection Process

12.4.1 Code Block

```
void mesh_proc_keyboard ()
{
    static u32      tick_key_pressed, tick_key_repeat;
    static u8       kb_last[2];
    int det_key = kb_scan_key (0, 1);
    ......
    ////////////////////////////////////////////////////////////////////////////////
    //          key change:pressed or released
    ////////////////////////////////////////////////////////////////////////////////
    if (det_key)    {
        ......
        if(kb_event.cnt)
        {
            ...... // key was detected pressed. MCU run the code here one time for one press
↪   action.
        }
        ////////////////////////   key released  ////////////////////////////////////
        else {
            ...... // key was released . MCU run the code here one time for one release action.
↪
            rc_repeat_key = 0;
            key_released = 1;
        }
        ......
```

```
    }
    ////////////////////////////////////////////////////////////////////////////////
    //              no key change event
    ////////////////////////////////////////////////////////////////////////////////
    else if (kb_last[0])
    {
        //  long pressed // key was detected in a continuously pressed state. for each
        ↪  main_loop, MCU run the code here until the key is released.
        if (clock_time_exceed(tick_key_pressed, 2000000))   // long pressed // 2000000 is the
        ↪  threshold for long press detection
        {
             if ((kb_last[0] == RC_KEY_A_ON && kb_last[1] == RC_KEY_1_OFF) ||
                 (kb_last[1] == RC_KEY_A_ON && kb_last[0] == RC_KEY_1_OFF))
             {
                    if(SWITCH_MODE_NORMAL == switch_mode){  // long pressed event
                        switch_mode_set(SWITCH_MODE_GATT);
                    }
             }
        }
        ......
    }else{
        ...... // no key was detected.
        key_released = 1;
    }
    ......
}
```

Introduction to key events:

- Key pressed: where the comment "key was detected pressed" indicates that a key press was detected.
- key was released: where the comment "key was released" indicates that a key release was detected.
- Long key press: where the comment "// long pressed // 2000000 is the threshold for long press detection" indicates that a long key press was detected.
- No key event: where the comment "no key was detected." indicates that a key press was detected.

Developers can add their own keystroke functionality to the above.

12.5 Switch Engineering Long Press Handling Logic

Determine the current key is pressed and use clock_time_exceed to start timing from the time the key is pressed, when the set time is reached, then trigger the processing of a long key press.

Example: Press RC_KEY_A_ON and RC_KEY_1_OFF for two seconds to trigger the switch to enter GATT mode.

```
else if (kb_last[0])
{
    //  long pressed // key was detected in a continuously pressed state. for each main_loop,
     ↪  MCU run the code here until the key is released.
    if (clock_time_exceed(tick_key_pressed, 2000000))   // long pressed // 2000000 is the
     ↪  threshold for long press detection
    {
            if ((kb_last[0] == RC_KEY_A_ON && kb_last[1] == RC_KEY_1_OFF) ||
                (kb_last[1] == RC_KEY_A_ON && kb_last[0] == RC_KEY_1_OFF))
            {
                if(SWITCH_MODE_NORMAL == switch_mode){  // long pressed event
                    switch_mode_set(SWITCH_MODE_GATT);
                }
            }
    }
    ......
}
```

12.6 Example of Sending Commands Using the Soft_timer Cycle

For an example of sending commands using soft_timer cycle, please refer to this section "Example of Sending Commands Using the Soft_timer Cycle".

12.7 Configuration of Switch Part

12.7.1 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}, }
```

User can configure the contents of actual "key_table" according to the number of drive pins and scan pins which correspond to the number of columns and rows respectively.

12.7.2 Configure IOs for Drive Pins and Scan Pins

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

Modify macros corresponding to "KB_DRIVE_PINS" and "KB_SCAN_PINS" according to actually used pins.

Then customize IO attributes for drive pins and scan pins, as shown below:

IO attribute setting corresponding to drive pins:

```
#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
```

IO attribute setting corresponding to scan pins:

```
#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
```

Suppose it's needed to modify "GPIO_PB6" as "GPIO_PB7" in drive pin part, the following parts should be modified accordingly.

```
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
```

12.7.3 Turn on/off Light via Switch

According to different key values, different commands will be sent so as to process correspondingly.

Please refer to key processing program "mesh_proc_keyboard ( )". Switch cannot control light nodes before it's added into the network. User can simultaneously press the "RC_KEY_A_ON" and "RC_KEY_1_OFF" button on the switch for more than 2 seconds to trigger pairing mode, and add the switch into the mesh network via the provisioner, so that all light nodes in this network can be turned on/off via the "RC_KEY_A_ON" and "RC_KEY_A_OFF" button. Please refer to section 12.5 for details.

12.8 Switch Operation

First follow the provision operations in section 10.4. Connect the switch with PC USB via Telink burning EVK (as shown in the figure below), and then burn the switch with corresponding firmware.

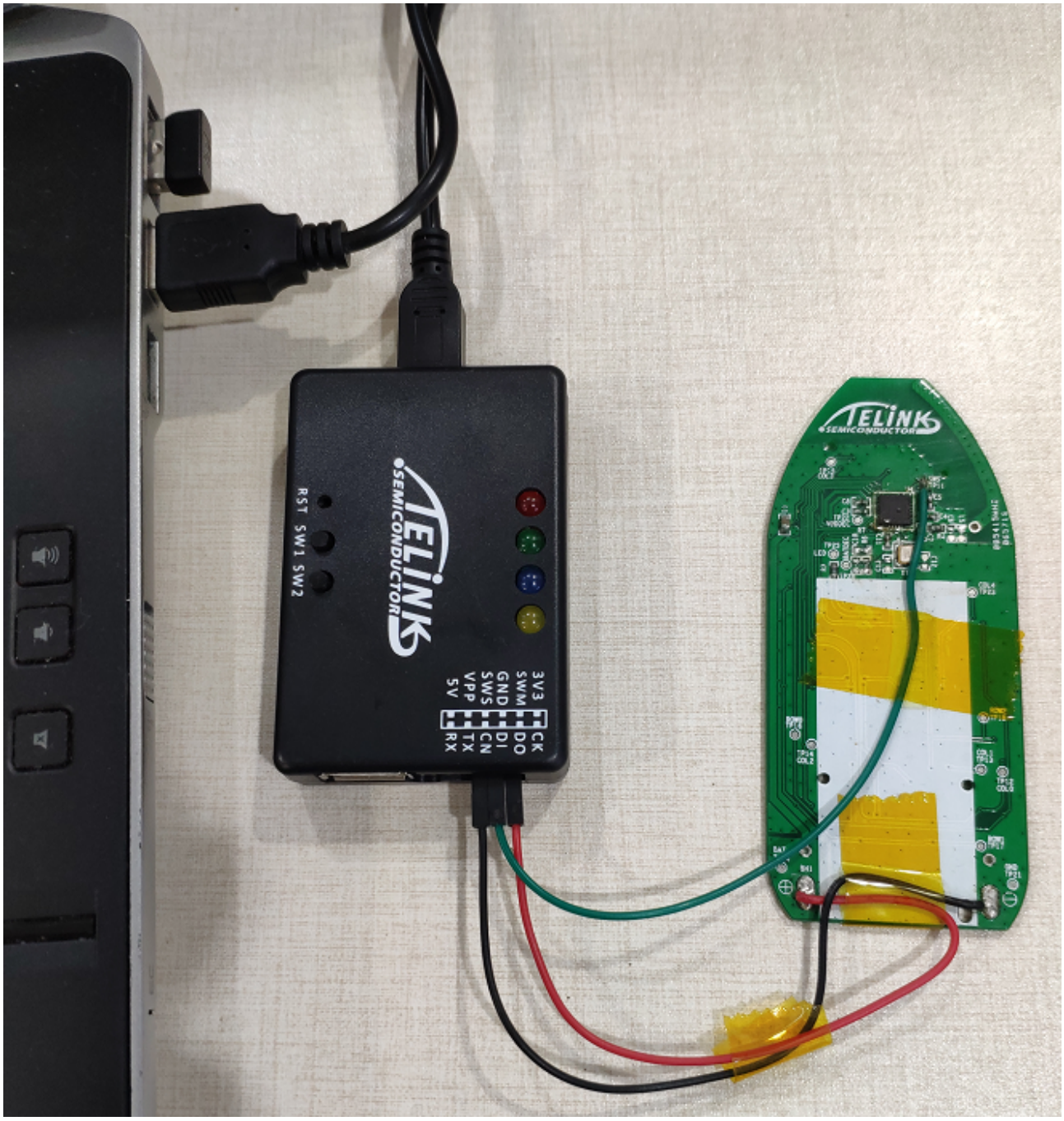


Figure 12.1: Switch Burning Connection

The switch buttons are shown as below:

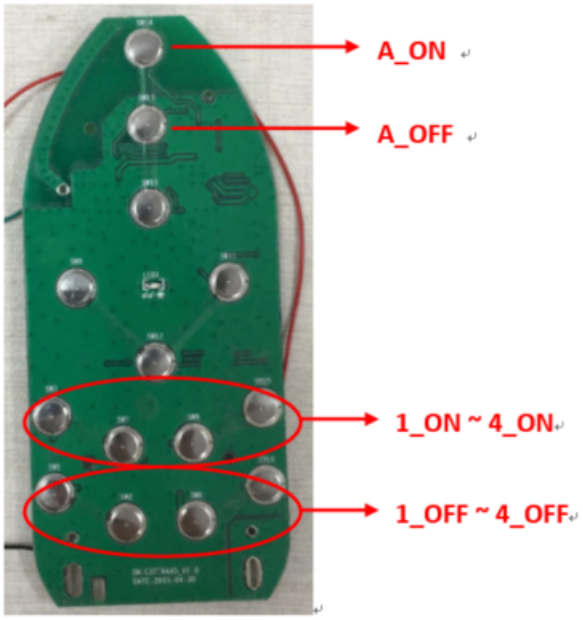


Figure 12.2: Switch button

Power on the switch device. After power on, as a low power node, the switch must trigger pairing mode by simultaneously pressing the "RC_KEY_A_ON" and "RC_KEY_1_OFF" for more than 2s, so that it can be added into mesh network via the provisioner.

After the switch triggers pairing mode, its LED light will continuously blink four times to indicate it enters pairing mode. Power on the provisioner (if it's powered down), and wait for 15s or so. The LED on the switch will continuously blink four times to indicate the switch has already been added into the network.

Then the "RC_KEY_A_ON" and "RC_KEY_A_OFF" on the switch can be used to turn on/off all light nodes in the network.

12.9 Flow chart for Switch RC

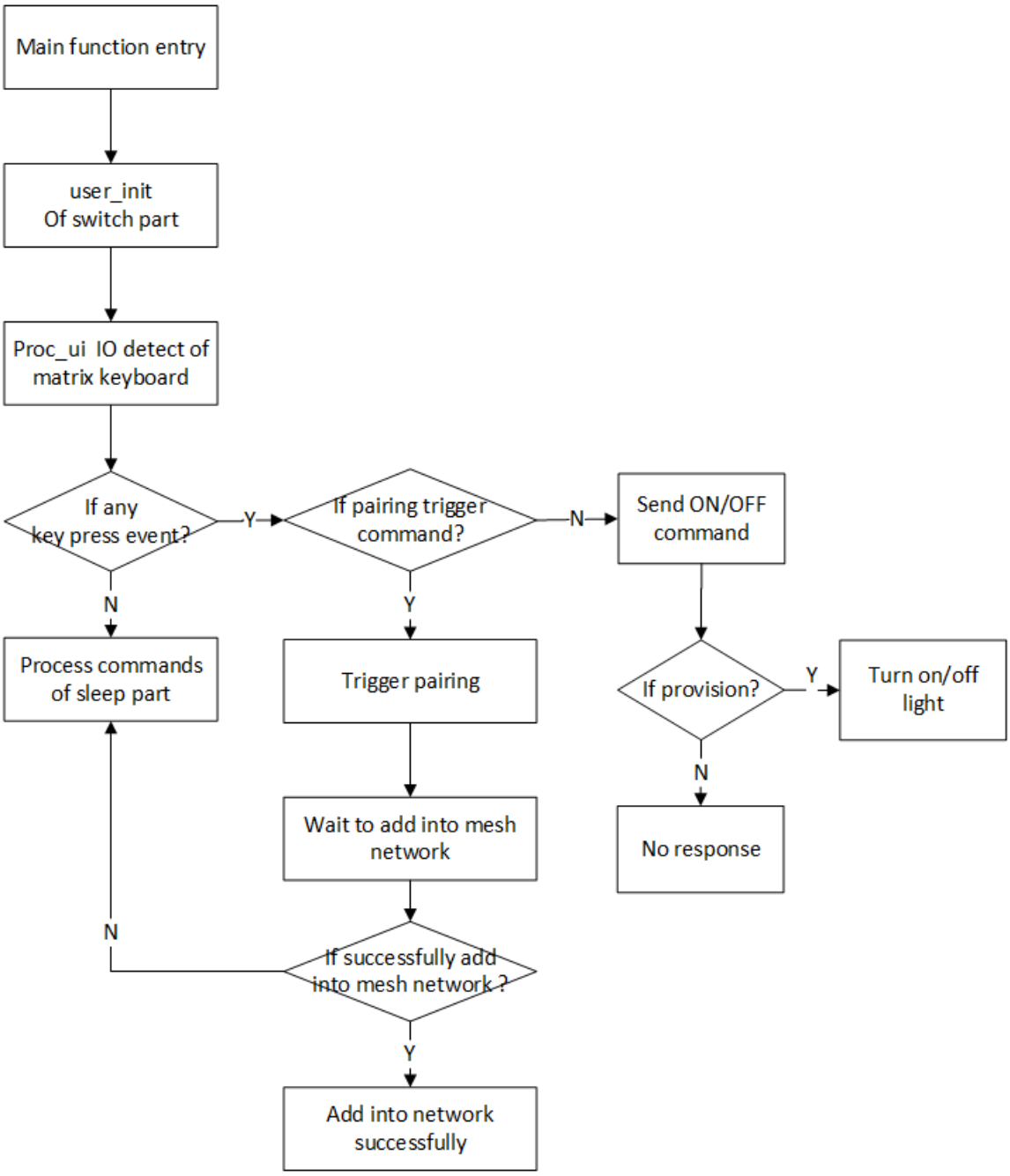


Figure 12.3: Flow chart for switch RC

12.10 Flow chart for sleep processing

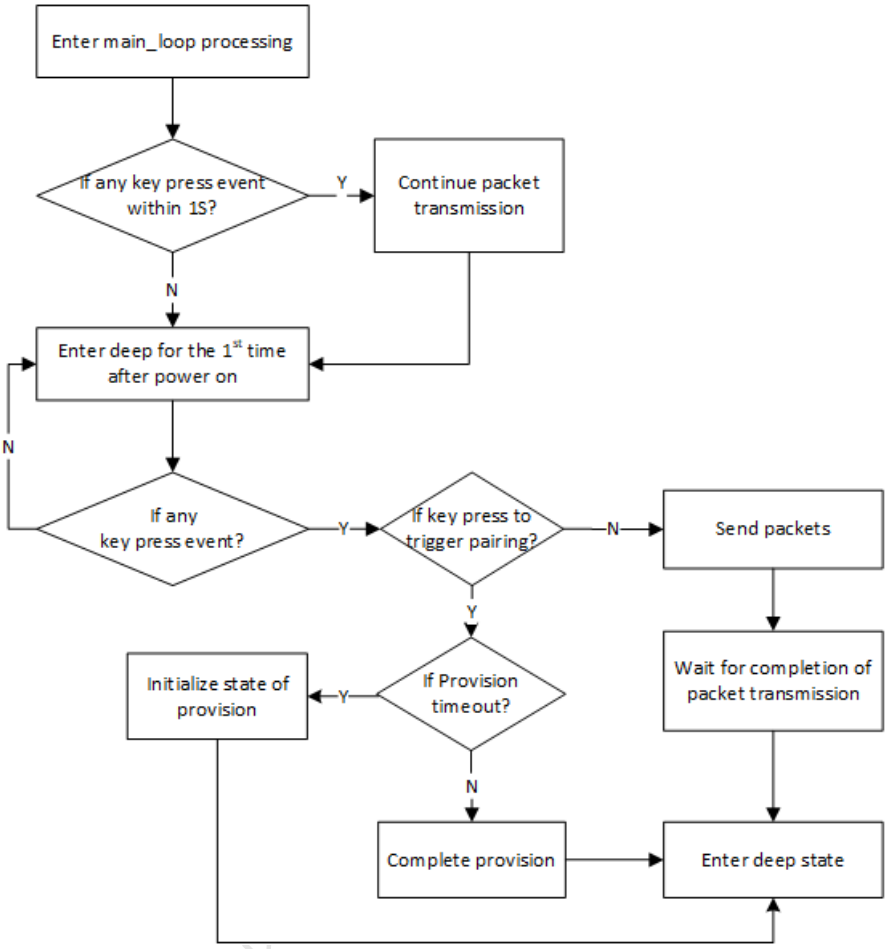


Figure 12.4: Flow chart for sleep processing

12.11 Modify the destination address of button send command

The 4 sets of buttons shown below support modifying the destination address of the command:

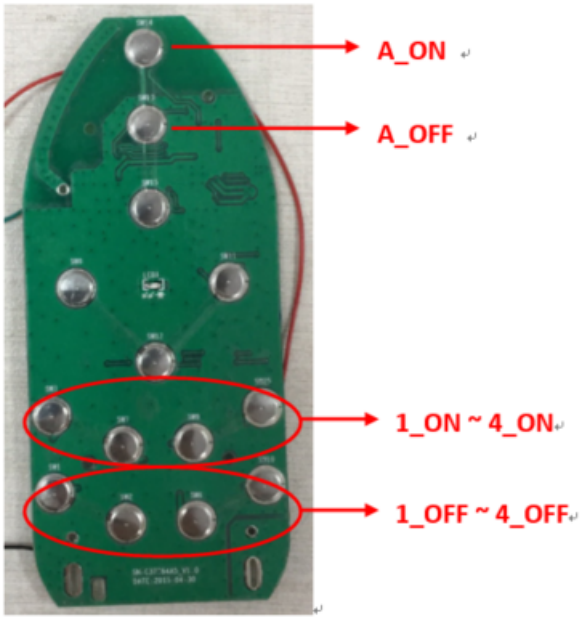


Figure 12.5: Switch button

- 1_ON / 1_OFF：The default function of the key is to send the onoff command with the destination address 0xC000.
- 2_ON / 2_OFF：The default function of the key is to send the onoff command with the destination address 0xC001.
- 3_ON / 3_OFF：The default function of the key is to send the onoff command with the destination address 0xC002.
- 4_ON / 4_OFF：The default function of the key is to send the onoff command with the destination address 0xC003.

If it is needed to modify the address of the key sending command, for example, to change the destination address of "1_ON / 1_OFF" from 0xC000 to 0xD000, you can send the publish set command through the INI command of the host computer, and the configuration example is as follows (the primary address of the remote control node, i.e., the node address of the example remote control node is 0x0025):

```
cfg_pub_set_sig0025     =e8 ff 00 00 00 00 00 00 25 00 03 25 00 00 0D 00 00 ff 00 15 00 10
```

Change the destination address of "2_ON / 2_OFF" from 0xC001 to 0xD001, as shown in the following example:

```
cfg_pub_set_sig0026     =e8 ff 00 00 00 00 00 00 25 00 03 26 00 01 0D 00 00 ff 00 15 00 10
```

The parameters of publish set are:

Field	Size (bits)	Description	Req.
Opcode	8	The message opcode	M
ElementAddress	16	Address of the element	M
PublishAddress	16	Value of the publish address	M
AppKeyIndex	12	Index of the application key	M
CredentialFlag	1	Value of the Friendship Credential Flag	M
RFU	3	Reserved for Future Use	M
PublishTTL	8	Default TTL value for the outgoing messages	M
PublishPeriod	8	Period for periodic status publishing	M
PublishRetransmitCount	3	Number of retransmissions for each published message	M
PublishRetransmitIntervalSteps	5	Number of 50-millisecond steps between retransmissions	M
ModelIdentifier	16 or 32	SIG Model ID or Vendor Model ID	M

Table 4.98: Config Model Publication Set message structure

Figure 12.6: publication_set_parameters

For details, refer to "4.3.2.16 Config Model Publication Set" in mesh spec V1.1.

If you want to set up via mobile app, please refer to "Device Setting (Switch Device)" section in the chapter Android and iOS APP User Guide.

If you want to add another set of keys to configure the onoff publish address, such as "5_ON / 5_OFF", change the value of ELE_CNT_EVERY_LIGHT to 5, and then configure the onoff publish address of the client model (primary address + 4).

For the description of ELE_CNT_EVERY_LIGHT, please refer to this section "Definition of the number of elements of a node".

12.12 IV Index Update Mode for Switch

- Scenario 1: The Switch wakes up every 96 hours, sends a security beacon, then enters the scan adv state, scans for security beacons, and goes to sleep if any of the valid security beacons are scanned. If it is not received after timeout (SWITCH_IV_RCV_WINDOW_S), it also goes to sleep.

See switch_trigger_iv_search_mode(int force) for details on handling.

- Scenario 2: After powering down and then re-powering up, switch_trigger_iv_search_mode(1) is also called to perform the send and scan security beacon action of scenario 1.

13 Connect with a Platform

When connect with a certain platform, you need to configure some options, especially the provision method. Select by configuring MESH_USER_DEFINE_MODE.

13.1 Normal Mode

13.1.1 No OOB provision mode

Configuration method:

Provision uses MESH_NO_OOB mode.

VENDOR_ID is 0x0211

When testing, you can directly use our mobile app or host computer tools to provision.

13.1.2 Static OOB provision mode

13.1.2.1 Light Node Burn Static oob

When burning the firmware, just write 16 bytes directly in the flash fixed location FLASH_ADR_STATIC_OOB (for example, 0x77800). If there is no burning (all 0xff), it means that no oob mode is used. If you need to modify the flash address, modify the macro FLASH_ADR_STATIC_OOB.

13.1.2.2 Light node Device uuid

The device uuid is generated by user_prov_multi_device_uuid () -> uuid_create_by_mac (tbl_mac, prov_para.device_uuid) by default and can be obtained by the following methods:

1) Read prov_para.device_uuid through BDT tool
2) Obtain unprovision broadcast package through the general APP

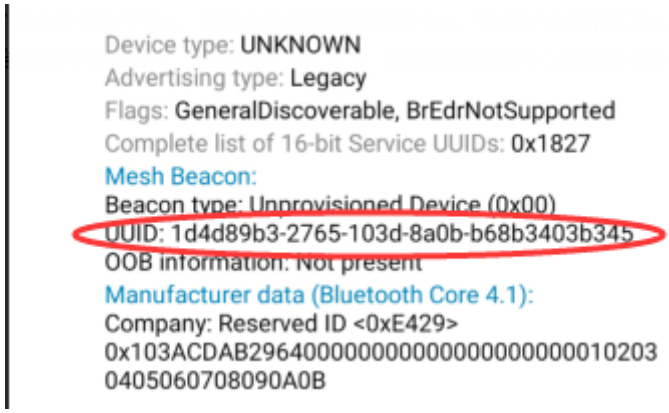


Figure 13.1: Device uuid

3) Obtain unprovision broadcast package through TI sniffer

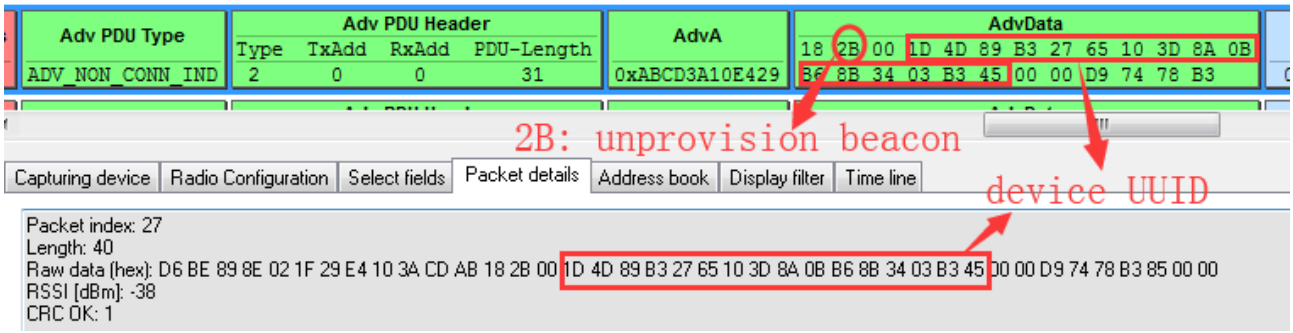


Figure 13.2: unprovision broadcast package

4) when the connection is successful, the device uuid will be printed out

```
<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
```

Figure 13.3: Connection successful and print device uuid

5) In the case of gateway provision, when selecting a node obtained by scan, the device uuid will be printed.

```
<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
```

Figure 13.4: Print device uuid at a scan node

13.1.2.3 User Customized uuid Method

If the user wants to customize the device uuid, set NORMAL_MODE_DEV_UUID_CUSTOMIZE_EN to 1.

13.1.2.4 Provisioner static oob database

The Provisioner needs to fill in the oob data of the node to the oob database file (oob_database.txt):

The data format of oob database is as follows:

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

For example: 1d4d89b32765103d8a0b29e4103acdab ff000000000000000000000000000000

Field analysis is as follows:

1d4d89b32765103d8a0b29e4103acdab: The device uuid of the provisioned node.

ff000000000000000000000000000000: The oob data of the provisioned node, that is, the value written by Light Firmware at the fixed flash location FLASH_ADR_STATIC_OOB (for example, 0x77800)

If there are multiple nodes, the line break can be increased, for example:

1d4d89b32765103d8a0b29e4103acdab ff000000000000000000000000000000

9f6f6e6e5e90943db9be6d79446a9e37 ffaa000000000000000000000000000000

13.1.2.5 Test steps

Please follow the general process for provision and testing.

The test results of the static oob provision success, the screenshot using GATT master dongle mode is as follows:

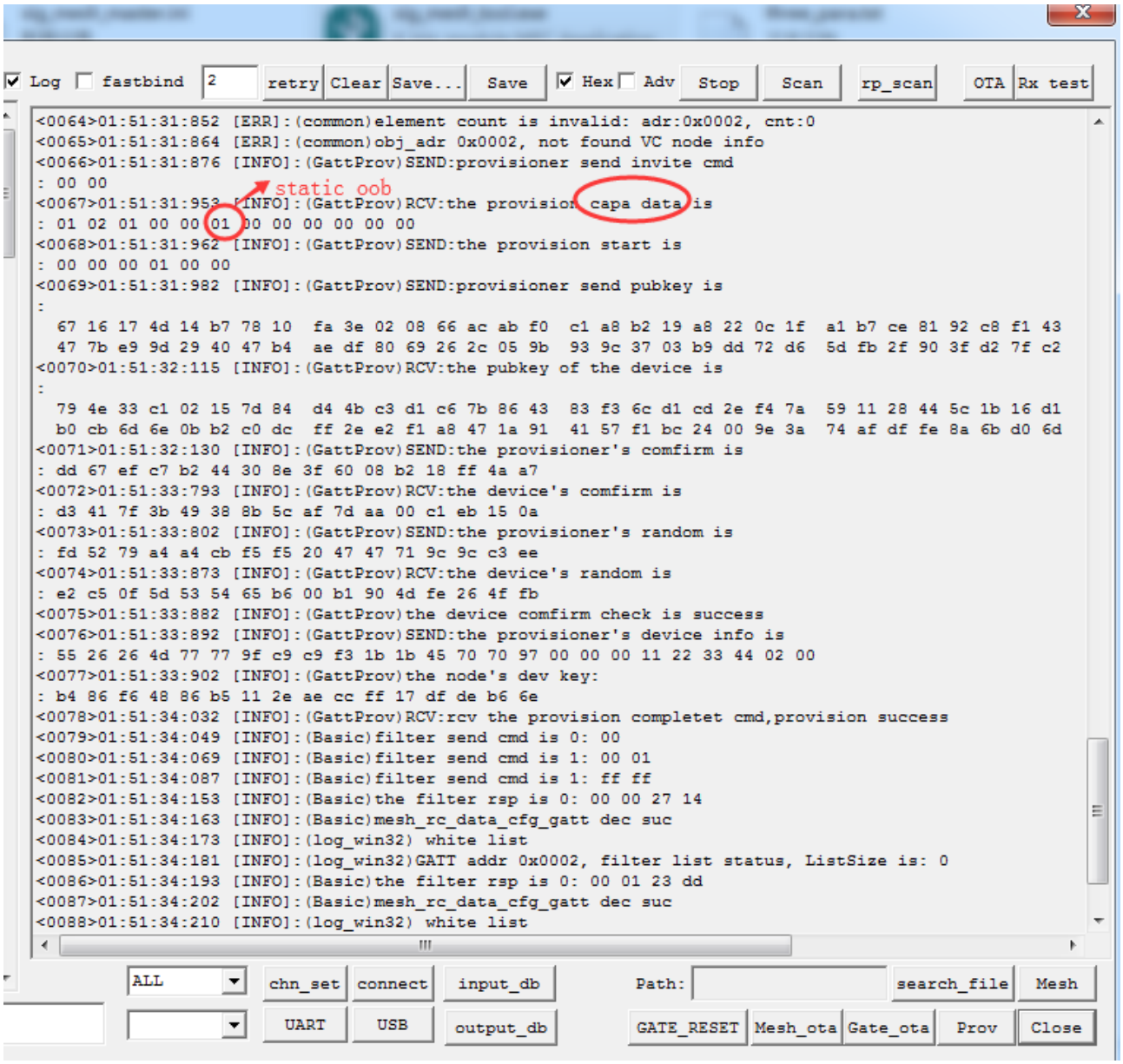


Figure 13.5: static oob provision success

Capability data, please refer to the following chapters of spec for more detailed analysis.

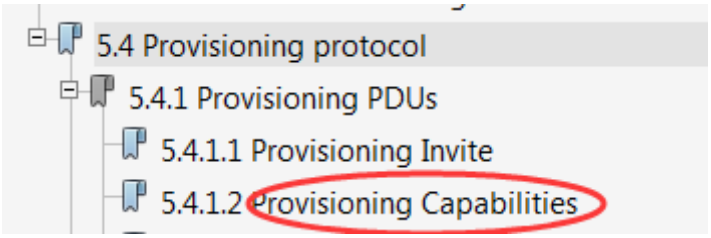


Figure 13.6: capability data

Data analysis is as follows:

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

01：Provisioning PDU Type, 01 indicates 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 Ali Tmall Genies Platform

13.2.1 Configuration

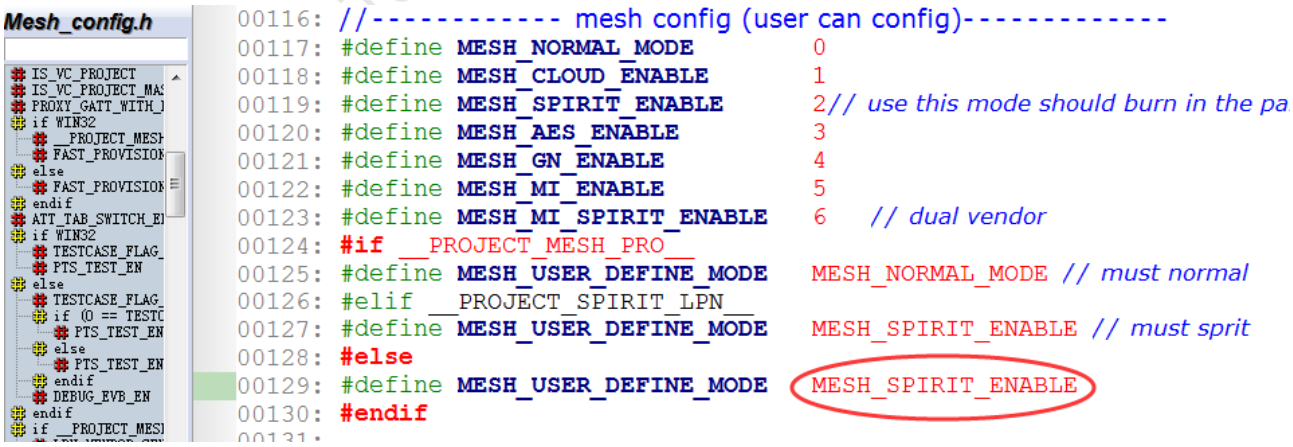


Figure 13.7: Configuration

Provision uses MESH_STATIC_OOB mode.

VENDOR_ID is 0x01A8.

13.2.2 Apply tri-truple from Ali

The default tri-truple information is null(con_sec_data[16] is null), code is in user_ali.c, shown as following:

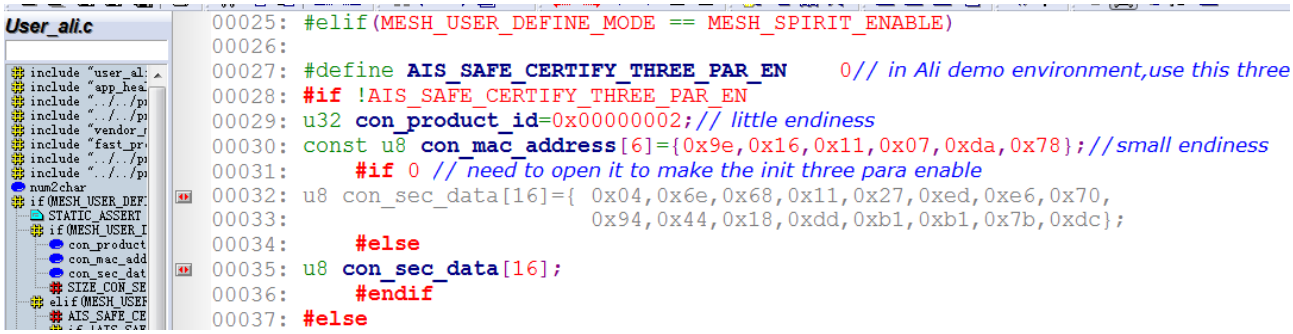


Figure 13.8: Apply tri-truple

So user need to apply tri-truple from Ali:

(Total 24byte: PID(4byte, smaller end) + MAC(6byte, bigger end) + secret data(16 byte)),

Then burn to FLASH_ADR_THREE_PARA_ADR(0x78000), and program will read parameter in 0x78000 automatically.

13.2.3 Use SDK Default tri-truple

User can use SDK default tri-truple for demo, Open it as follows: (Enable the preset con_sec_data [] in the user_ali.c file)

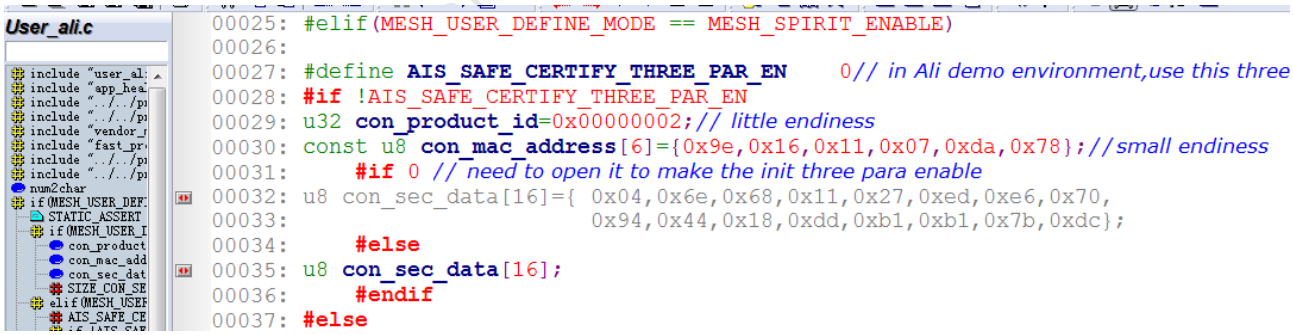


Figure 13.9: Apply tri-truple

However, because there is only one default tri-truple, it can only be used for a single node demonstration when testing.

13.2.4 Provision via Tmall Genie

Provision can be done directly through Tmall Genie's voice commands.

13.2.5 Provision via Firmware

Because Tmall Genie mode only supports the static oob mode by default, oob and tri-truple have a binding relationship, so the firmware needs to know the information of the tri-truple to provision. So you need to add the tri-truple to this file of the firmware:

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

SDK default tri-truple information has been added to this file. When adding new tri-truple information, just refer to this format.

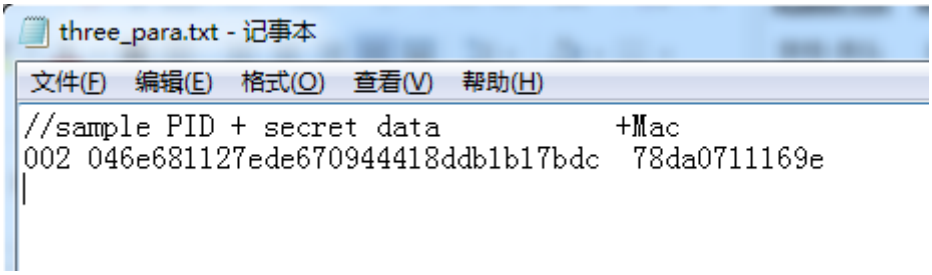


Figure 13.10: firmware file

Then refer to Provision part in chapter 4 for detail steps.

13.2.6 Dual Modes of static oob and no oob

Tmall Genie mode, the node end only responds to the static oob mode by default. If you need to support no oob mode at the same time, change ENABLE_NO_OOB_IN_STATIC_OOB from 0 to 1.

13.3 Xiaomi Xiao'ai Platform

13.3.1 Configuration

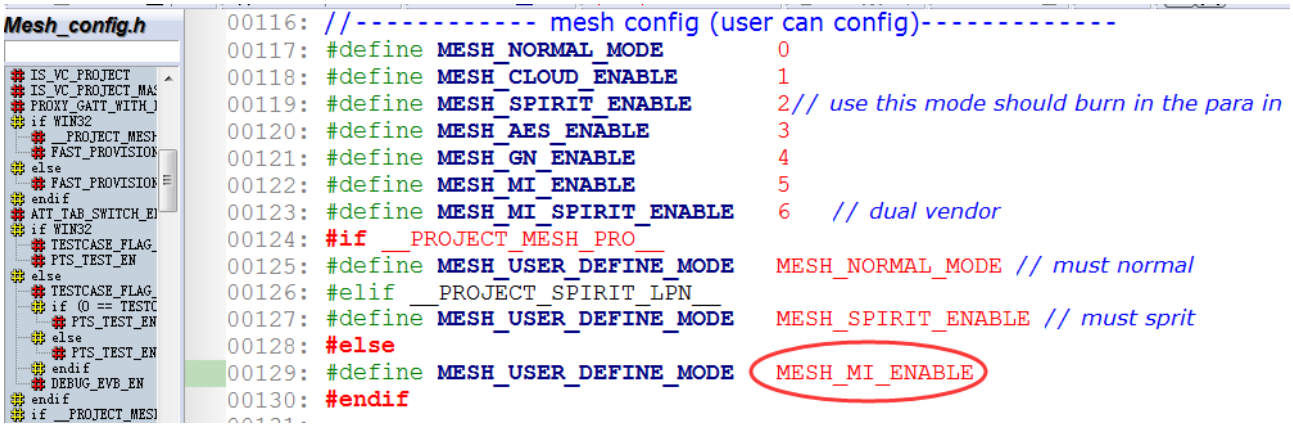


Figure 13.11: Xiaomi defined mesh provision mode

Provision uses Xiaomi defined mesh provision mode

VENDOR_ID is 0x038F.

13.3.2 Certification Data Setting

- R & D test mode: The default certification data of the SDK is used, dev_cert_pri [], so it can only be used in the single node test mode. There are 6 certificates by default. Because these certificates are public, if they have been used by others, conflict will happen. So it is recommended for users to use the certificate they applied for and then testing it in production mode.
- Production mode: When production or multi-node network testing is required, flash writing is required. The steps are as follows:

```
#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
```

Figure 13.12: Apply certificate

Step 1 Define MI_CER_MODE as FLASH_CER_MODE, generate firmware

Step 2 Burn certification data to DEV_SK_FLASH_ADR(0x7f000)

13.3.3 Provision Test

After burning firmware, please make sure that the MAC address of the flash (512K flash is at 0x76000 and 1M flash is at 0xFF000) is empty (that is, all 0xff), otherwise it will prompt "Unable to connect" or "Provision failure".

When firmware is powered on for the first time, it will extract the MAC from the certificate, write it to the MAC address sector, and generate some necessary parameters.

After the node is powered on, it can be directly provisioned through Xiao'ai (Voice instruction example: "Xiao'ai, add device").

13.4 Dual Vendor Mode (Tmall Genies and Xiaomi Xiaoai)

13.4.1 Function Introduction

When the node leaves the factory, it will send adv. packets in Ali and Xiaomi modes at the same time, which can be provisioned by either Tmall Genies or Xiao'ai. Once provision is successful, subsequent functions are

performed according to the selected mode, including parameters such as the vendor model and the transmit count. For parameter switching functions, see mesh_ais_global_var_set ().

The production test function in unprovisioned state of is performed according to the Xiaomi mode.

After provision is successfully, you can execute the kick light command or restore the factory settings to return to the dual vendor mode and select again.

Which mode you are currently in can be viewed through the provision_mag.dual_vendor_st variable.

13.4.2 Configuration

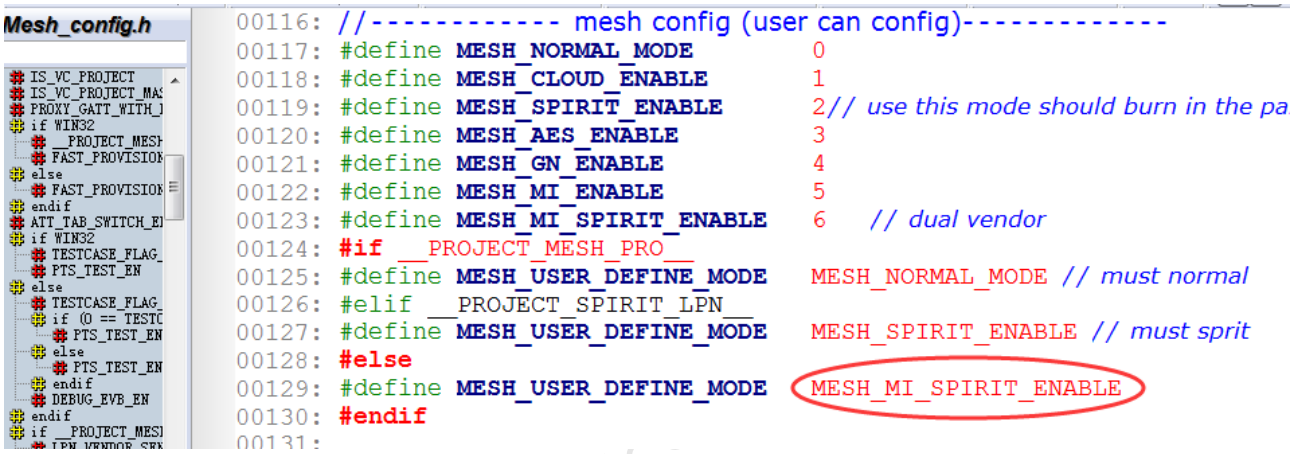


Figure 13.13: Provision method

Provision method and parameter configuration, please check 13.2 and 13.3.

14 Factory Reset

14.1 8258_mesh/8269_mesh Node

14.1.1 Function Introduction

Factory reset reset execution action, please refer to factory_reset_handle () or kick_out():

```
{
irq_disable();
factory_reset();   // Flash erasing
#if DUAL_MODE_WITH_TLK_MESH_EN
UI_resotre_TLK_4K_with_check();
#endif
show_ota_result(OTA_SUCCESS);  // LED indication
start_reboot(); // MCU reboot
}
```

If the customer has modified the flash map, or used the customer flash section (the default is 0x7a000—0x7f000, and erase is not performed on the area by default), you need to reconfirm the factory_reset () function to confirm whether there are sector errors or missing erases.

Power-up sequence detection function factory_reset_cnt_check ():

(a) After power-on, the power-on sequence will not be detected until after VALID_POWER_ON_TIME_US (default 50ms). Because it is necessary to filter the pulse voltage generated when some power supplies are powered on.

(b) clear_st is 4:

First reset_cnt_get_idx () to obtain the sequence before power off, if it is an odd number, it means that the previous power-on sequence does not meet expectations, directly clear the power-on sequence and restart counting. If it is even, then it is as expected.

Then, check whether the stored power-on sequence value satisfies the condition that triggers a factory reset, and if it does, execute a factory reset. If not, immediately add 1 to the sequence value.

(c) clear_st is 3, get the power-on sequence value by get_reset_cnt (), get the timing time, and start the timing of the first phase.

(d) clear_st is 2, the first phase timing meets the requirements, get the power-on sequence value through get_reset_cnt (), get the timing time, and start the second phase timing.

(e) clear_st is 1. If the second phase is over and power has not been turned off, it means that the power-on time is not as expected, and the power-on sequence is directly cleared.

14.1.2 Default trigger action

Low power node, e.g., LPN, cannot be triggered with booting sequence, because LPN cannot count time and determine timing sequence. User can trigger this with button-pressing, call functions described in 14.1.1.

User can follow the steps below to reset a SIG_mesh module (non-LPN) to factory default configuration:

Step 1 Power on the SIG_mesh module, and wait for more than 30s (equivalent to initial power on).

This operation will erase previous power on record, and ensure it conforms to the timing sequence requirement of initial power on rather than any power-on sequence defined by "factory_reset_serials[]".

Step 2 Power cycle the SIG_mesh module three times. Note: After each power on, the module must be powered down within the range of 0~3s. This operation conforms to the requirement of former three power-on sequences in the "factory_reset_serials[]".

Step 3 Power cycle the SIG_mesh module two times. Note: After each power on, the module must be powered down within the range of 3~30s. This operation conforms to the requirement of latter two power-on sequences in the "factory_reset_serials[]".

Step 4 Power cycle the SIG_mesh module.

The "factory_reset_handle()" in the "user_init()" will detect and get the result that previous five power-on sequences match with the trigger requirement of "Factory Reset". The red LED light on the module will blink for 8s with the frequency 1Hz to indicate factory reset success.

Note:

> Since some module may need some time to finish power down, to ensure its MCU is stopped completely, it's needed to wait for a duration (e.g. 2s, depend on module) after power-down operation.

Power-on timing sequence is defined by the array below:

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

14.1.3 Method to modify power-on sequence

To modify power-on sequence, it's only needed to modify/ add/ delete sequences in the array "factory_reset_serials[]" correspondingly, as long as the requirements below are met:

- The left value should be smaller than the right value.
- When it's needed to add/delete sequences, since one sequence corresponds to two values, multiple of two values must be added/deleted correspondingly.

E.g. To modify power-on sequence as six sequences, the following method can be followed.

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

14.1.4 The function of the previous mesh network can be restored after the reset action is triggered

The above 1 ~3 mode is the normal mode for restoring factory settings.

After trigger factory reset, if it does not re-configure the network within a certain period of time (such as 30 seconds), some users may hope to automatically restore the previous network information. The SDK provides 2 APIs needed to implement this function:

- mesh_reset_network (u8 provision_enable): Restore the network information in the ram to the default network state. Since the parameter provision_enable is 1, the device will send unprovision beacon and pb_adv, and the device can be reconfigured. At this time, only the RAM data is restored, but the flash information has not changed.
- mesh_revert_network (): Reload network information from flash.

Implementation: When the user triggers the factory reset action and enters the kick_out function, it directly calls mesh_reset_network (1) to restore the network information in the ram to the default network, without calling factory_reset () and start_reboot ().If the user wants to restore the network, directly call mesh_revert_network () to reload the mesh information from the flash.

14.2 Gateway Node + Host Computer

To reset the gateway, you need to perform two actions: one is to delete the mesh_database.json, and the other is to clear the parameter area of the flash of the gateway dongle (for example, by resetting the device 5 times). Because there are two actions that need to be performed, in order to facilitate the operation, a "GATE_RESET" button is added to the firmware. After executing this button, the above two things will be performed. After the gateway dongle clears its own flash parameter area, it will automatically call start_reboot () for a soft restart.

Note: this button just resets the gateway itself and will not send commands to reset other nodes.

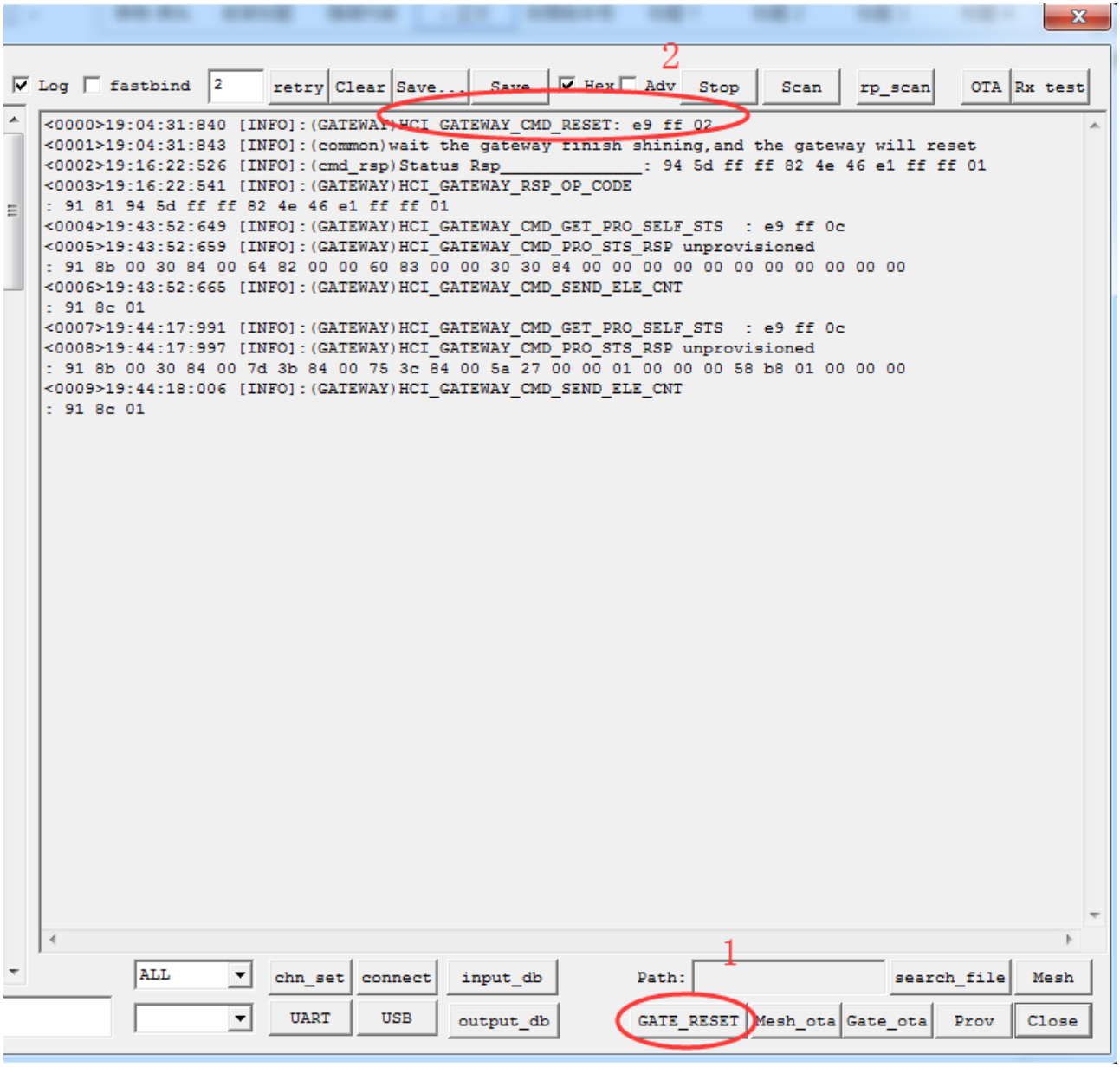


Figure 14.1: gateway reset

The send command E9 FF 02, E9 FF represents the macro HCI_CMD_GATEWAY_CTL, and the related processing flow is in the function app_hci_cmd_from_usb_handle.

```
else if (HCI_CMD_GATEWAY_CTL == type){
        #if IS_VC_PROJECT
        ret = fifo_push_vc_cmd2dongle_usb(buff, n);
        #else
        #if GATEWAY_ENABLE
        ret = gateway_cmd_from_host_ctl(hci_data, hci_data_len);
        #endif
        #endif
```

```
    }
```

An opcode of 02 indicates HCI_GATEWAY_CMD_RESET, and the related processing flow is in the function gateway_cmd_from_host_ctl.

```
else if (op_code == HCI_GATEWAY_CMD_RESET){
        factory_reset();
        light_ev_with_sleep(4, 100*1000);   //10hz for about the 1s
        start_reboot();
    }
```

14.3 GATT master dongle + Host Computer

There is no storage parameter in the flash of master dongle, so just delete the mesh_database.json file of the upper computer, and then reopen the upper computer tool.

14.4 LPN Node

Low-power nodes cannot be reset by powering on 5 times, because when it is in sleep state, it will take some time to consume power after power off. Therefore, LPN nodes generally need to press keys and other methods to trigger and call the functions in the first section of this chapter.

DEMO LPN: long pressed SW1 key (MESH_LPN_FACTORY_RESET_KEY) for more than three seconds (LONG_PRESS_TRIGGER_MS) .it will flash 4 times, indicating that the reset is successful..

14.5 Switch Node

Low-power nodes cannot be reset by 5 power-ups. Need to press keys, press and hold the key combination: RC_KEY_A_ON + RC_KEY_4_OFF for more than three seconds, it will flash 4 times, indicating that the reset is successful.

15 Fast bind Mode (PROVISION_FLOW_SIMPLE_EN Mode)

15.1 Function Introduction

This mode is a non-standard mode and is used to speed up the provision. The improvements are as follows: After the implementation of the Spec definition provision flow, the unicast address, netkey, and other information are assigned, you need to send get composition data, app key add in order, and perform key bind on each model in the composition data obtained. This is not so complicated, in most cases a device will only have an APPkey. So we defined this Fast bind pattern.

The Fast bind mode is mainly to optimize the key bind part. When the provisioner sends the app key add, it no longer sends the key bind command. After the node receives the app key add, it performs the key bind action on all of its own models. For details, see the code enclosed in the PROVISION_FLOW_SIMPLE_EN macro.

In addition, in order to further simplify provision, the demo app does not need to execute the get composition data command, it directly queries the database to obtain all information of the corresponding composition data through the PID in the device UUID obtained by provision. The following figure shows the function of firmware to write PID to device uuid.

```
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
}
```

Figure 15.1: Write PID to device uuid

If the customer does not want to obtain the composition data by querying the database, he/she can also modify the flow of the app and add the get composition data command.

15.2 Configuration

Node Firmware: set PROVISION_FLOW_SIMPLE_EN to 1.

15.3 Function Demonstration

15.3.1 Firmware Configuration

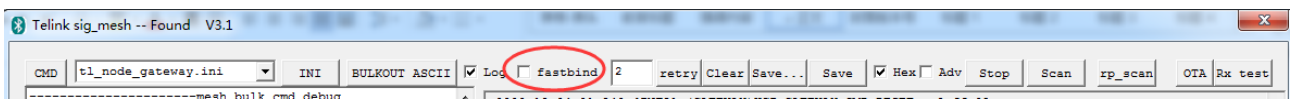


Figure 15.2: Firmware Configuration

15.3.2 APP Interface Configuration

Enable this option in the app:

Homepage – setting – settings – Enable Private Mode（Default Bound）

16 Private Fast provision Function

16.1 Function Introduction

Remote provision is done node by node, when the network is big, the provision time is still too long, to solve this problem, we provide this private fast provision function, i.e., in the default key network, add vendor commands, e.g., VD_MESH_RESET_NETWORK, and send network key, app key, iv index to target address 0xffff(send only once, and the whole network can receive it), then assign unicast address one by one according to mac. In this way, user can provision nodes within multiple hops. Device keys are generated based on mac address according to a certain rule, no need to assign by mesh commands.

User can also add un-provisioned new devices into an existing mesh network by fast provision way.

16.2 Configuration

Set FAST_PROVISION_ENABLE to 1. Compile 8258_mesh project, download to 3(more than 1)8258 dongles.

GATT master dongle mode and APP support fast provision function by default. The gateway V3.3.4 and later versions support this feature, which is disabled by default.

16.3 Function Demo

The following demo shows how to add multiple（more than 2）8258 nodes into mesh network at the same time for Gateway.

1) Power up the 8258 mesh node.
2) Set FAST_PROVISION_ENABLE to 1, compile the 8258_mesh_gw compilation option to get 8258_mesh_gw.bin and burn it to the 8258 Dongle, that is, Gateway.
3) Plug the Gateway into the USB port, open "SIG_MESH_TOOL" and select tl_node_gateway.ini, the title bar will show Found indicating that the gateway device is found (Provisioner). The tool will automatically get the uuid and mac address of the gateway, and the format of the commands sent is detailed in the "Provisioner (Gateway) Project Introduction" in the "Provisioner Lighting" section.

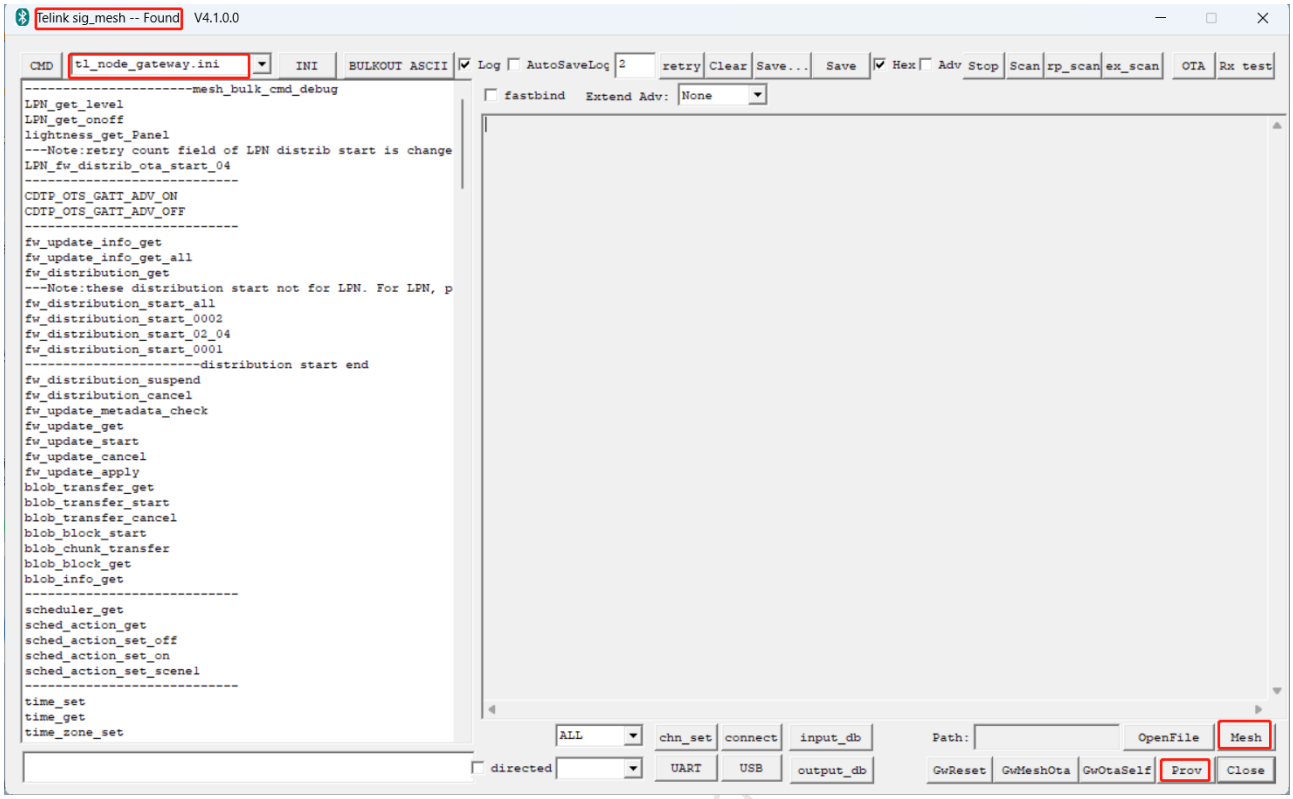


Figure 16.1: Normal connection

4) Click the "Prov" button at the bottom right corner to enter the provision interface, select Fast prov mode, set the network parameters and then click SetPro Internal to configure the provision data of Gateway. For the command format of "Prov" and "SetPro Internal", please refer to "Provisioner (Gateway) Project Introduction", "Provisioner Lighting".

5) Add the appkey of the Gateway: since there is no binding process in the fast provision, if there is no app key added to the Gateway before, you need to add the app key to the Gateway, the corresponding command format is:

HCI_CMD_GATEWAY_CMD + netkey index + appkey index + retry cnt + response max + destination + op + par.

that is: e8 ff + 0x0000 + 0x0000 + 0x00 + 0x01 + gateway address + 0x00 + netkey appkey index(3 bytes) + appkey(16 bytes).

Figure 16.2: Click in order

6) Click the "Provision" button to start the network provision.

The corresponding command format of Provision control is HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_FAST_PROV_ + pid(2 bytes) + new device address(2 bytes).

That is: e9 ff + 0x17 + pid + new device address.

Note: The device type to be added can be specified by PID. If all device types are to be added, PID is set to 0xffff.

7) The gateway reports the address assigned to the device during fast provision to sig_mesh_tool.exe of PC. The report format is:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_SEND_NODE_INFO + VC_node_info_t.

That is: 0x91 + 0x81 + VC_node_info_t. For details of the VC_node_info_t format, please refer to the "Provisioner Lighting" section in the "Introduction to the Provisioner (Gateway) Project".

Note: Since the valid parameter length of the vendor message is only 8 bytes, the device only returns 6 bytes of mac and 2 bytes of PID during the gateway scanning, and the gateway side will get the number of elements of the device according to the PID, please refer to mesh_fast_prov_get_ele_cnt_callback(u16 pid) for details.

8) After fast provision is completed, you will see all nodes are blinking 3 times. The gateway reports to sig_mesh_tool.exe of PC with successful binding event. The report format is:

TSCRIPT_GATEWAY_DIR_RSP + MESH_KEYBIND_EVE_SUC + event.

That is: 0x91 + 0x8a + 0x01.

9) Click the "Mesh" button to enter the mesh window to turn on/off the lights.

17 Private online status function demo

17.1 Function Introduction

Currently, the online and offline monitoring mechanism provided by the SIG mesh spec can be implemented through the heartbeat and publish mechanisms. The real-time monitoring of status such as onoff of nodes can be implemented through the publish mechanism.

When both online and offline monitoring and real-time monitoring of status are required, a publish mechanism is required. However, the publish mechanism has the following limitations:

- Publish messages generally need to set relay, so there will be more packets on air.
- The publish period cannot be set too short (it usually takes tens of seconds or longer), otherwise there will be too many packets on air, affecting normal control.
- Sometimes it takes several publish status messages to include all the statuses that need to be reported. At this time, there will be more packets on air.

Therefore, we have added an online status mechanism, the purpose of which is to achieve fast and effective online and offline detection, and to report important status of nodes, while also effectively reducing data packets in the network.

17.2 Configuration

Change the ONLINE_STATUS_EN of app_config.h of both mesh and mesh_provision project from 0 to 1.

17.3 Packet Format

The online status data packet is sent by adv of ADV NON CONN IND, and the type of the payload is customized MESH_ADV_TYPE_ONLINE_ST (0x62), as shown in the figure below.

Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
				Type	TxAdd	RxAdd	PDU-Length					
+319991	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	0x000080	-38	OK
=1599997									64 13 B8 41 57 93 BA 1C 6B 9E DD 91 9F AB 4A			
Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
				Type	TxAdd	RxAdd	PDU-Length					
+360010	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	0x000076	-38	OK
=1960007									64 13 B8 41 57 93 BA 1C 6B 9E DD 7E 1D 3E 25			
Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
				Type	TxAdd	RxAdd	PDU-Length					
+319994	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	0x000084	-38	OK
=2280001									64 13 B8 41 57 93 BA 1C 6B 9E DD 0C C1 5E 66			

Figure 17.1: Packet format

The effective payload length is 24 bytes. By default, each node requires 6 bytes. For details, please refer to

```
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;
```

Figure 17.2: reference details

The effective length of each node is MESH_NODE_ST_PAR_LEN (3). The specific data filled can be customized according to each product. The default fill BYTE 0 is brightness, BYTE 1 is the color temperature value, and BYTE 2 is reserved. Modify device_status_update () according to customer needs.

```
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));
}
```

Figure 17.3: device_status_update

Note: MESH_NODE_ST_PAR_LEN (3), can be modified, but the larger the length, the slower the transmission speed. And there is no length field to describe this length, so when defining the network, you must first determine the value of MESH_NODE_ST_PAR_LEN. If the MESH_NODE_ST_PAR_LEN values configured by nodes in the network are inconsistent, there will be compatibility issues, resulting in data format parsing errors.

17.4 SIG_MESH_TOOL Firmware Demo

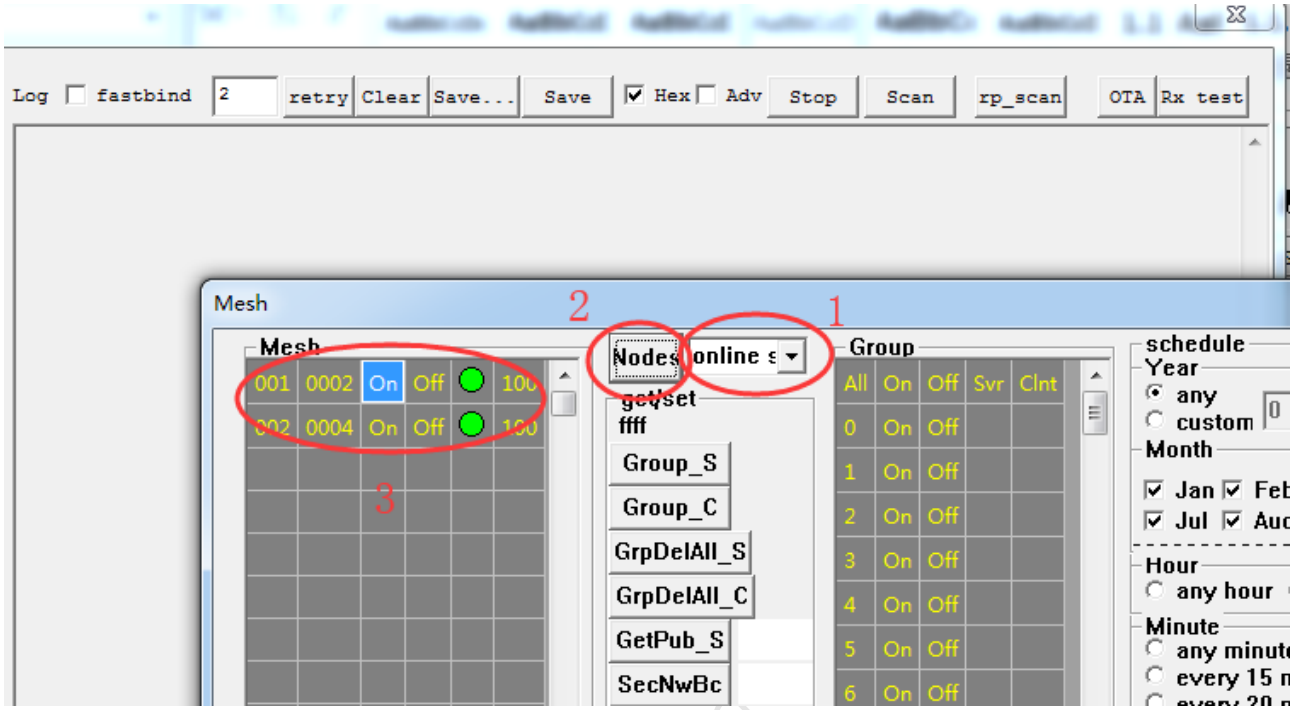


Figure 17.4: SIG_MESH_TOOL firmware demo

Step 1 Choose online status as shown in figure above.

Step 2 Click Node button as shown in figure above. After clicking, you can see from the log window that you did not send similar commands such as lightness get. Instead, you can directly obtain the online status data in the directly connected node through the custom UUID.

Step 3 After clicking the Node button, the node display window on the left shows the node information of the current network.

18 Telink Proprietary OTA Test Brief

18.1 GATT master dongle OTA for firmware update of BLE directly connected nodes

This mode is a point-to-point BLE Direct Connect OTA.

1) Download the BIN file (New FW) that requires OTA to the flash address of 8269 master dongle starting from 0x20000 according to the instructions in 7.1, burn 8269_mesh_master_dongle.bin from address 0x0000.

Note: The difference of burning New FW:

BDT tool, please refer to the following steps:

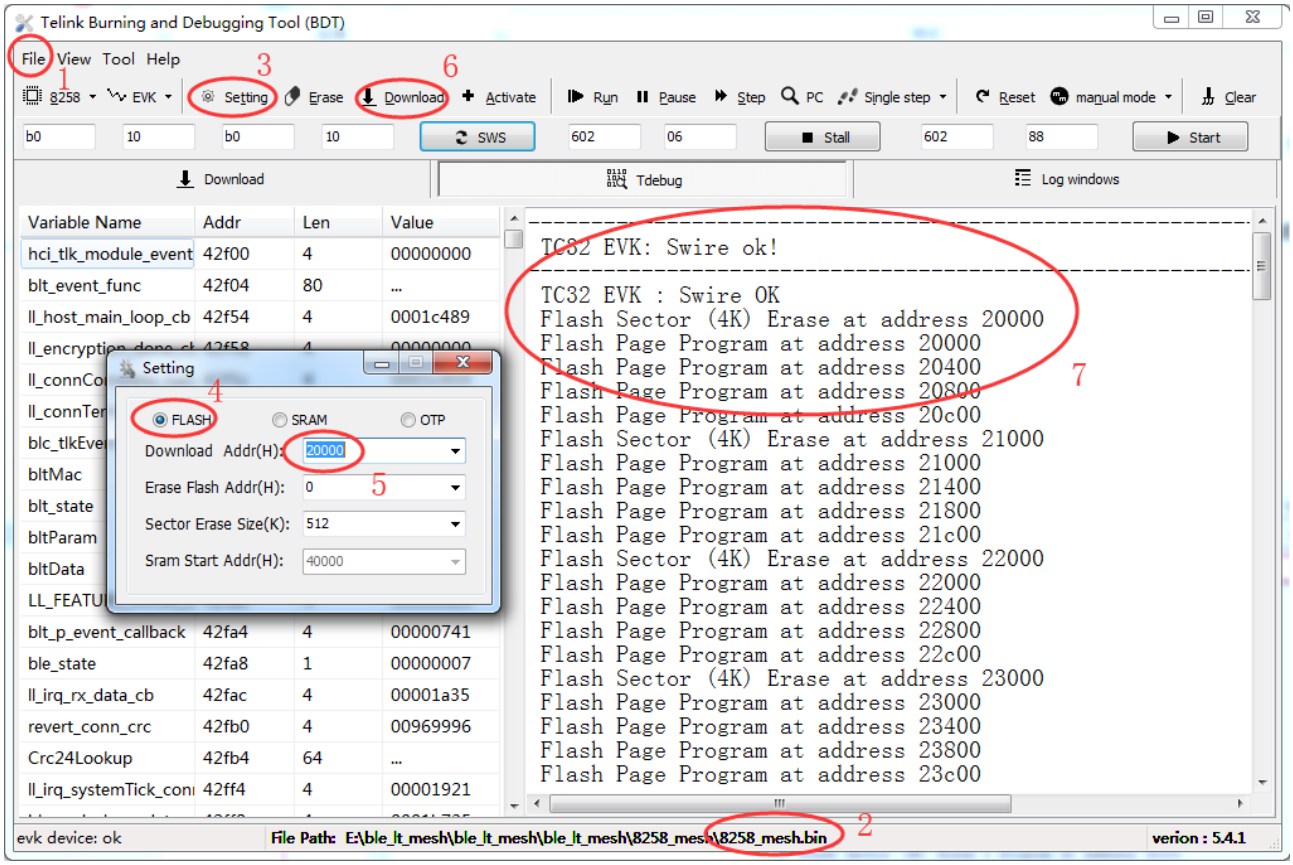


Figure 18.1: BDT tool

For wtcdb tool, click the "WF20000" button to start burning. Please refer to the following steps:

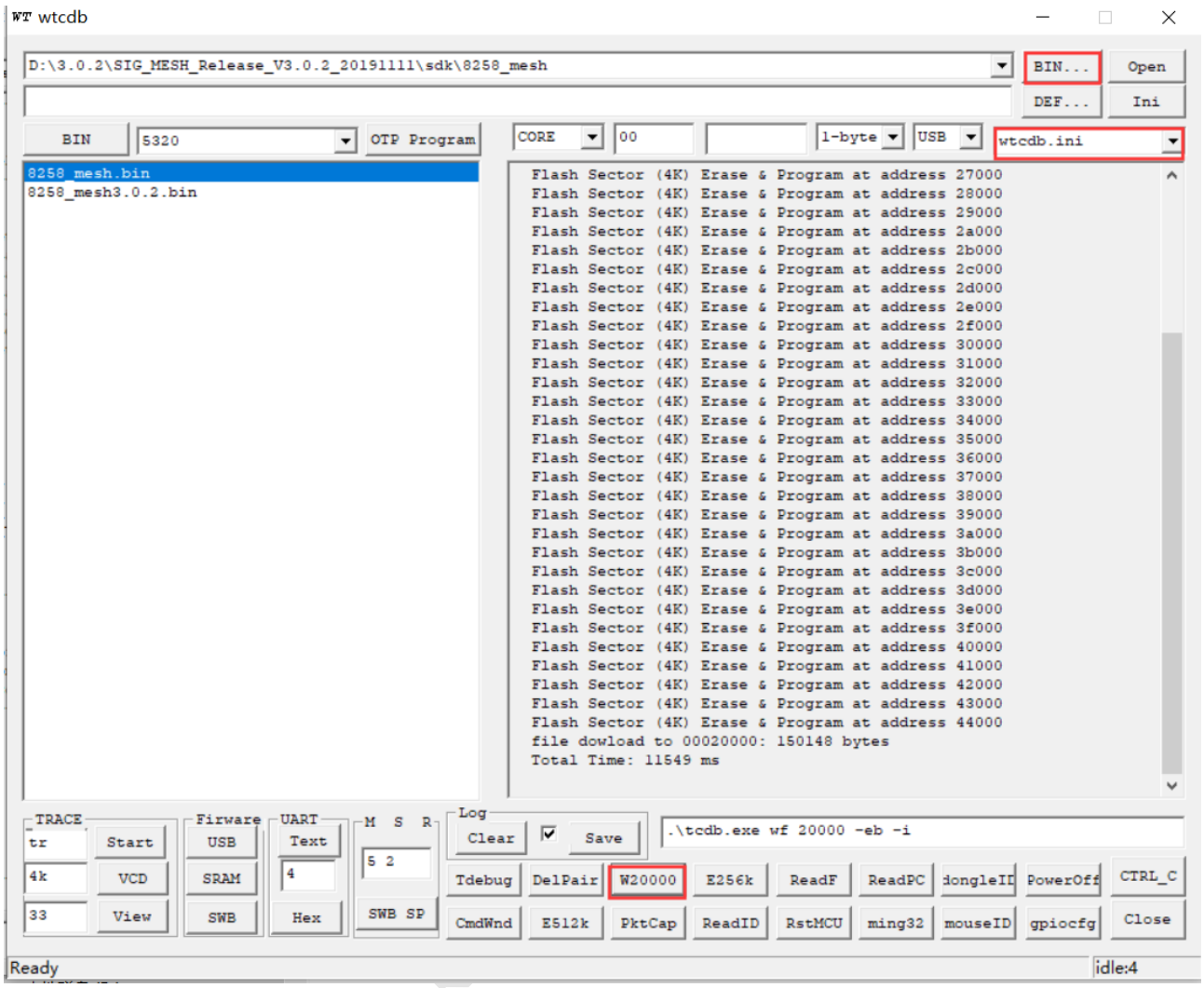


Figure 18.2: wtcdb tool

2) Refer to the steps (1) to (5) in Section 4.4 to establish a BLE connection between the target node and the tool.

3) Click the OTA button to start the OTA process. If the OTA is completed normally, the node will flash 8 times continuously.

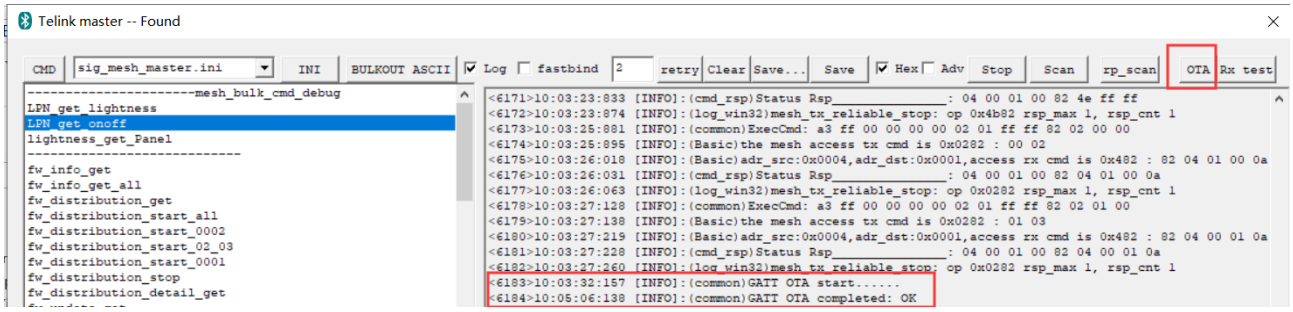


Figure 18.3: Start OTA

4) For the commands of the OTA part and the details of the protocol part, please refer to Chapter 6.4

of "AN_17092701_Telink 826x BLE SDK Developer Handbook". There is a detailed description of the command and protocol format.

18.2 OTA OTA where the Gateway node updates its firmware

The purpose of Gateway Gate OTA is to upgrade the firmware of gateway itself.

1) Open the BDT tool and download 8258_mesh_gw.bin to 8258 dongle.
2) "Found" in the upper left corner of the tool means that the 8258 Dongle and the PC tool are normally connected and can communicate normally.
3) Click the search_file button in the lower right corner and select a different version of the 8258_mesh_gw.bin file;

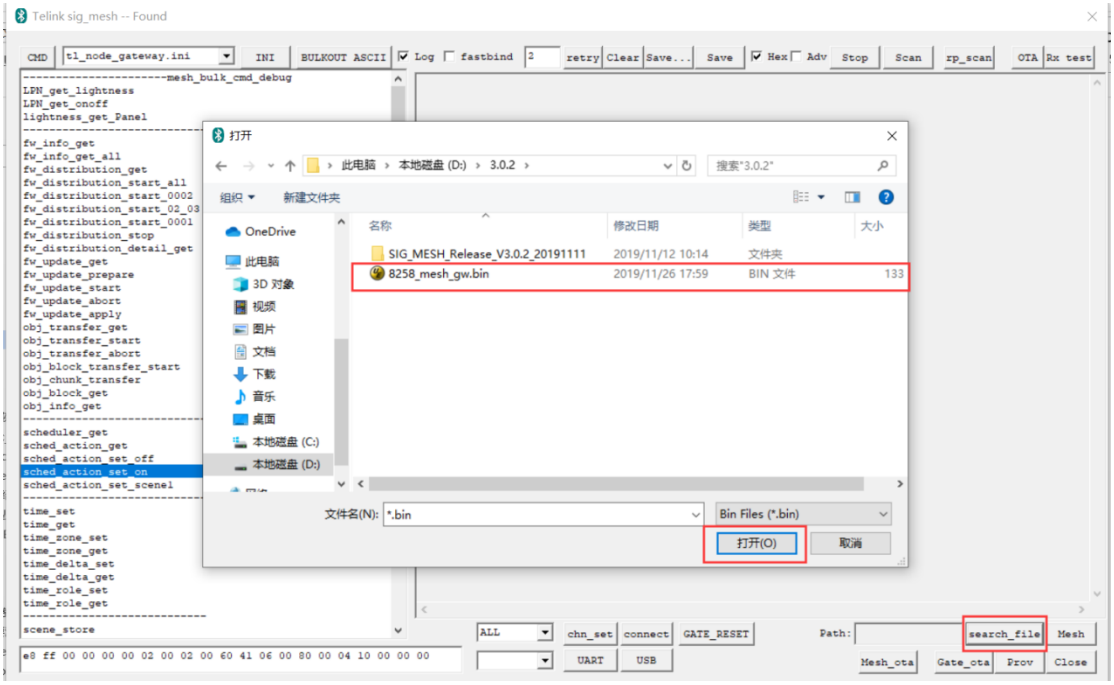


Figure 18.4: Select bin file

4) Click the Gate_ota button to start the upgrade. LOG prompts gateway firmware load suc to indicate that the upgrade was successful. After the upgrade is successful, the gateway will automatically restart and enable the new firmware.

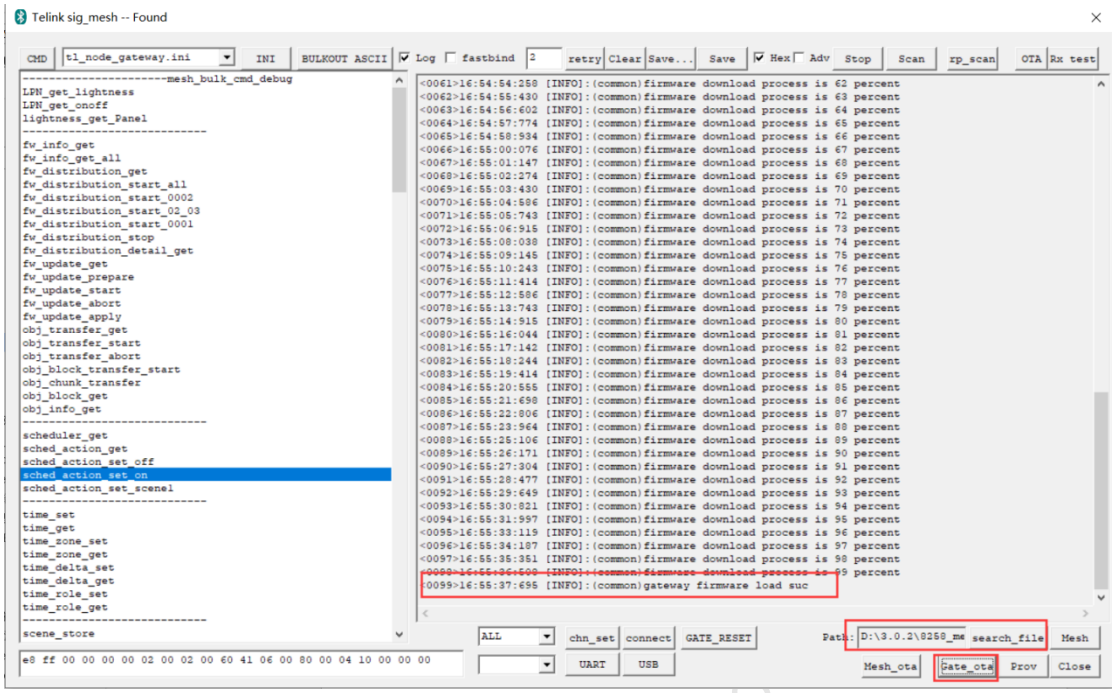


Figure 18.5: Click Gate_ota

19 Network Sharing

For more information, you can also refer to the section 33.5.1.2 Share Export of the chapter Android and iOS APP User Guide.

19.1 Share Mode of App share from Gateway or GATT Master Dongle

Step 1 Use VC master or gateway networking to make sure that the 8258 dongle nodes can be properly networked and controlled;

Step 2 Click on "output_db" on the homepage; (you can't copy and paste the JSON file directly because you have to remove some added fields like iv index, etc.)

Step 3 Import the output json file into TelinkSigmesh APP.

IOS APP Steps:

Step 1 Connect your phone to a computer with iTunes installed.

Step 2 Click the phone icon in the upper left corner of iTunes to enter the iTunes device details interface.

Step 3 Select "File Sharing" on the left side of iTunes, then find and click the demo APP "TelinkSigMesh" in the app, and wait for iTunes to load the file.

Step 4 After the file is loaded, drag the json file on your computer into the "TelinkSigMesh" document on the right

Step 5 Click the IMPORT button in the APP to select the JSON file to load. Detailed steps are in the pictures below:

a) Open APP TelinkSig mesh, click Setting button.

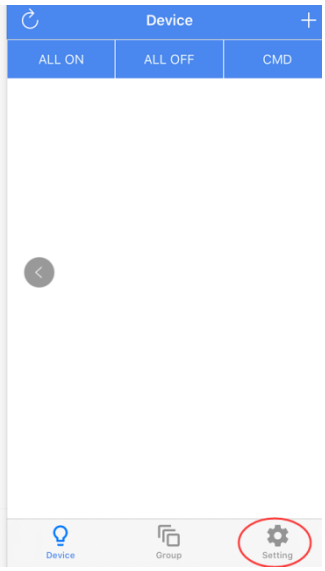


Figure 19.1: Click Setting

b) Click Share button

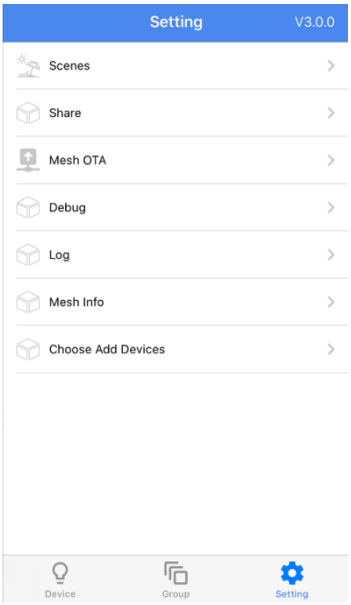


Figure 19.2: Click Share

c) Click IMPORT button

Figure 19.3: Click IMPORT

d) Click mesh.json, click IMPORT

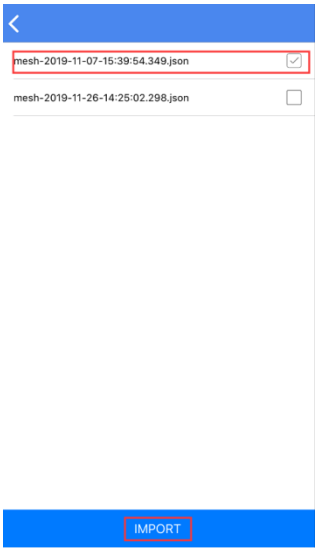


Figure 19.4: Click mesh.json

Import file with Android APP:

Step 1 Connect the phone to computer, import the mesh.json file into any folder on the phone, and remember the path to the folder.

a) Open APP TelinkSigmesh, click Setting button.

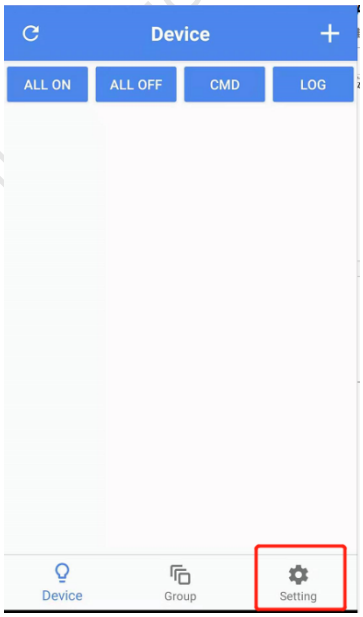


Figure 19.5: Click Setting

b) Click Share button 

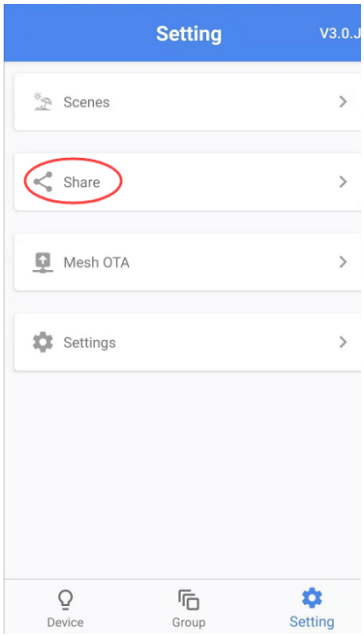


Figure 19.6: Click Share

c) Click IMPORT button

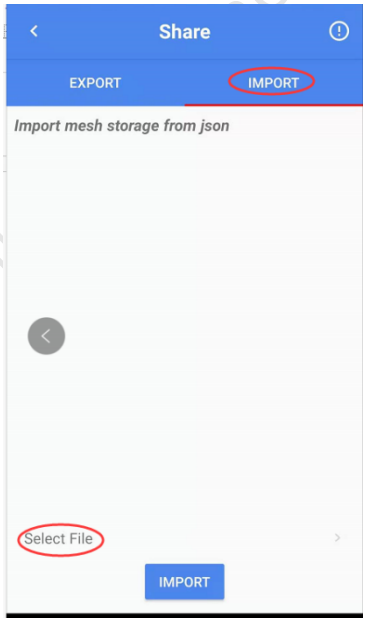


Figure 19.7: Click IMPORT

d) Click Select File to select mesh.json imported from the computer

Figure 19.8: Click mesh.json

e) Click Import button 

Step 2 After the import is successful, return to the main page: APP Device interface. At this time, the shared nodes will be displayed. These nodes are the nodes of the VC tool network. APP can be controlled, and both VC and APP can control the nodes.

19.2 Share Mode of Gateway or GATT Master Dongle share from App

1) Provision with iOS or Android TelinkSIGmesh APP, and works normally.
2) Click Setting button, then click Share button to enter share interface, click EXPORT button, generate JSON file. The path of JSON file folder will be displayed on the interface.

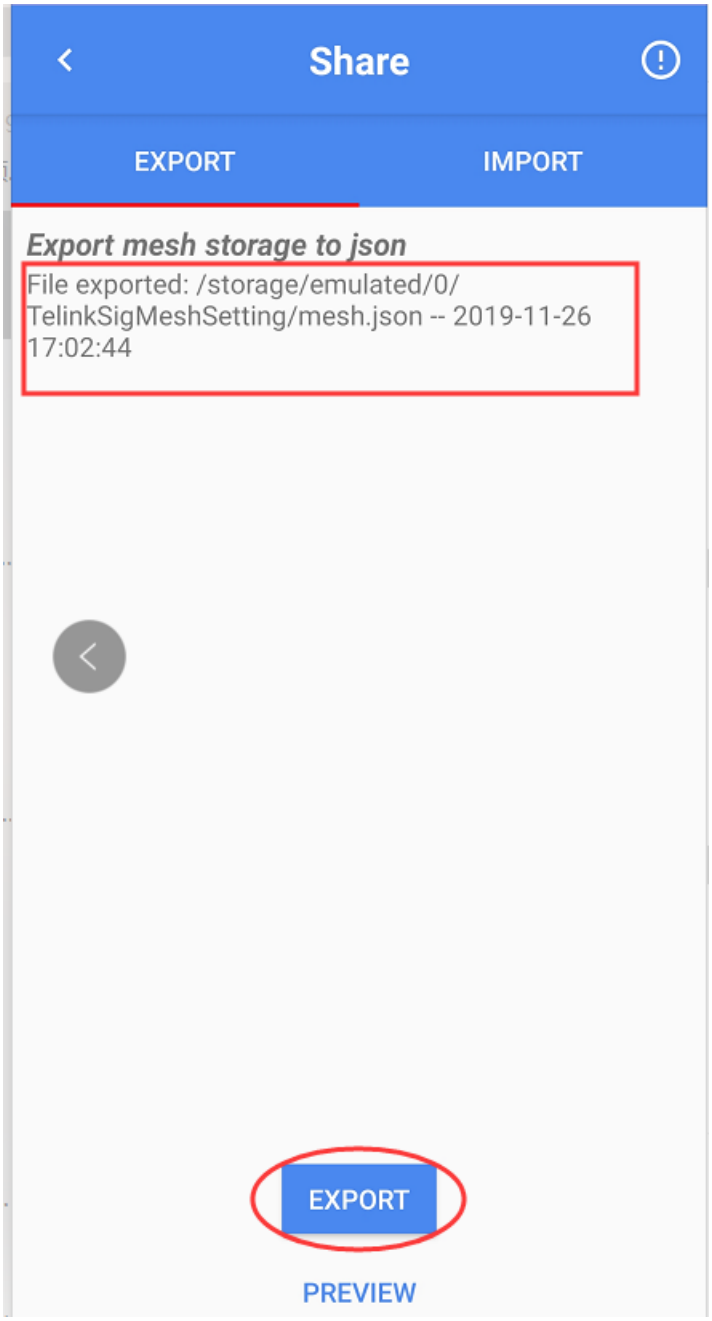


Figure 19.9: JSON file path

3) Connect the master 8269 Dongle or gateway dongle to the PC.
4) Open the "SIG_MESH_TOOL" tool, then it will show that master 8269 Dongle and PC tool are connected normally or gateway 8258 Dongle and PC tool are connected normally.
5) Click the "input_db" button to import the mesh.json generated by the app.(You can't copy and paste the JSON file directly, because you have to clear the node max, etc. inside the ini file, or else the address space of the unicast address will be wasted, etc.)

Note: The Gateway dongle is generally one that has not yet been configured for the network, if it has been configured, the information in it will be deleted and the imported data will be used.

6) Click mesh button, the imported nodes will show in the mesh window, and can be controlled.

7) Not the sharing is completed, gateway/master dongle and APP can all control the nodes.

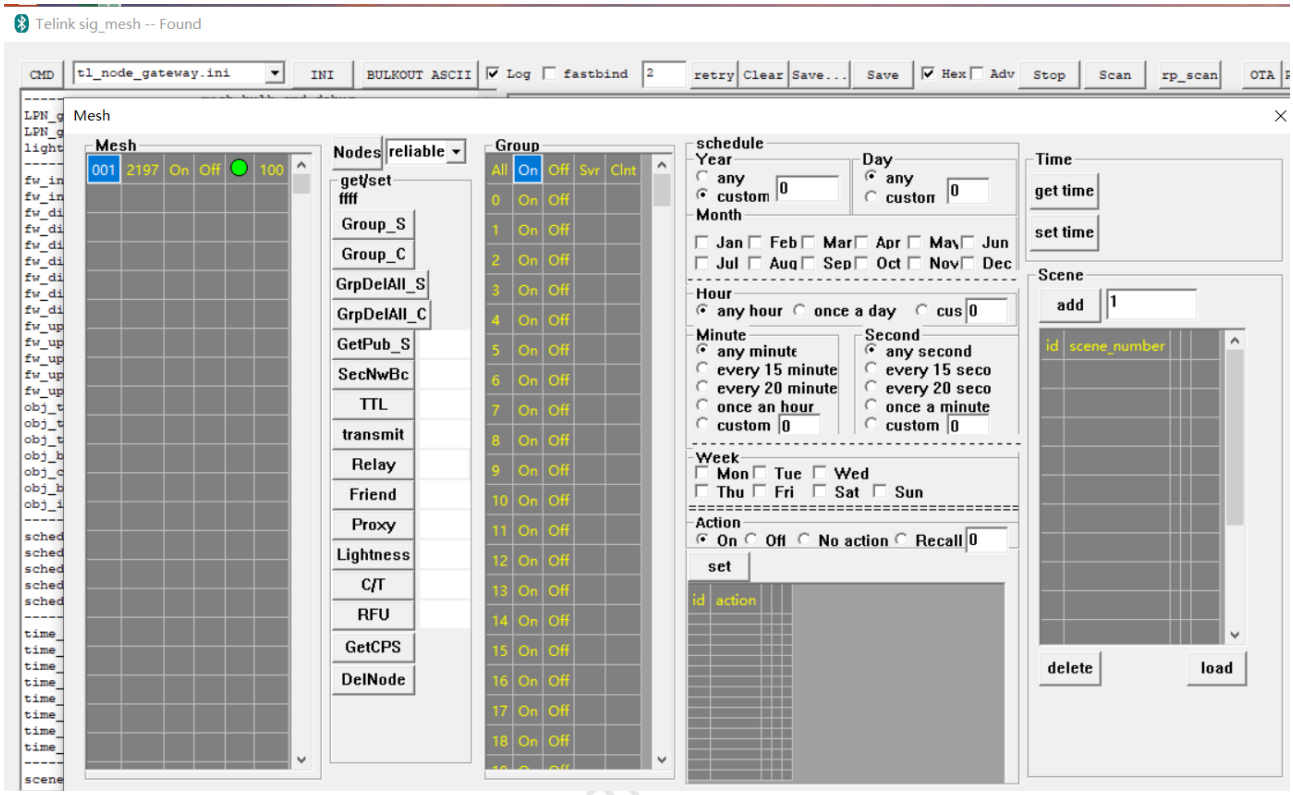


Figure 19.10: Complete sharing

20 Control Nodes via INI Demo

20.1 Provision Device

Device provision is slightly different for PB-GATT and PB-ADV, but it is the same for interface and operation procedure.

Step 1 Scan UNprovision_beacon adv devices;

Step 2 Connect according to MAC of the scanned UNprovision adv devices;

Step 3 Provision device;

Step 4 Bind model.

The following demo is to test with master dongle. Click stop, then click scan to enter scan mode, connect the scanned device, with the mac of: 112233445566.

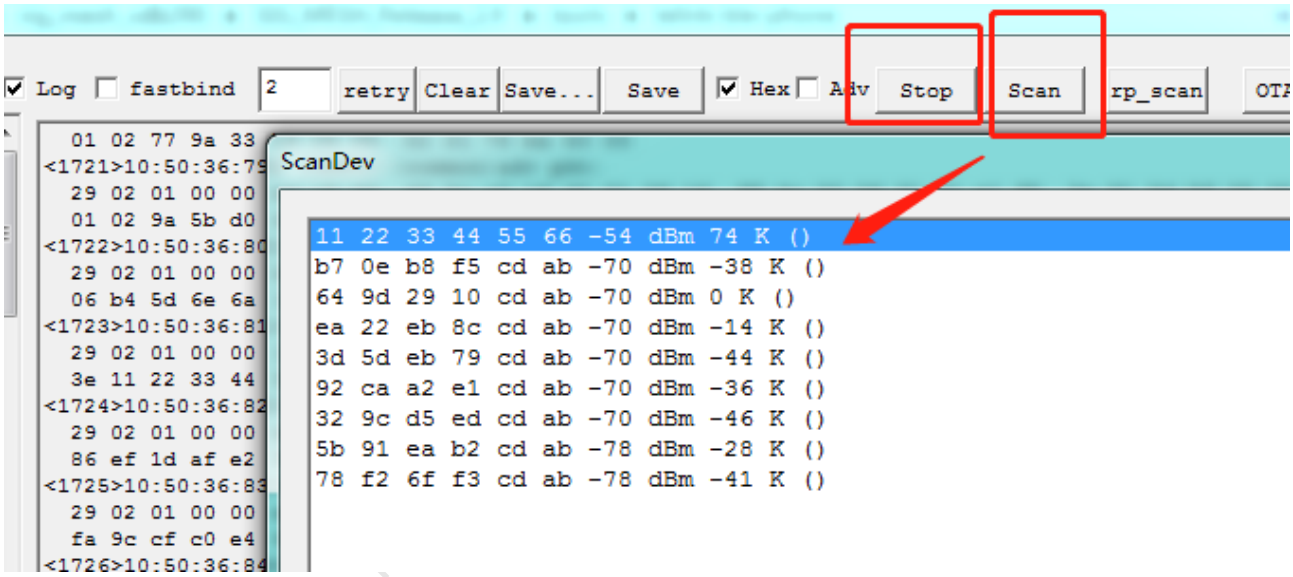


Figure 20.1: Connect device

Log information after the device is connected:

```
  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
```

Figure 20.2: log info

Provision Parameter Setting and Device Provision

1) Click prov button to enter provision interface, first click SetPro_internal to assign net key to dongle.
2) Click provision button to provision the connected device with MAC of 112233445566, during provision, assign netkeyindex，IVindex，unicast_addr generated by firmware to device.
3) Click bind_all button to bind APPkey(based on the model reported by devicecomposition data).

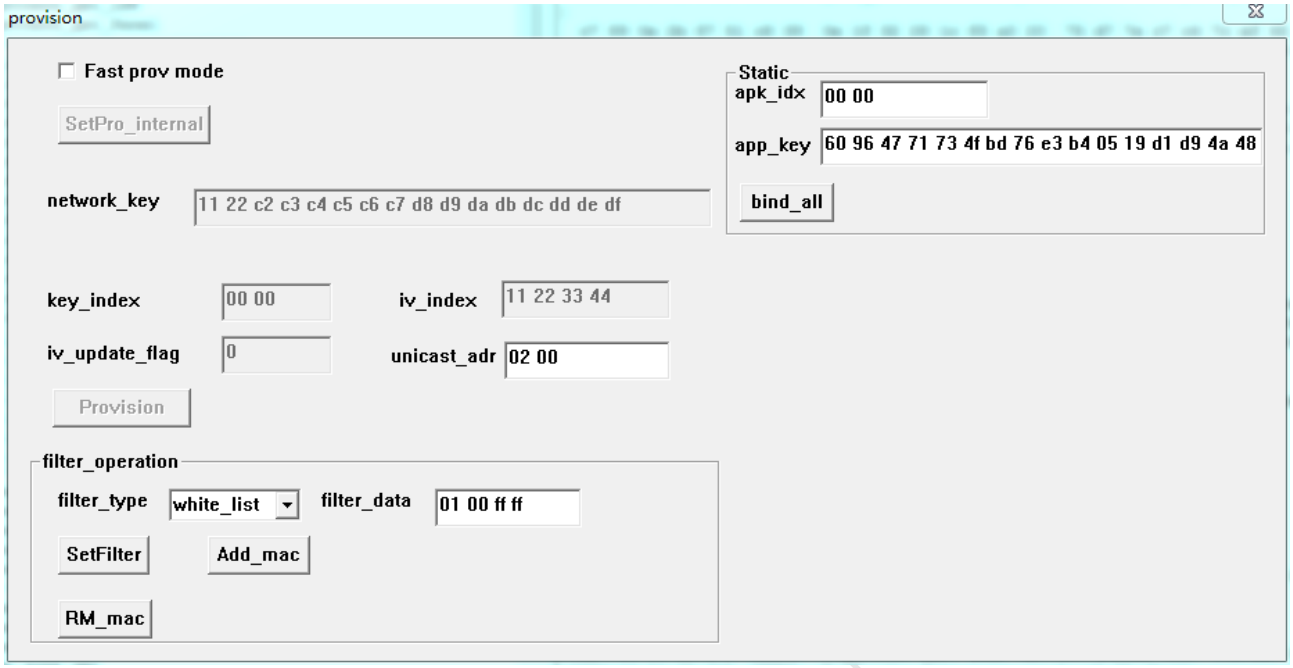
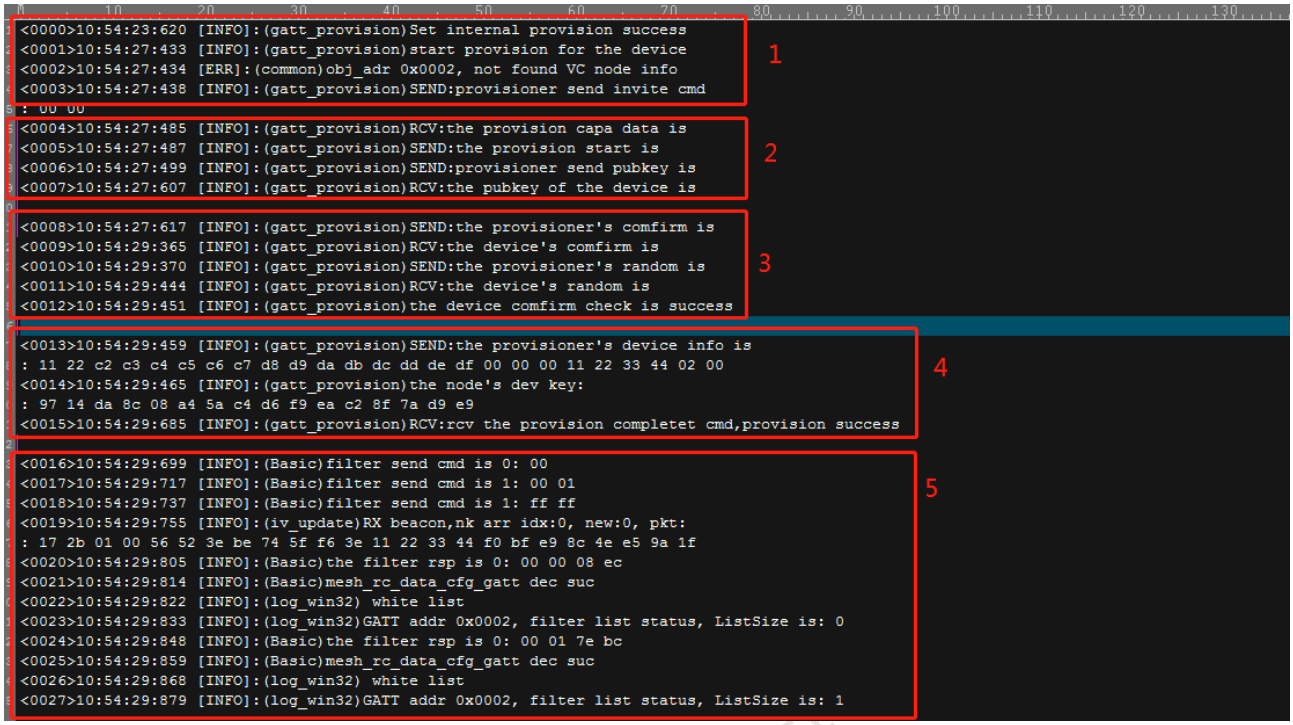


Figure 20.3: Provision parameter setting and device provision

As shown in figure below, the data interaction of Provision is described as following:

1) Start provision, provisioner send out data
2) public key interaction
3) check confirm;
4) send provision data (net_key/nkey_index/IV_update_flag/IV_index/unicast_addr);
5) Add proxy white list




```
<0000>10:54:23:620 [INFO]:(gatt_provision)Set internal provision success
<0001>10:54:27:433 [INFO]:(gatt_provision)start provision for the device
<0002>10:54:27:434 [ERR]:(common)obj_adr 0x0002, not found VC node info
<0003>10:54:27:438 [INFO]:(gatt_provision)SEND:provisioner send invite cmd
: 00 00
<0004>10:54:27:485 [INFO]:(gatt_provision)RCV:the provision capa data is
<0005>10:54:27:487 [INFO]:(gatt_provision)SEND:the provision start is
<0006>10:54:27:499 [INFO]:(gatt_provision)SEND:provisioner send pubkey is
<0007>10:54:27:607 [INFO]:(gatt_provision)RCV:the pubkey of the device is

<0008>10:54:27:617 [INFO]:(gatt_provision)SEND:the provisioner's comfirm is
<0009>10:54:29:365 [INFO]:(gatt_provision)RCV:the device's comfirm is
<0010>10:54:29:370 [INFO]:(gatt_provision)SEND:the provisioner's random is
<0011>10:54:29:444 [INFO]:(gatt_provision)RCV:the device's random is
<0012>10:54:29:451 [INFO]:(gatt_provision)the device comfirm check is success

<0013>10:54:29:459 [INFO]:(gatt_provision)SEND:the provisioner's device info is
: 11 22 c2 c3 c4 c5 c6 c7 d8 d9 da db dc dd de df 00 00 00 11 22 33 44 02 00
<0014>10:54:29:465 [INFO]:(gatt_provision)the node's dev key:
: 97 14 da 8c 08 a4 5a c4 d6 f9 ea c2 8f 7a d9 e9
<0015>10:54:29:685 [INFO]:(gatt_provision)RCV:rcv the provision completet cmd,provision success

<0016>10:54:29:699 [INFO]:(Basic)filter send cmd is 0: 00
<0017>10:54:29:717 [INFO]:(Basic)filter send cmd is 1: 00 01
<0018>10:54:29:737 [INFO]:(Basic)filter send cmd is 1: ff ff
<0019>10:54:29:755 [INFO]:(iv_update)RX beacon,nk arr idx:0, new:0, pkt:
: 17 2b 01 00 56 52 3e be 74 5f f6 3e 11 22 33 44 f0 bf e9 8c 4e e5 9a 1f
<0020>10:54:29:805 [INFO]:(Basic)the filter rsp is 0: 00 00 08 ec
<0021>10:54:29:814 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<0022>10:54:29:822 [INFO]:(log_win32) white list
<0023>10:54:29:833 [INFO]:(log_win32)GATT addr 0x0002, filter list status, ListSize is: 0
<0024>10:54:29:848 [INFO]:(Basic)the filter rsp is 0: 00 01 7e bc
<0025>10:54:29:859 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<0026>10:54:29:868 [INFO]:(log_win32) white list
<0027>10:54:29:879 [INFO]:(log_win32)GATT addr 0x0002, filter list status, ListSize is: 1
```

Figure 20.4: The data interaction of Provision

Bind introduction:

Bind is to bind APP key with all device models according to the composition data of the device after the provision. This process will determine the bind time according to the number of models (only 20s). Therefore, bind has been optimized on Tmall Genies and other platforms, and fastbind will be introduced next.

Before Bind, you need to get the composition data of the device first. For detailed analysis format of Composition data, please refer to <4.2.1 Composition Data> of <Mesh_v1.0>.

```
<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
```

Figure 20.5: Get device composition data

According to the corresponding information of the obtained element and model, the bind command is subsequently issued to bind model by model.

```
<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
<0345>19:20:54:096 [INFO]:(Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 03 00
<0346>19:20:54:192 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0347>19:20:54:202 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 3e 00 02 00 00 00 03 00
<0348>19:20:54:219 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0349>19:20:54:231 [INFO]:(KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0004
<0350>19:20:54:245 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 04 00
<0351>19:20:54:258 [INFO]:(Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 04 00
<0352>19:20:54:352 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0353>19:20:54:368 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 3e 00 02 00 00 00 04 00
<0354>19:20:54:384 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0355>19:20:54:396 [INFO]:(KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0xfe00
<0356>19:20:54:407 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 00 fe
<0357>19:20:54:418 [INFO]:(Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 00 fe
<0358>19:20:54:512 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0359>19:20:54:527 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 80 3e 00 02 00 00 00 00 fe
<0360>19:20:54:547 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
```

Figure 20.6: Bind

```
<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
```

Figure 20.7: Bind

20.2 Configuration Operations

20.2.1 Key add/bind Operation

APPKey add command format analysis:

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		

Figure 20.8: APPKey add command

Command format analysis:

- Flag: Defined by Telink, to identify the header packet of USB or UART communication, UART is E8FF, USB is A3FF.
- NK_idx: network key index
- Ak_idx: APP key index
- Reliable retry cnt: Application layer retry (the number of resends if the firmware cannot receive a reply after issuing a command)
- Reliable resp_max: set the number of nodes to reply
- Dst: destination address filling
- Op: standard command code defined by sig mesh specification, please refer to <Mesh_v1.0>, if the command code is not fixed, please refer to <4.3.4 Messages summary> in the document.

[9:12]: Transmission parameter part. Please refer to <4.3.2.37 Config AppKey Add> in <Mesh_v1.0> for filling data.

Field	Size (octets)	Notes
NetKeyIndexAndAppKeyIndex	3	Index of the NetKey and index of the AppKey
AppKey	16	AppKey value

Figure 20.9: Filling data

Key bind command format analysis:

CMD-cfg_appkey_bind_001 = a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 00 10

Command format analysis:

- Flag: Defined by Telink, to identify the header packet of USB or UART communication, UART is E8FF, USB is A3FF.
- NK_idx: network key index
- Ak_idx: APP key index
- Reliable retry cnt: Application layer retry (the number of resends if the firmware cannot receive a reply after issuing a command)
- Reliable resp_max: set the number of nodes to reply
- Dst: destination address filling
- Op: standard command code defined by sig mesh specification, please refer to <Mesh_v1.0>, if the command code is not fixed, please refer to <4.3.4 Messages summary> in the document.

[9:12]: Transmission parameter part, light HSL control command filling data please refer to <4.3.2.46 Config Model App Bind> in <Mesh_v1.0>.

Transmission parameter format reference:

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

Figure 20.10: Transmission parameter format reference

20.2.2 Subscription Configuration

CMD-cfg_sub_add = a3 ff 00 00 00 00 00 01 02 00 80 1b 02 00 01 c0 00 10

Or please refer to group index control in 4.5.2.

20.2.3 Publish configuration

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

Or please refer to GetPub_S control button in 4.5.3

20.2.4 Relay/Friend Function Configuration

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
```

Figure 20.11: Relay

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
```

Figure 20.12: Friend

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
```

Figure 20.13: Proxy

Or Please refer to "Relay"，"Friend"，"Proxy" control buttons in section 4.5.3.

20.2.5 Heartbeat setting

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 Control Operations

20.3.1 Control Generic model Demo

Test Preparation

- Provisionner dongle (burn 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2#（burn mesh.bin)
- SDK no need to modify

Test Introduction

The test is to achieve the control of the Generic model, mainly to achieve the G_ONOFF_SET command test.

Test and calculate the response time of CMD send and ACK.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#.

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-g_on_03 = e8 ff 00 00 00 00 00 00 03 00 82 02 01 00

Step 5 The following information will show after firmware send successfully.

```
.. 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
```

Figure 20.14: Firmware send successfully

Step 6 As shown in above figure <0011>, the interval between CMD sending time and rsp is 199 – 075 = 124ms. Gateway and node are controlled in adv way, the ack reply time is different because of the network.

```
Status Rsp_____________: 03 00 02 00 82 04 00 01 0a
```

The corresponding structure is

```
typedef struct{
    u16 len; // length
    u16 src; // source address
    u16 dst; // destination address
    u8 data[ACCESS_WITH_MIC_LEN_MAX];   // access layer(op code, parameters)
}mesh_rc_rsp_t;
```

Step 7 If the destination address of the control is a multicast or broadcast address, the node will add a random delay before replying to the ACK after receiving the command, so that multiple device reply messages are avoided as much as possible. When broadcasting an address as shown in the figure below, the time interval between CMD and rsp is: <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
```

Figure 20.15: Broadcast address

20.3.2 CTL model

Test Preparation

- Provisionner dongle (burn 8269_mesh_gw.bin or 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn mesh.bin)
- SDK need to modify #define LIGHT_TYPE_SEL LIGHT_TYPE_CT

Test Introduction

The test is to achieve the control of the CTL model, mainly to achieve the LIGHT_CTL_SET command test.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#.

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-light_ctl_set = e8 ff 00 00 00 00 00 00 ff ff 82 5e 01 00 20 4e 00 00 00

The following information will show after firmware send successfully.

```
.. 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
```

Figure 20.16: Firmware send successfully

20.3.3 HSL model

Hsl model will allocate 3 element addresses after provision. The main element is used for lightness, generic model control and configuration model, element2 is for hue control, element3 is for saturation control.

Test Preparation

- Provisionner dongle (burn 8269_mesh_gw.bin or 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn mesh.bin)
- SDK need to modify #define LIGHT_TYPE_SEL LIGHT_TYPE_HSL

Test Introduction:

The test is to achieve the control of the HSL model, mainly to achieve the LIGHT_HSL_SET command test.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-light_hsl_set = a3 ff 00 00 00 00 00 00 ff ff 82 76 01 00 00 50 00 80 00

The following information will show after firmware send successfully

```
<0000>15:57:25:186 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 76 00 20 00 50 00 80 00 00
<0001>15:57:25:210 [INFO]:(GATEWAY) gateway mesh cmd sendback src:0001 dst:ffff,opcode is 7682: 00 20
<0002>15:57:25:506 [INFO]:(cmd_rsp)Status Rsp_____________: 02 00 01 00 82 78 00 20 00 50 00 80
<0003>15:57:25:515 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 02 00 01 00 82 78 00 20 00 50 00 80
```

Figure 20.17: Firmware send successfully

20.3.4 Vendor model

Self-defined OP operation

Test Preparation

- Provisionner dongle (burn 8269_mesh_master_dongle.bin)
- Firmware tool(sig_mesh_tool.exe)
- Dongle_2# (burn 8258_mesh.bin)
- SDK need to modify, please refer to test introduction

SDK Modify Introduction

1) Configure in vendor_model.h:

```
#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) Declear in vendor_model.c

```
{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) Add to main_loop.

Test Introduction

Telink SDK has the following limitation based on customer usage:

1) A friend node can habe 16 LPN at most, the default value is 2, user may modify MAX_LPN_NUM to 16.
2) The data of 1 LPN node cached by the Friend node can support long packets, and the maximum length of a long packet is 41 bytes. (When the APP sends data, the maximum parameter transmission is 41 bytes.)

20.3.5 Gateway Transmit Long Packet to LPN

Test Preparation

- Provisionner dongle (burn 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn LPN.bin)
- SDK need to modify, please refer to test introduction

Test Introduction

Telink SDK has the following limitation based on customer usage:

1) A friend node can have 16 LPN at most, the default value is 2, user may modify MAX_LPN_NUM to 16.
2) The data of 1 LPN node cached by the Friend node can support long packets, and the maximum length of a long packet is 41 bytes. (When the APP sends data, the maximum parameter transmission is 41 bytes.)

The SDK is modified as following:

1) Set DEBUG_SUSPEND =1 (no Low Power mode);
2) Add the following information in cb_vd_light_onoff_set().

```
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);
```

Figure 20.18: cb_vd_light_onoff_set

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time.

Step 2 Burn LPN node bin in dongle_2#

Step 3 Firmware add LPN into the network.

Step 4 Control LPN node by existing vendor_on/vendor_off command, LPN can be controlled normally.

Step 5 Lengthen vendor_on command transmission parameter to 41 byte, then send

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 Check debug_fn_reciver_data[64] with tdebug tool after firmware firmware send successfully.

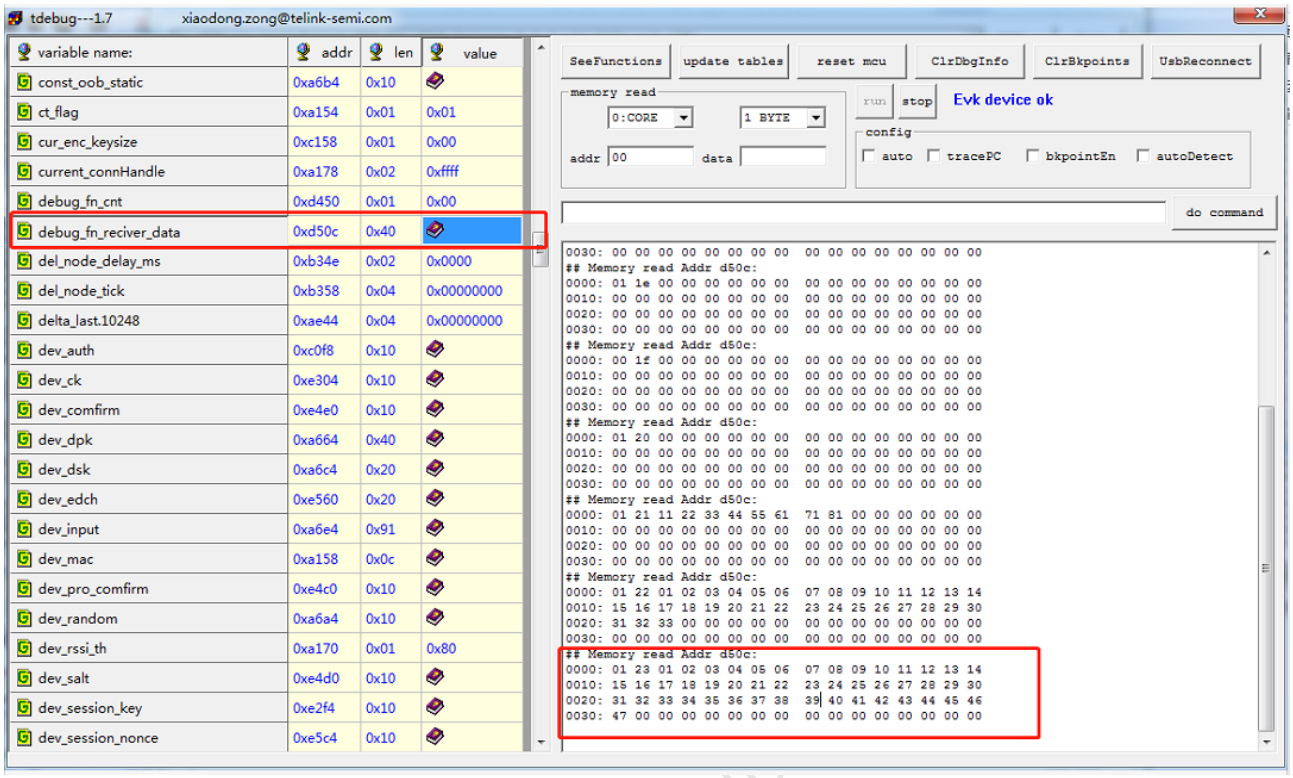


Figure 20.19: tdebug tool

21 Summary of mesh_1.1_feature

The summary content and profile of mesh V1.1 feature can be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/

The corresponding mesh spec can be downloaded here:

https://www.bluetooth.com/specifications/specs/?types=specs-docs&keyword=mesh&filter=

The link contains the following spec:

(1) Mesh protocol stack spec:

Mesh Protocol 1.1 Specification

The mesh protocol stack, which contains data communication format definitions, foundation models, and so on. The foundation models include Configuration model, Health model, Remote Provisioning model, Directed Forwarding Configuration model, Bridge Configuration model, Mesh Private Beacon model, On-Demand Private Proxy model, SAR Configuration model, Solicitation PDU RPL Configuration model, Opcodes Aggregator model, Large Composition Data model. These models are described in later sections. When you need to see the parameters of the command codes of these models, you need to refer to the "message" section in the "4 Foundation models" chapter of this document.

(2) Mesh product model spec:

Mesh Model 1.1 Specification

Mesh product models, such as the Generic OnOff model, Lightness model, and so on. When you need to see the parameters of the command codes for these models, you need to refer to the "message" subsection of each model chapter in this document.

(3) Mesh OTA model spec:

Mesh Binary Large Object Transfer Model Mesh Device Firmware Update Model

(4) NLC spec

NLC: Networked Lighting Control

Ambient Light Sensor NLC Profile Basic Lightness Controller NLC Profile Basic Scene Selector NLC Profile Dimming Control NLC Profile Energy Monitor NLC Profile Occupancy Sensor NLC Profile

(5) Device attribute definitions

Includes sensor ID, definition of sensor data format, introduction of brightness and time attributes in light control model.

Device Properties

(6) Mesh definition of import and export file formats for network sharing

Mesh Configuration Database Profile

22 Certify_base_provision_certificate Mode

22.1 Function

The certificate-based authentication provisioning mode can be judged by the provisioner and only nodes that meet the requirements can join the network.

The difference between this mode and the normal provisioning mode is that: when provisioning, the provisioner first obtains the certificate of the unprovisioned device, which contains the public key, uuid, and authentication information, etc. When the provisioner gets the certificate, it will judge whether it is legal or not, and if it is legal, it will start to group the nodes. When provisioning, the provisioner does not need to obtain the public key of the other party, and the other processes are the same as those in the normal provisioning mode. The judgement of whether the certificate is legal or not is made only on the provisioner side.

An overview of the functions can also be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/ This SIG is described on the official website as well as https://www.bluetooth.com/bluetooth-certificate-based-provisioning-a-technical-overview/.

22.2 Test Using the Code's Default Certificate and Compiling It Directly into Firmware

22.2.1 Code Configuration

- Open CERTIFY_BASE_ENABLE。
- CERTIFY_BASE_ENABLEcertify_base_crypto.c, The CERTIFY_TYPE inside is changed to CERTIFY_OOB_BY_DEFAULT_CERT.

Note:

> At this point, all nodes are using the same certificate, and the same device UUID.(Mac address may be different)

The scanning interface is as follows:

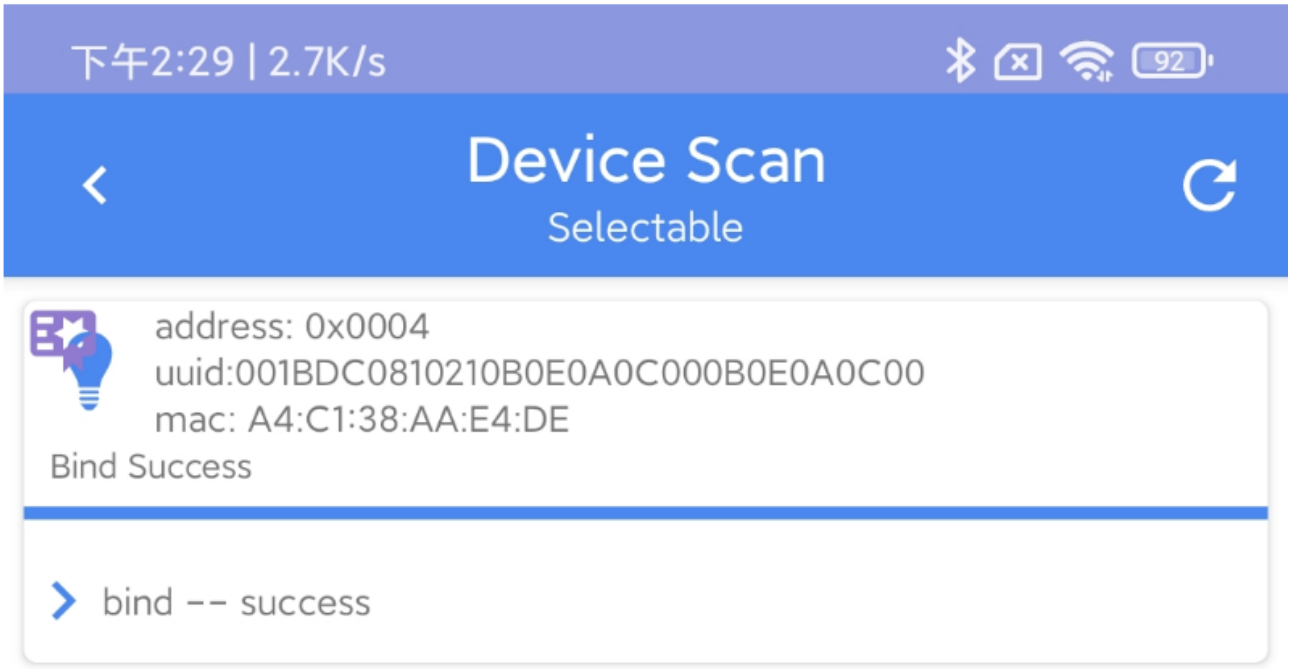


Figure 22.1: Certificate mode

Then directly click "Add" and other operations according to the normal network operation process, you can network and control.

22.3 Testing Ways to Use Newly Generated Certificates

22.3.1 Code Configuration

- Open CERTIFY_BASE_ENABLE
- certify_base_crypto.c The CERTIFY_TYPE inside remains the default CERTIFY_OOB_BY_READING_FLASH

Install git bash, if git is already installed on your computer. Note that the git bash version should be greater than or equal to version 2.41.0.

Note:

> To run bash, run the macOS or Linux command or the window git bash by typing . /xxx.bash.

22.3.1.1 Open a Git_bash Terminal

In the telink_sig_mesh_src/sig_mesh_tool/bash-certifybase directory, open the git bash terminal as follows:

right mouse button – Git Bash here

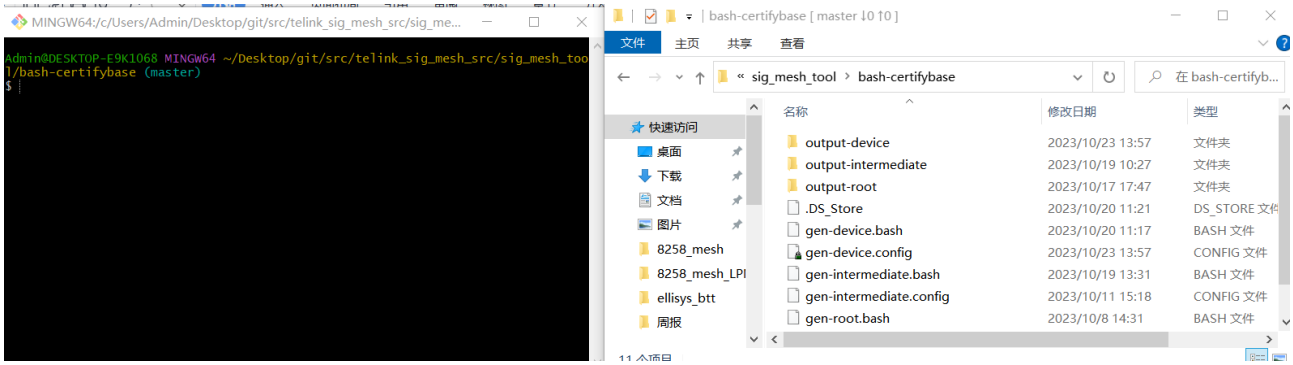


Figure 22.2: Open git bash

22.3.1.2 Generate Root Certificates

We have a default certificate in our sdk, it is not recommended to regenerate the root certificate, we suggest to skip this step directly. If you want to regenerate the root certificate, you need to import the certificate file root.der from output-root into the app, and set the newly imported certificate as the root certificate. The procedure is as follows:

- To run gen-root.bash to create a root certificate, run the command . /gen-root.bash.

Figure 22.3: Generate root certificates

- Import root certificate to APP

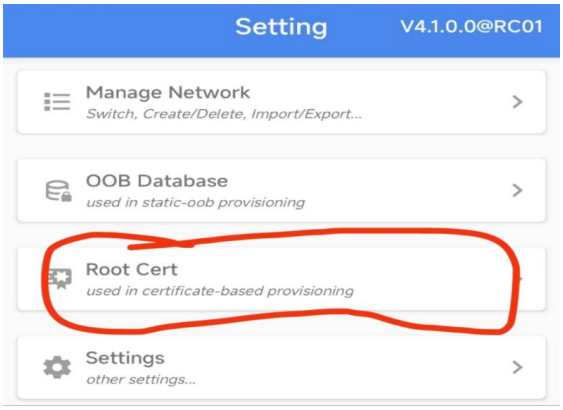


Figure 22.4: Go to the certificates page

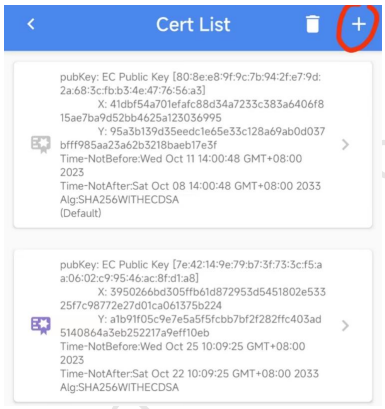


Figure 22.5: Importing a new certificate

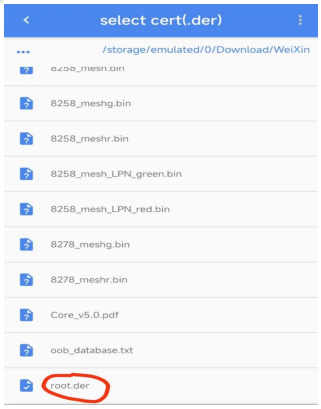


Figure 22.6: Importing a new certificate

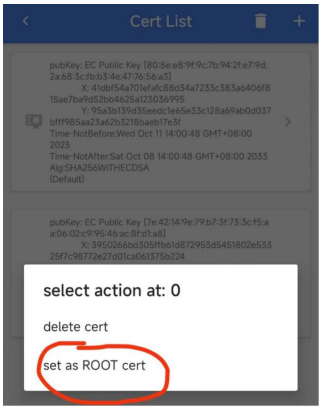


Figure 22.7: Set the newly imported certificate as root certificate

22.3.1.3 Run Gen-intermediate.bash to Create an Intermediate Certificate

The certificate is signed by the root certificate and the result is output in the output-intermediate directory.

Run the command: ./gen-intermediate.bash

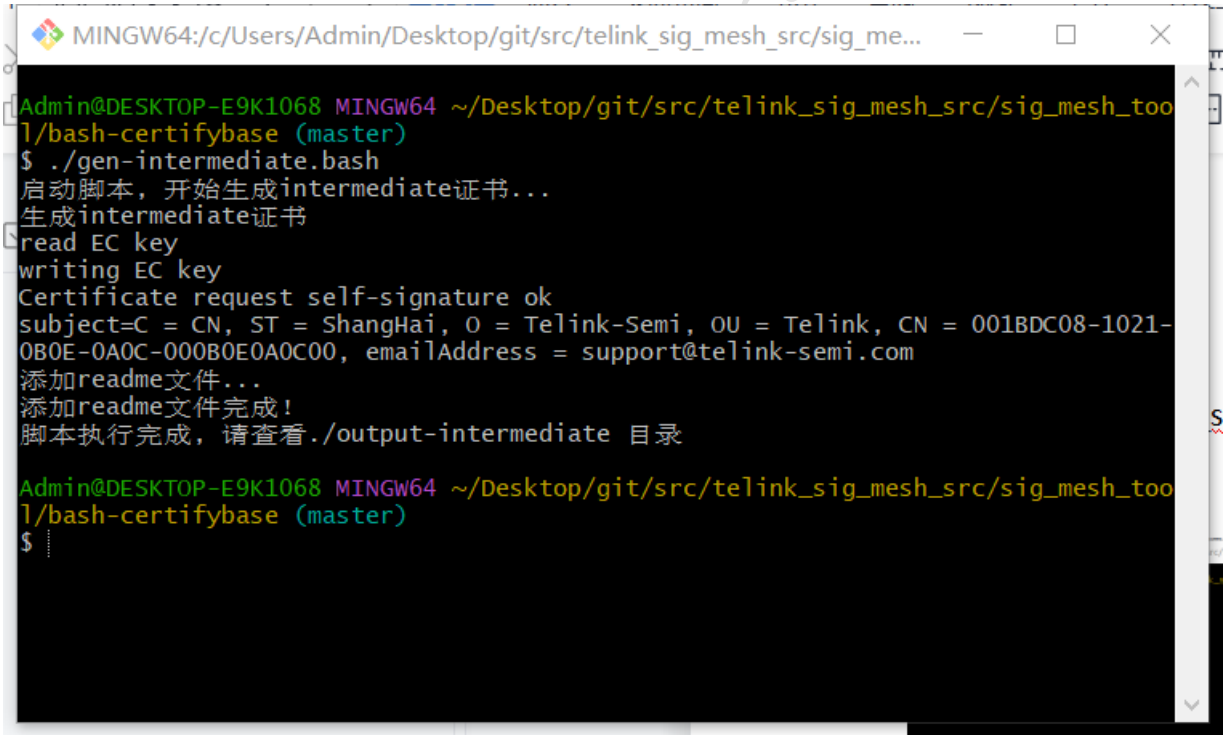


Figure 22.8: Creating an intermediate certificate

22.3.1.4 Configure Device Certificate Parameters

Change the following parameters in gen-device.config:

CN(common name): This parameter is the device UUID, which should be changed first each time a certificate is generated, because the device UUID of each node cannot be the same.

BPID：This parameter is PID (Product ID) and must be equal to MESH_PID_SEL in the firmware sdk code.

BCID：This parameter is CID (Company ID) and must be equal to MESH_VID inside the firmware sdk code.

Note that the CID and PID in the gen-device.config file are in big-endian byte order.

gen-device.config

```
[req]
prompt = no
distinguished_name = req_distinguished_name

[req_distinguished_name]
#Country
C=CN
#State
ST=ShangHai
#Organization
O=Telink-Semi
#Organization Unit
OU=Telink
#common name, need to be replaced to device uuid, CID, PID of unprovision node.
CN=001BDC08-1021-0B0E-0A0C-000B0E0A0C00 BCID:0211 BPID:0001
#emailAddress
emailAddress=support@telink-semi.com

[v3_req]
authorityKeyIdentifier = keyid
subjectKeyIdentifier = hash
basicConstraints = CA:FALSE
keyUsage = Certificate Sign, CRL Sign
#TODO: static oob
#2.25.234763379998062148653007332685657680359 = DER:31:7a:6f:16:58:44:72:74:15:10:33:62:5a:fb:c4:f1
certificatePolicies = critical,@pol

[pol]
policyIdentifier = 2.16.840.1.101.3.2.1.48.1
```

Figure 22.9: Change the device UUID CID PID in gen-device.config

22.3.1.5 Run Gen-device.bash to Generate the Device Certificate

The certificate is signed by an intermediate certificate and the result is output in the output-device directory.

Run command：./gen-device.bash

Figure 22.10: Generate device certificate

22.3.1.6 Burn the Certificate into the Device's Flash

Burn the 4Kbin file generated in the output-device file to the flash address of the device:

Location of FLASH_ADR_CERTIFY_ADR (default is 0x78000)

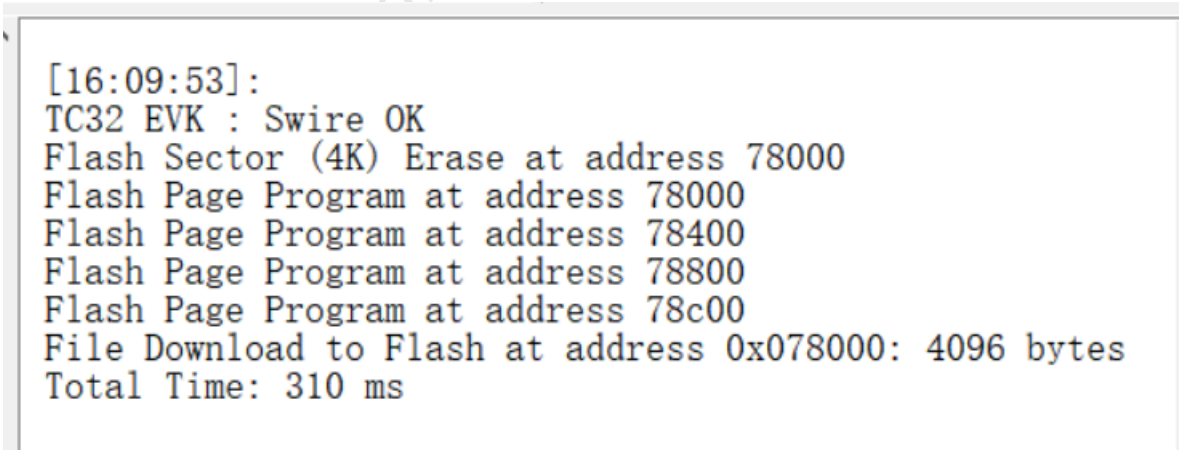


Figure 22.11: Burning certificates to flash

Once the burning is complete, reboot to perform certify base provisioning.

22.3.1.7 Codes Described Below

(1) <Introduction of code processing flow, non-operational steps> The programme will call the crc16 function to check the location of flash address 78000, the length of the check is f00, to confirm whether the read certificate is complete and whether it has been abnormally modified.

```
static u32 val;
val = crc16((u8 *)0x78000, 3840);
```

Figure 22.12: CRC comparison

(2) <Introduction of code processing flow, non-operational steps> The result is compared with the crc value at flash address 78f00, and the expected comparison result is that both are the same.

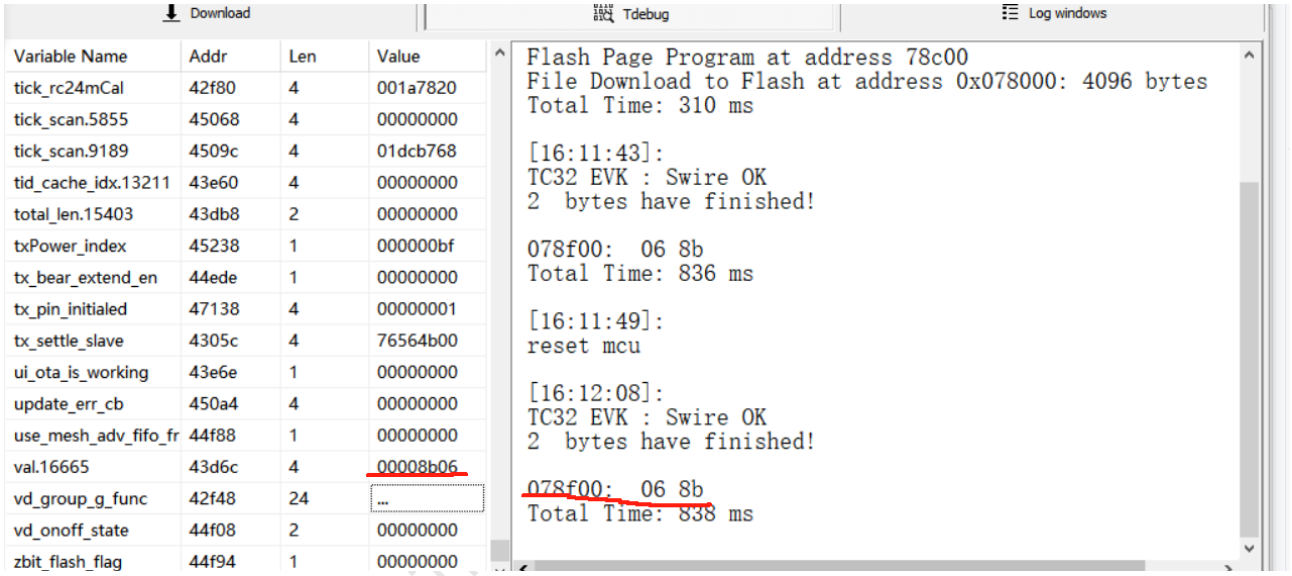


Figure 22.13: CRC comparison

23 Remote Provision Functional Description and Development Instructions

23.1 Remote Provision Function Introductions

In Sig Mesh Spec V1.0 / V1.0.1, Provisioning requires that the Provisioner and the Provisionee are within one hop of each other, because the unprovision beacon packets cannot be relayed directly, so the command interactions in the provisioning process cannot be relayed.

In order to add nodes beyond one hop to the network, Sig Mesh V1.1 adds Remote Provision function.

Remote provision also adds nodes one by one when provisioning, but there is a relay function so you can add more distant nodes into the network.

Important Application Scenario: After adopting Remote Provision, when the host (Provisioner) is not convenient to move, it is also possible to realise that the mesh nodes can be arranged according to the actual scenario in the application first, and then the network can be formed. Especially the application scenario with gateway.

In addition to remote provisioning, remote provisioning can also be used for updating Device Key, Node Address and Composition Data, as described in "3.11.8 Node Provisioning Protocol For details, see"3.11.8 Node Provisioning Protocol Interface procedures" in V1.1 spec.

Note: The above functional overview can also be found in the description on the SIG official website https:// www.bluetooth.com/ mesh-feature-enhancements-summary/ and https:// www.bluetooth.com/ mesh-remote-provisioning/ .

23.1.1 Introduction to Remote_provision Network Interaction Process

(The whole provisioning process is also done node by node).

(1) First network one or more nodes within one hop of the host (Provisioner).

(2) Scan for unprovision beacons sent from nodes further away (within the second hop range) through already networked nodes and report them to the Provisioner.

(3) Provisioner selects a node that reports an unprovision beacon (for example, networked node A reports a scan to unprovisioned node B).

(4) When Provisioner networks node B, it encapsulates the message to be sent to node B into a mesh network message and sends it to node A first, and then node A extracts the network information and sends it to the un-networked node B in the form of a generic provision PDU (either in the form of a PB-ADV, or a PB-GATT).

(5) The message that node B replies to the Provisioner will be sent to node A first. Then node A encapsulates the message into a mesh network message and send to the Provisioner.

(6) Steps 4 and 5 are executed several times until the networking is completed. During this process, for the unprovisioned device B, it can be considered that node A does not exist, and there is no difference with the normal provisioning mode.

(7) Finish provisioning the nodes in the second hop range by repeating steps 2 to 6. Then search and network the nodes in the third hop in the same way..... until the search fails to find any unprovision nodes.

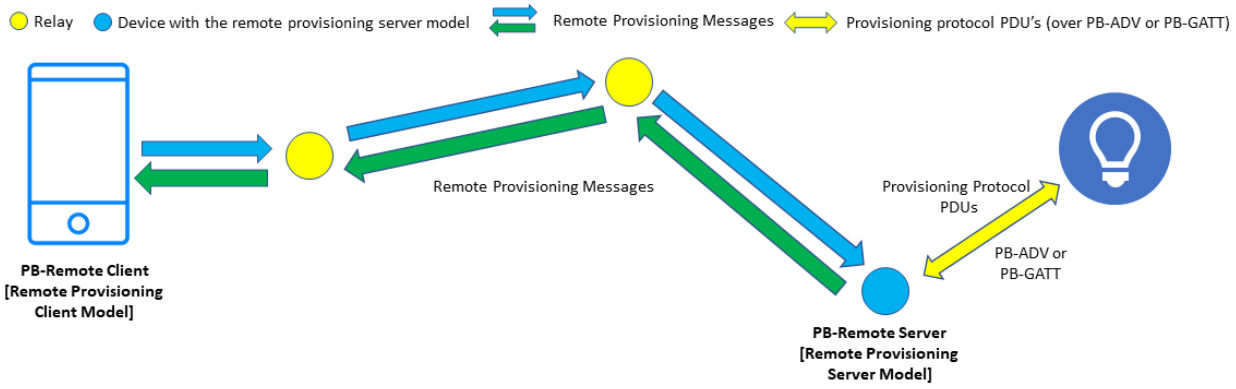


Figure 23.1: The Architecture Of Remote Provisioning

23.1.2 Remote Provision Opcode and Flowchart

Remote provisioning opcode (see "4.3.4 Remote provisioning information" in V1.1 spec for command parameters).

Element	Model Name	State	Message	Rx	Tx
Remote Provisioning Main	Remote Provisioning Server	Remote Provisioning Scan Capabilities	Remote Provisioning Scan Capabilities Get	M	–
			Remote Provisioning Scan Capabilities Status	–	M
		Remote Provisioning Scan Parameters	Remote Provisioning Scan Get	M	–
			Remote Provisioning Scan Start	M	–
			Remote Provisioning Scan Stop	M	–
			Remote Provisioning Scan Status	–	M
			Remote Provisioning Scan Report	–	M
			Remote Provisioning Extended Scan Start	M	–
			Remote Provisioning Extended Scan Report	–	M
		Remote Provisioning Link Parameters	Remote Provisioning Link Get	M	–
			Remote Provisioning Link Open	M	–
			Remote Provisioning Link Close	M	–
			Remote Provisioning Link Status	–	M
			Remote Provisioning Link Report	–	M
			Remote Provisioning PDU Send	M	–
			Remote Provisioning PDU Outbound Report	–	M
			Remote Provisioning PDU Report	–	M

Table 4.334: Remote Provisioning Server model messages

Figure 23.2: MD_REMOTE_PROV_opcode

Introduction to remote_provision network interaction process The remote scan flowchart in step 2: (where capa get is Remote Provisioning Scan Capabilities Get, you can get the maximum number of unprovisioned nodes that can be scanned by the current mesh node and whether active-scan is supported).

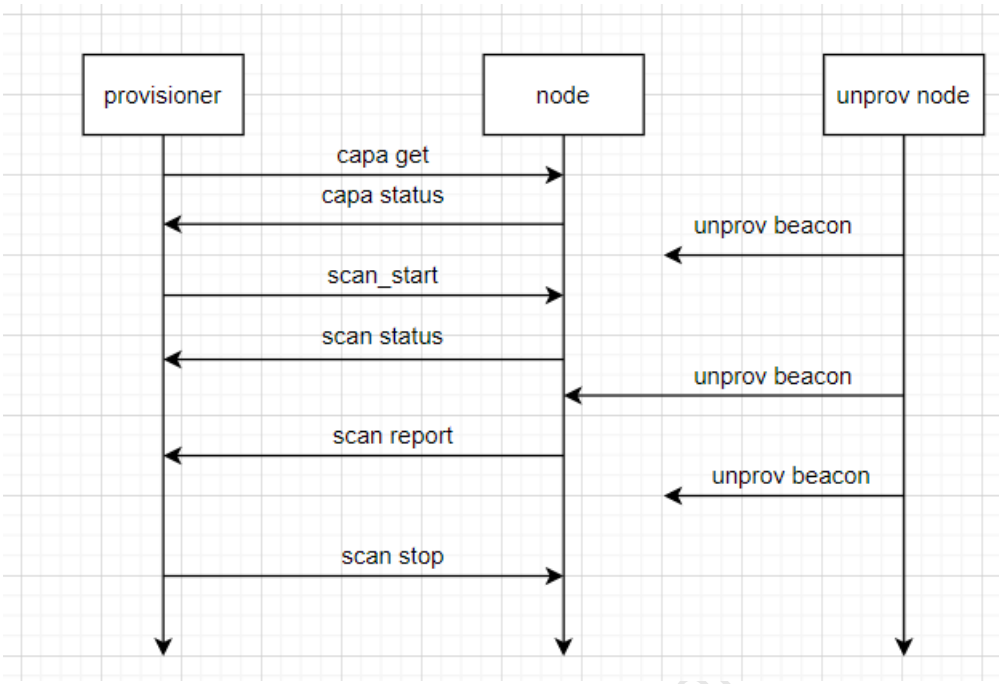


Figure 23.3: MD_REMOTE_PROV_scan_flow

Introduction to remote_provision network interaction process Flowchart of remote provision for step 3/4/5:

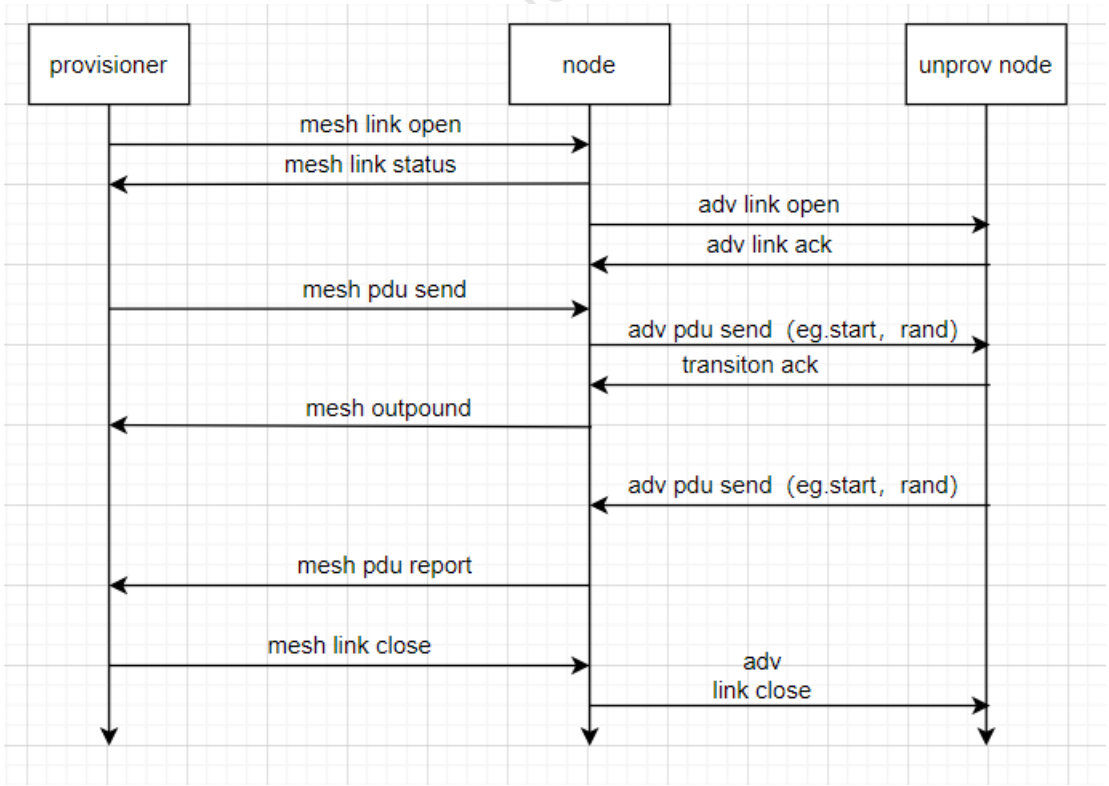


Figure 23.4: MD_REMOTE_PROV_provision_flow

23.2 Testing Remote Provisioning with the App

23.2.1 Test Conditions

8258 dongle greater than or equal to 2 (burn 8258_mesh.bin), Android or iOS SIG Mesh App.

23.2.2 Firmware SDK Code Configuration

In the default configuration, the RPR feature is turned off. To enable it, you need to turn on the MD_REMOTE_PROV macro switch on the node side in the mesh_config.h file. This is shown in the following figure:

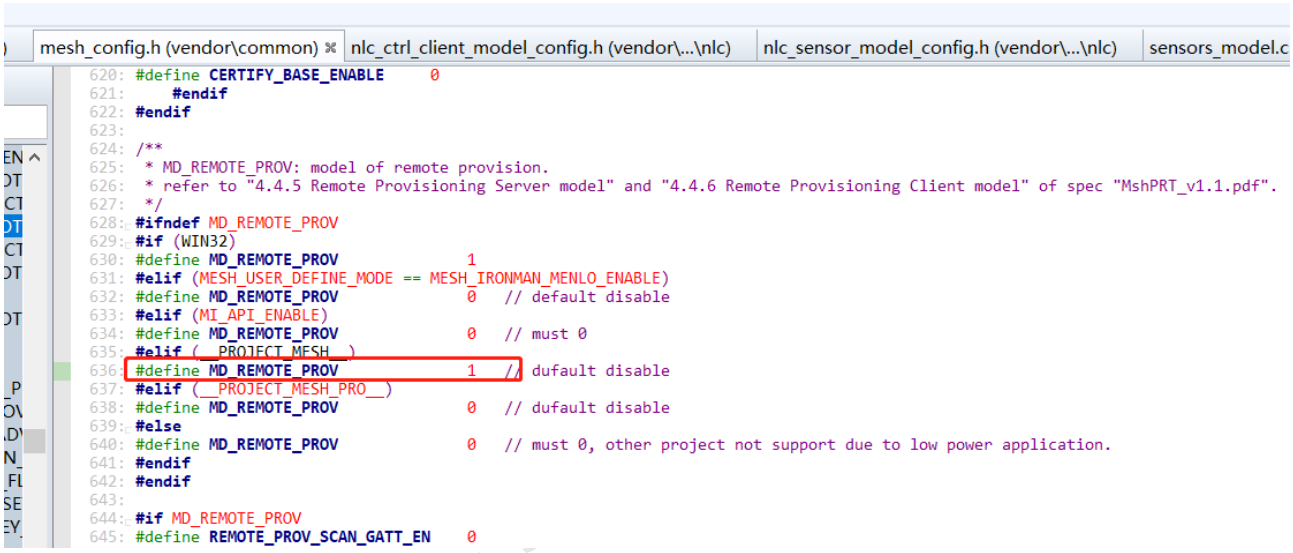


Figure 23.5: 打开 MD_REMOTE_PROV_App

23.2.3 App Settings

App Home Click - Settings - Preset Mode, select Remote Provision

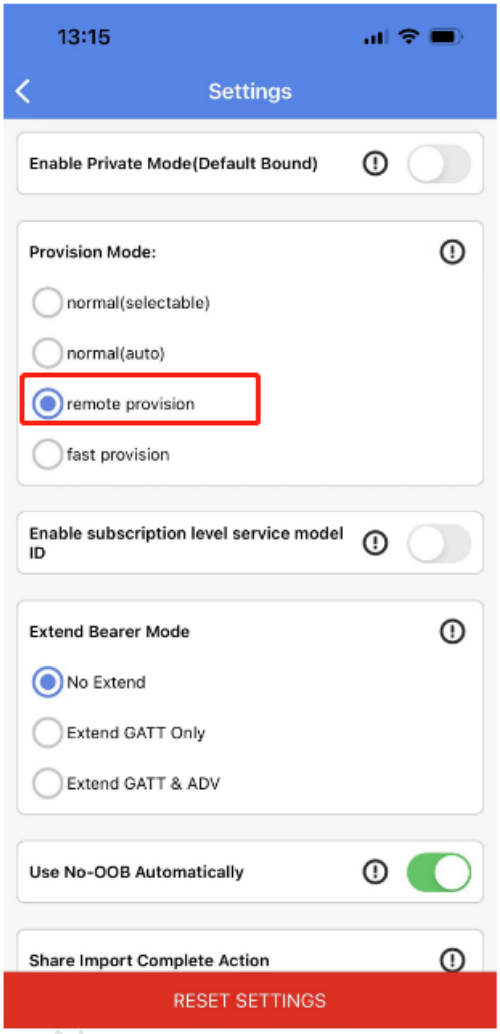


Figure 23.6: MD_REMOTE_PROV_App_setting

23.2.4 Test Steps

(1) Tap the "+" sign on the home page of the App to enter the Remote Provisioning page. Then it will start the automatic provisioning.

Before starting automatic provisioning, the app will judge whether the current app is in GATT connected state with the networked node that supports Remote Provision, if not, it will carry out normal PB-GATT networking, if yes, it will carry out remote provisioning to other un-networked nodes through this networked node. As shown in the figure below, the first one (top left) is networked by normal PB-GATT, and the others (top right and bottom left) are networked by remote provision.

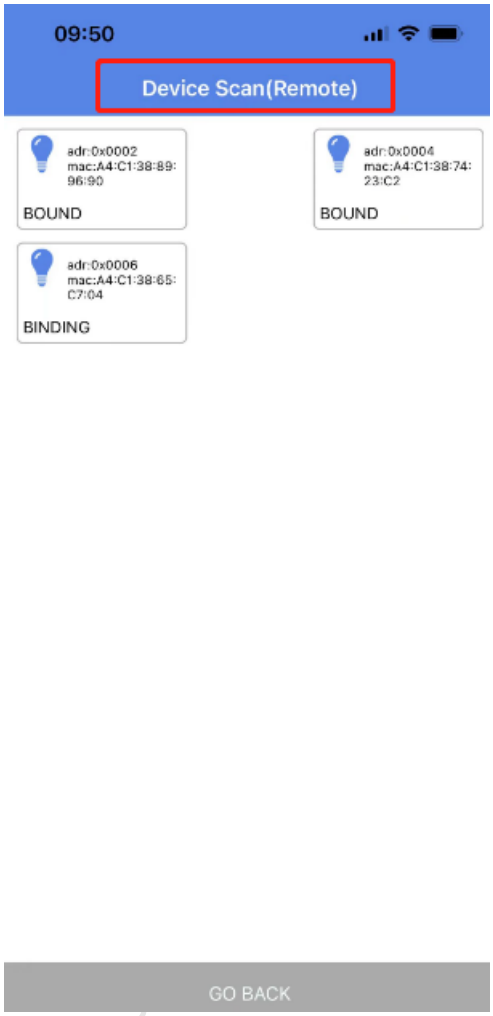


Figure 23.7: MD_REMOTE_PROV_App_provision

(2) When the timeout has not been scanned for un-networked nodes, it indicates that all nodes have been networked, and then returns to the home page to display the following:

Figure 23.8: MD_REMOTE_PROV_App_provision_success

23.3 Gateway Remote Provision Host Computer Development Guide

23.3.1 Code and Tool Parameter Configuration for Gateway's Remote Provision

Test conditions: 1 x 8258 dongle (burning 8258_mesh_gw.bin), 2 x 8258 dongles (burning 8258_mesh.bin)

(1) MD_REMOTE_PROV is turned on.

```
624: /**
625:  * MD_REMOTE_PROV: model of remote provision.
626:  * refer to "4.4.5 Remote Provisioning Server model" and "4.4.6 Remote Provisioning Client model" of spec "MshPR
627:  */
628: #ifndef MD_REMOTE_PROV
629: #if (WIN32)
630: #define MD_REMOTE_PROV              1
631: #elif (MESH_USER_DEFINE_MODE == MESH_IRONMAN_MENLO_ENABLE)
632: #define MD_REMOTE_PROV              0   // default disable
633: #elif (MI_API_ENABLE)
634: #define MD_REMOTE_PROV              0   // must 0
635: #elif (__PROJECT_MESH__)
636: #define MD_REMOTE_PROV              1   // dufault disable
637: #elif (__PROJECT_MESH_PRO__)
638: #define MD_REMOTE_PROV              1   // dufault disable
639: #else
640: #define MD_REMOTE_PROV              0   // must 0, other project not support due to low power application.
641: #endif
642: #endif
```

Figure 23.9: Open MD_REMOTE_PROV

(2) If you want to improve the efficiency of the distribution network, you can set the macro EXTENDED_ADV_ENABLE to 1, which means that you can use the extended broadcast packet mode to send remote provision messages. Note that this mode is private.

- Enable the Macro in code.

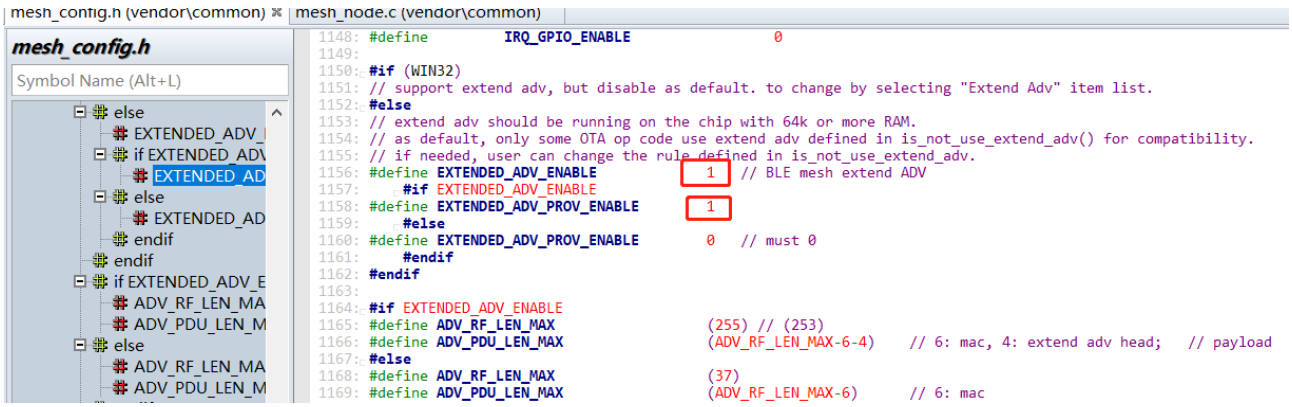


Figure 23.10: Open EXTENDED_ADV_ENABLE

- Check whether the circled code below has been added, and if not, add it.

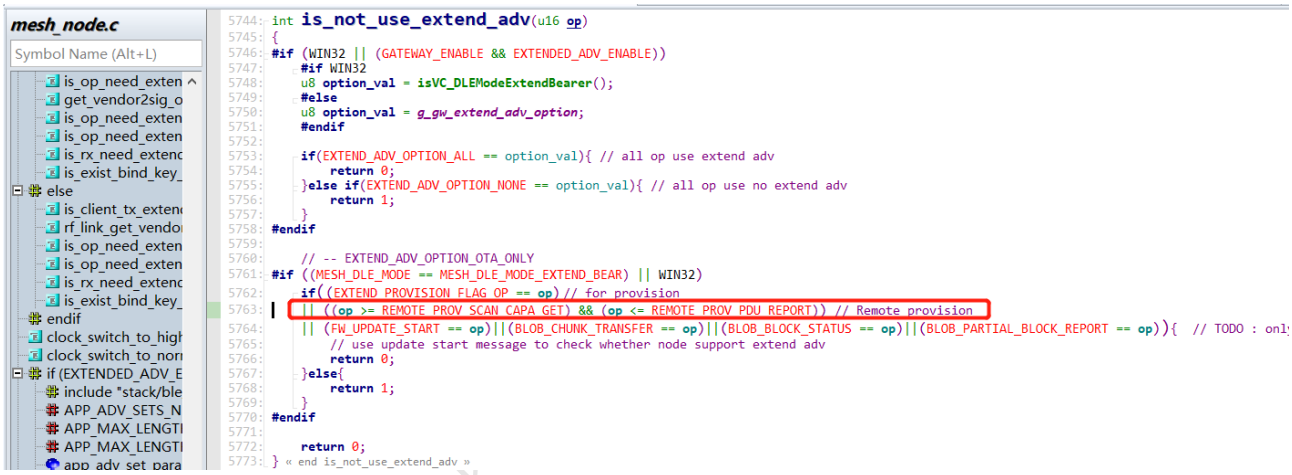


Figure 23.11: EXTENDED_ADV_ENABLE

- The host computer sets the gateway to extended broadcast packet mode.

The handling of the 3 modes of the "Extend Adv" control is detailed in the is_not_use_extend_adv() function.

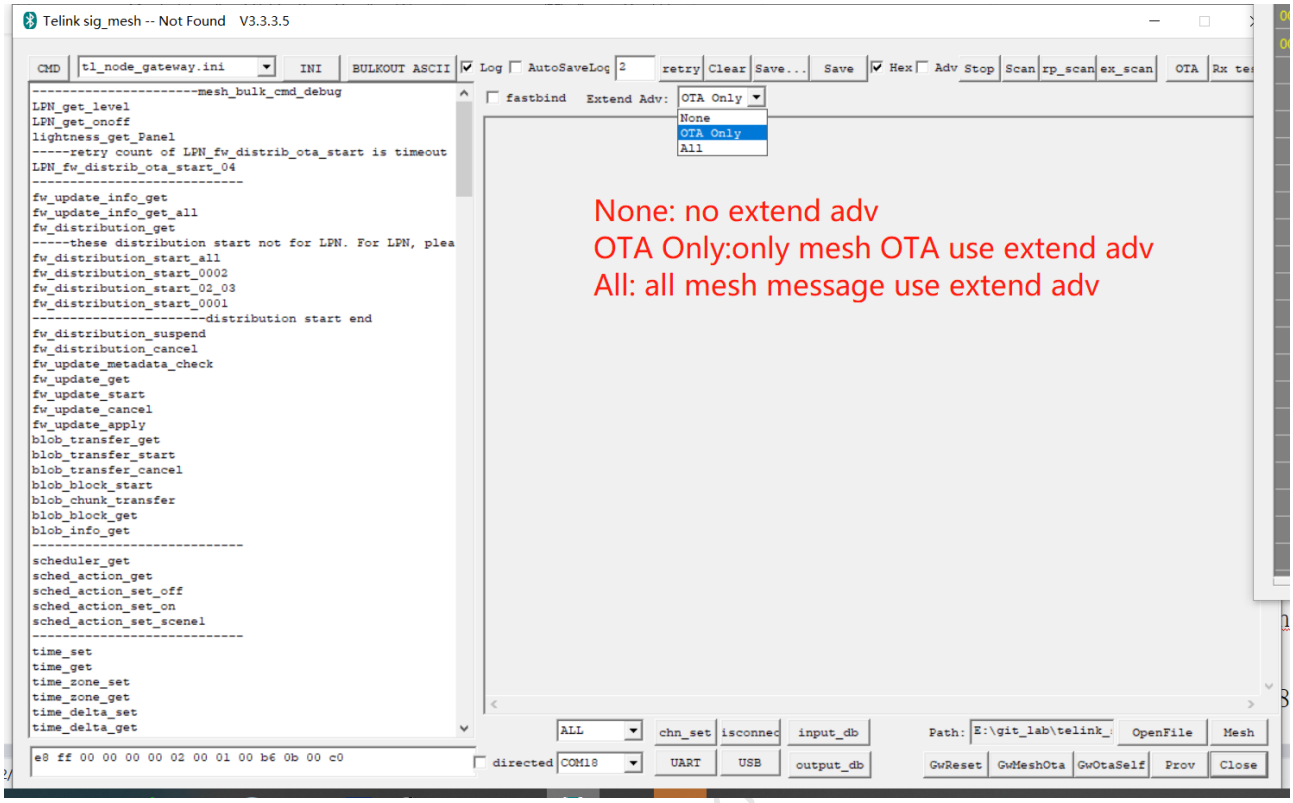


Figure 23.12: Gateway settings extended broadcast packets

Note:

> If the gateway dongle is accidentally powered off, the gateway will be set to "GATT Only" mode by default, and you need to click "Extend Adv" again to set the gateway to Extended Broadcast Packet Mode (All), or turn off and then turn on the host computer again, which automatically refreshes the current "Extend Adv" mode of the host computer to the "Extend Adv" mode of the gateway dongle.

23.3.2 Phase 1 Network One or More Nodes in Normal pb_adv Style

When the network is empty and there is no networked device, one or more nodes within one hop of the gateway need to be networked first by ordinary PB-ADV.

Provisioning steps: connect to the pc via gw dongle, open tools tool sig_mesh_tool.exe, select the ini file corresponding to gw as shown in the figure below, and provision one/multiple mesh nodes with pb_adv, refer to the sig mesh handle book gateway project for the specific process: "Provisioner operation and APIs".

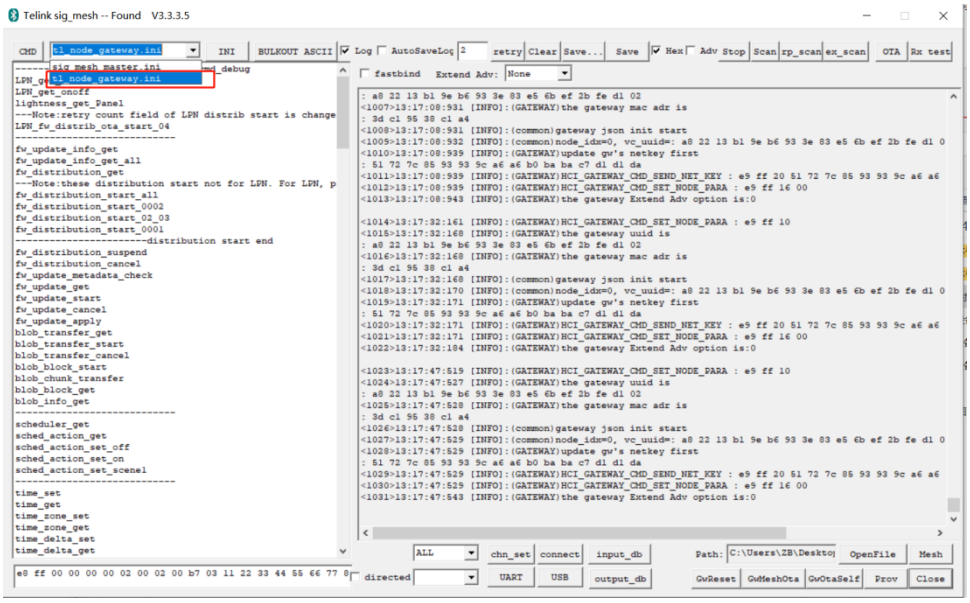


Figure 23.13: Setting the gateway mode

After successful provisioning, click "MESH" button to open the mesh control interface, and automatically send LIGHTNESS_GET all command to get all the current nodes (you can also send ONOFF_GET), and display in the UI interface. If you are already in the mesh control interface, and the UI interface does not show any node that supports remote provision, you need to click the "Nodes" button, and this command also sends the LIGHTNESS_GET all command.

The LIGHTNESS_GET all command has the following format:

HCI_CMD_GATEWAY_CMD + netkey index(2 bytes) + appkey index(2 bytes) + retry cnt + rsp_max + gateway addr + op + par

i.e.: e8 ff + netkey index + appkey index + retry cnt + rsp_max + gateway addr + 82 02 00 00

Note: The ini format is hexadecimal throughout this document unless otherwise noted.

23.3.3 Stage 2 Remote Provision Add Light

(1) Click the rp_scan button in the figure and set the limit and timeout parameters (see V1.1 spec "4.3.4.4 Remote Provisioning Scan Start" for details).

limit: Maximum number of scanned items to be reported. Value 0 indicates no limit.

timeout: Time limit for a scan (in seconds)

Then click Confirm and the tool will issue the scan start command on the list of nodes obtained in the previous step in the format:

HCI_CMD_GATEWAY_CMD + netkey index(2 bytes) + appkey index(2 bytes) + retry cnt + rsp_max + scan server addr + op + par

i.e. e8 ff + 00 00 + 00 00 + 02 + 01 + scan server addr + 80 52 + limit + timeout

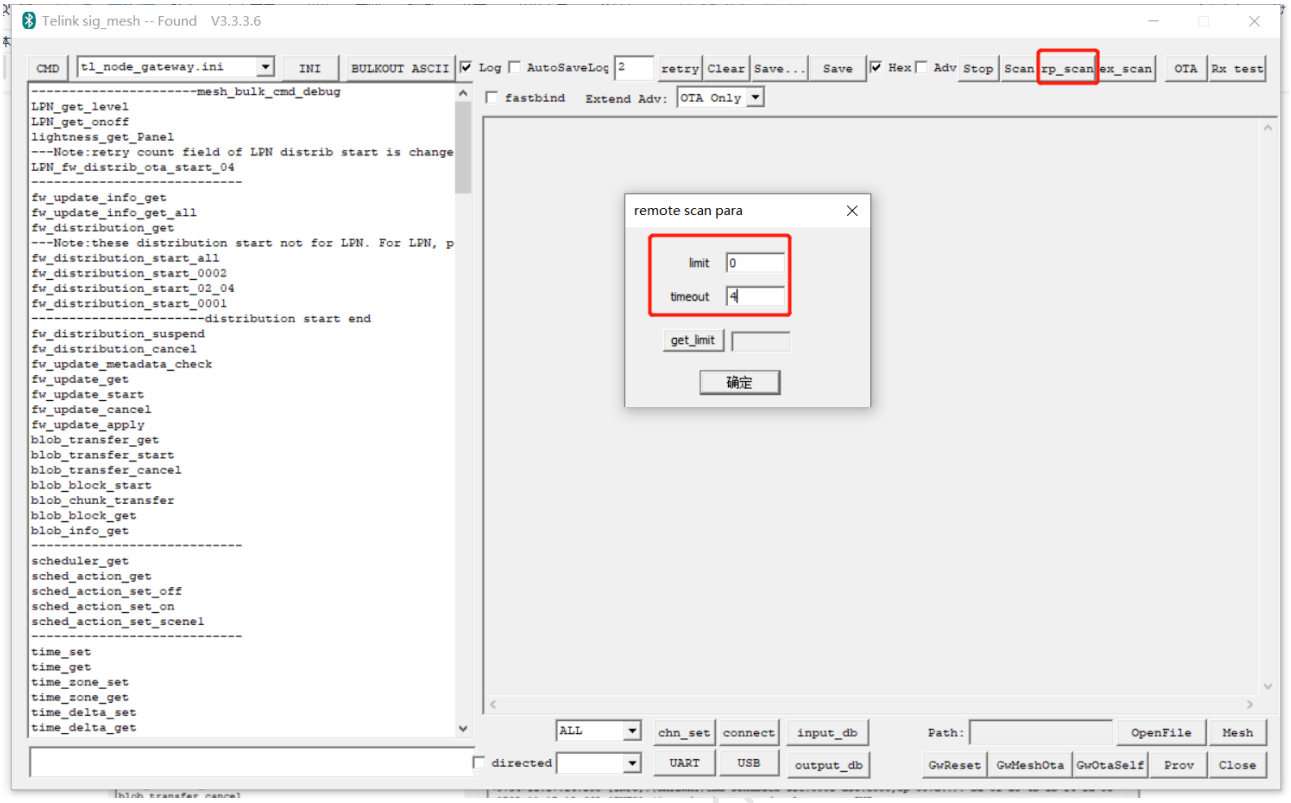


Figure 23.14: Remote_provision add lights

The node receives a scan start and replies with a scan status in the format of:

len(2 bytes) + TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + scan server addr + gateway addr + op + par.

i.e.: len + 91 + 81 + scan server addr + dst addr + 80 54 + par.

Note: If the rp_scan button is clicked with the error message shown below:

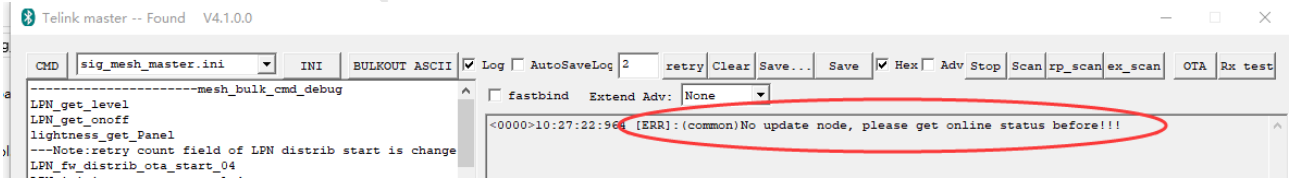


Figure 23.15: No nodes that support the remote_provision feature are selected

The error message means that there is not any node was found to be provisoned and supports remote provision function. so there is no way to do remote provision. it may happen when the action mentioned in the previous step, "Click the 'MESH' button to open the mesh control interface", has not been done. if that happen, click the "MESH" button to fix this error.

(2) To specify one or more nodes to SCAN and report unprovision beacon sending by unprovisoned node via the REMOTE_PROV_SCAN_REPORT message in the format.

len(2 bytes) + TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + src addr + gateway addr + op + rssi + uuid + oob info(2 bytes) + uri hash(2 bytes, optional).

i.e.: len + 91 + 81 + src addr + gateway addr + 80 55 + rssi + uuid + oob info.

Note: In order to facilitate the upper computer to display the mac, when the device opens the remote provision function, the last 6 bytes of the device uuid in the sdk are set to the mac address as default. see the processing inside the uuid_create_by_mac() for details. As shown in the figure below:

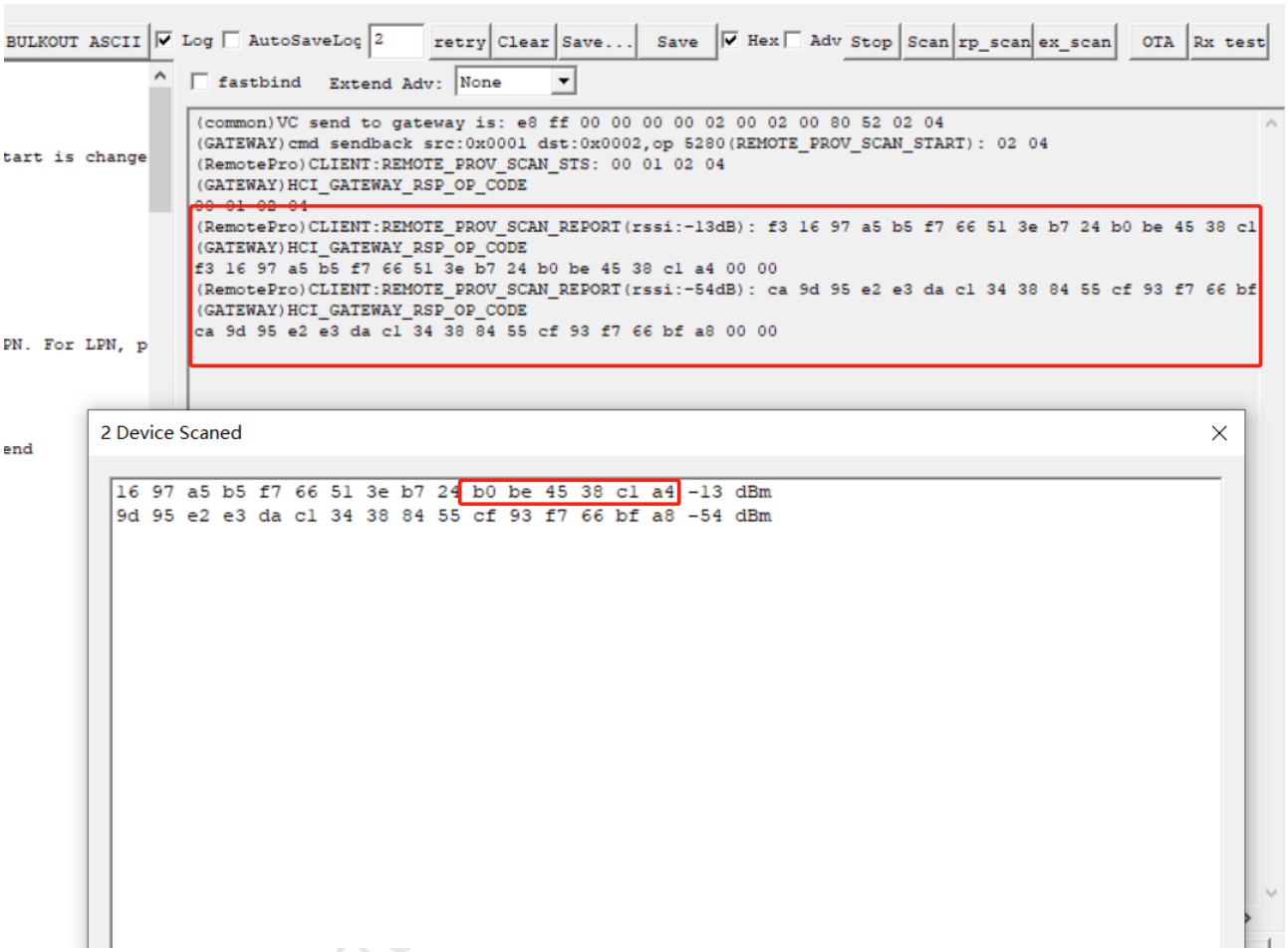


Figure 23.16: Remote provision nodes

(3) Double-click the device to be provisioned in the scanned device list, the corresponding command is:

HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_RP_LINK_OPEN + scan server addr + uuid

i.e.: E9 FF + 1A + scan server addr + uuid.

After the gateway receives HCI_GATEWAY_CMD_RP_LINK_OPEN, it sends the Remote Provisioning Link Open command to the scan server node, and the scan server node replies Remote Provisioning Link Status to the gateway after it receives it, and the format of Remote Provisioning Link Status is as follows:

len(2 bytes) + TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + scan server addr + gateway addr + op + par.

i.e.: len + 91 + 81 + scan server addr + gateway addr + 80 5B + par.

As shown below, double-click on the device that needs provisioning:

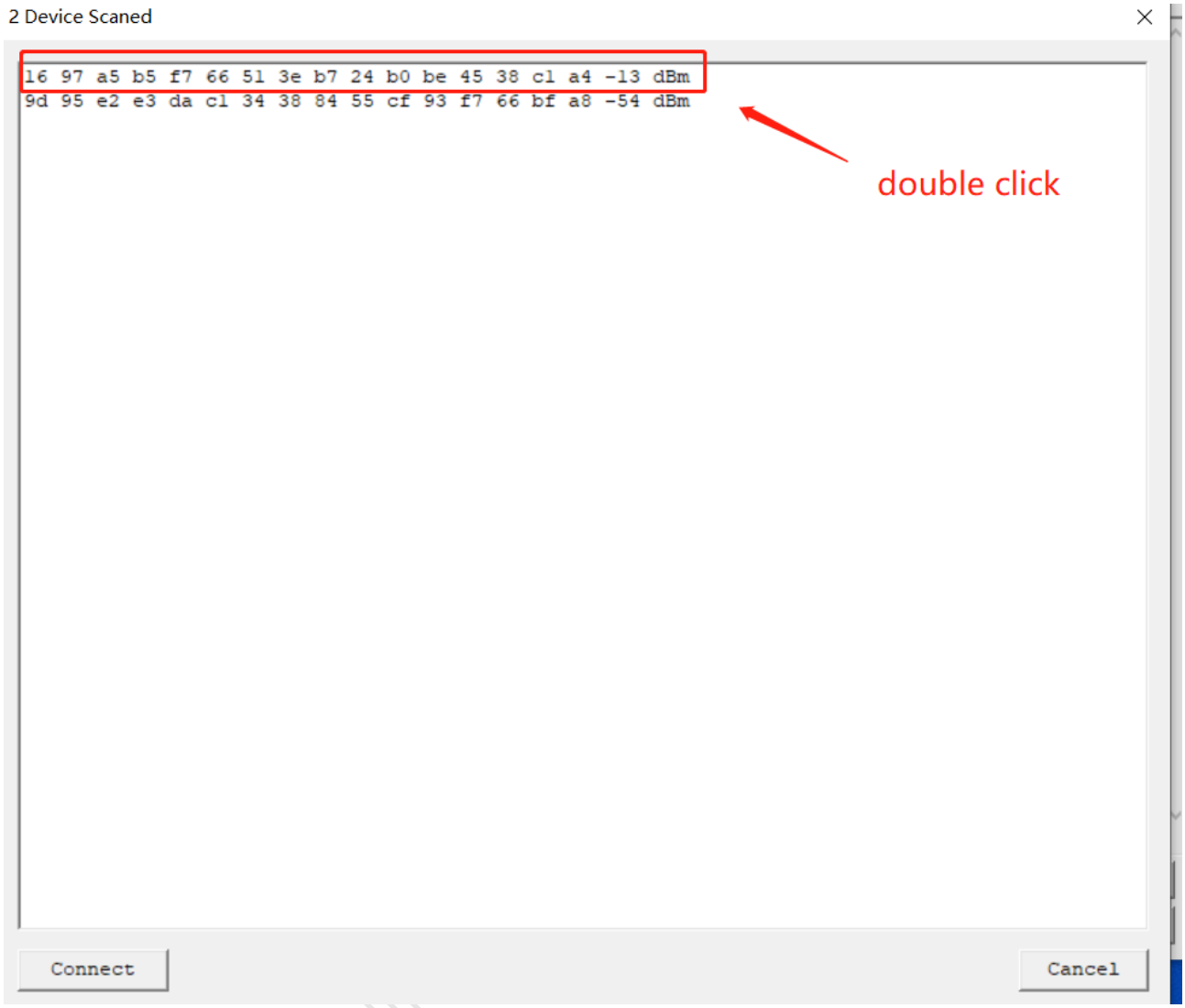


Figure 23.17: Double click nodes

(4) Click the "Prov" button to enter the Provision interface, the corresponding command for the "Prov" button is:

HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_GET_PRO_SELF_STS.

i.e.: e9 ff + 0c.

Click "Prov" button to enter the Provision screen as shown below:

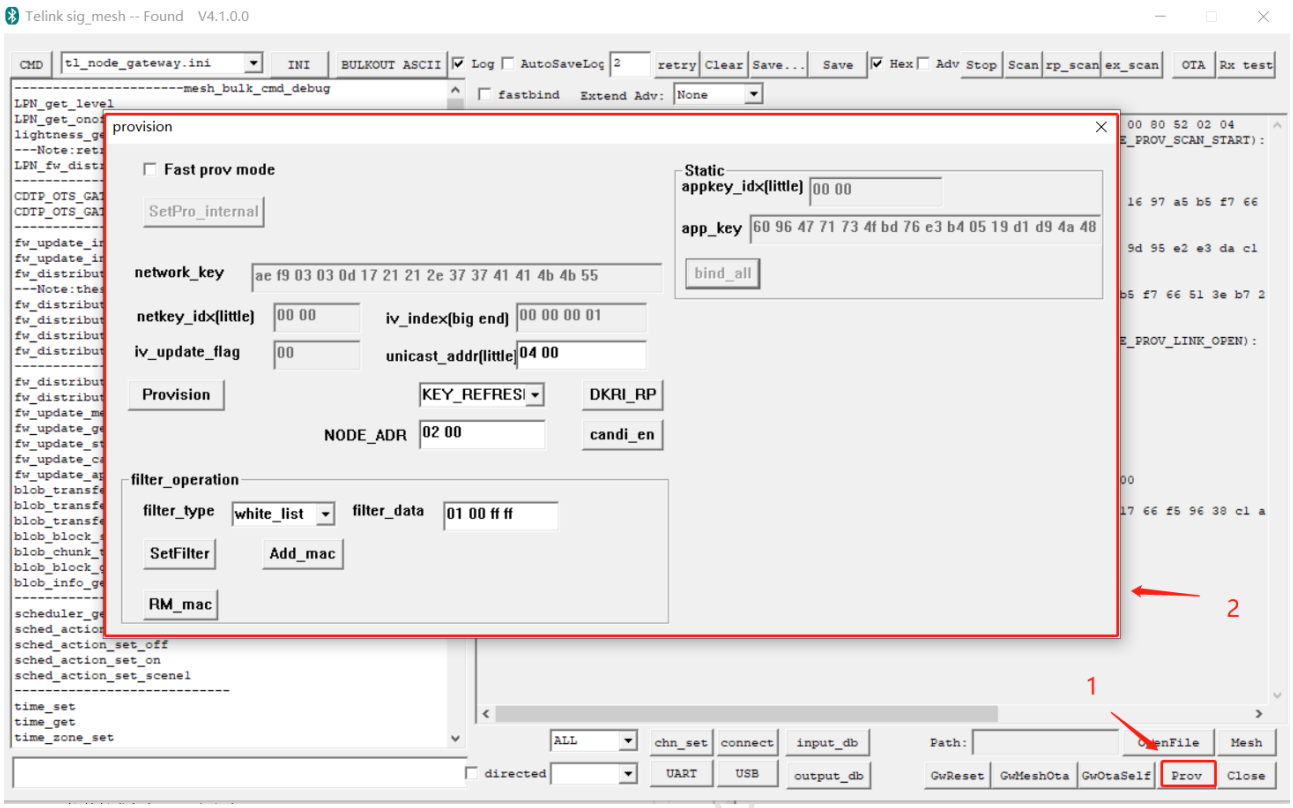


Figure 23.18: Click prov

The gateway receives this command and returns whether there is already configuration information and the number of elements address occupied by gateway itself. The gateway configuration information command format is.

len(2 bytes) + TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_PRO_STS_RSP + provision_flag + pro_net_info.

i.e.: len + 91 8b + provision_flag+ pro_net_info.

A provision_flag of 1 indicates that the gateway is provisioned. 0 indicates that network information needs to be set, this case please refer to "Add Light via Provisioner".

The number of elements command format for the gateway itself:

len(2 bytes) + TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_SEND_ELE_CNT.

i.e.: len + 91 + 8C + gateway element count.

(5) Clicking the "provision" button to trigger the addition of a light, corresponding to the command:

HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_RP_START + provision data.

i.e.: E9 FF + 1B + provision data.

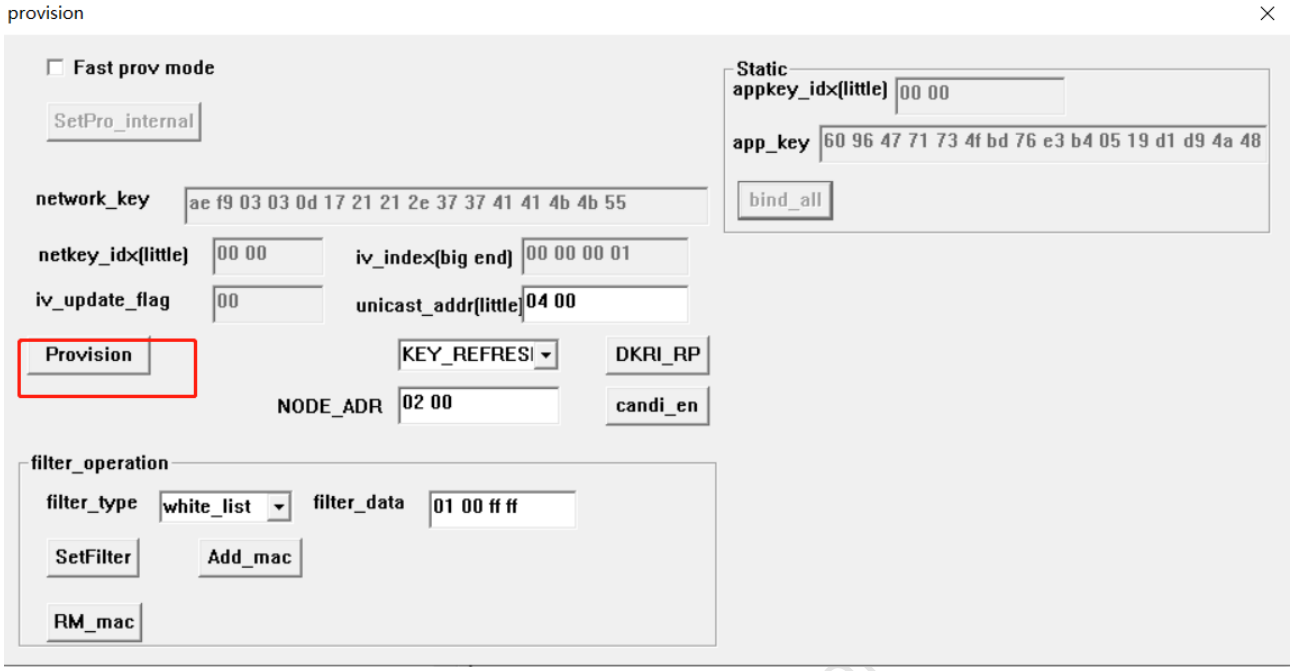


Figure 23.19: Trigger adding light

(6) The device provisioning status is reported when provisioning is complete. The format is:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_PROVISION_EVT+ gateway_prov_event_t.

i.e.: 91 + 89 + gateway_prov_event_t.

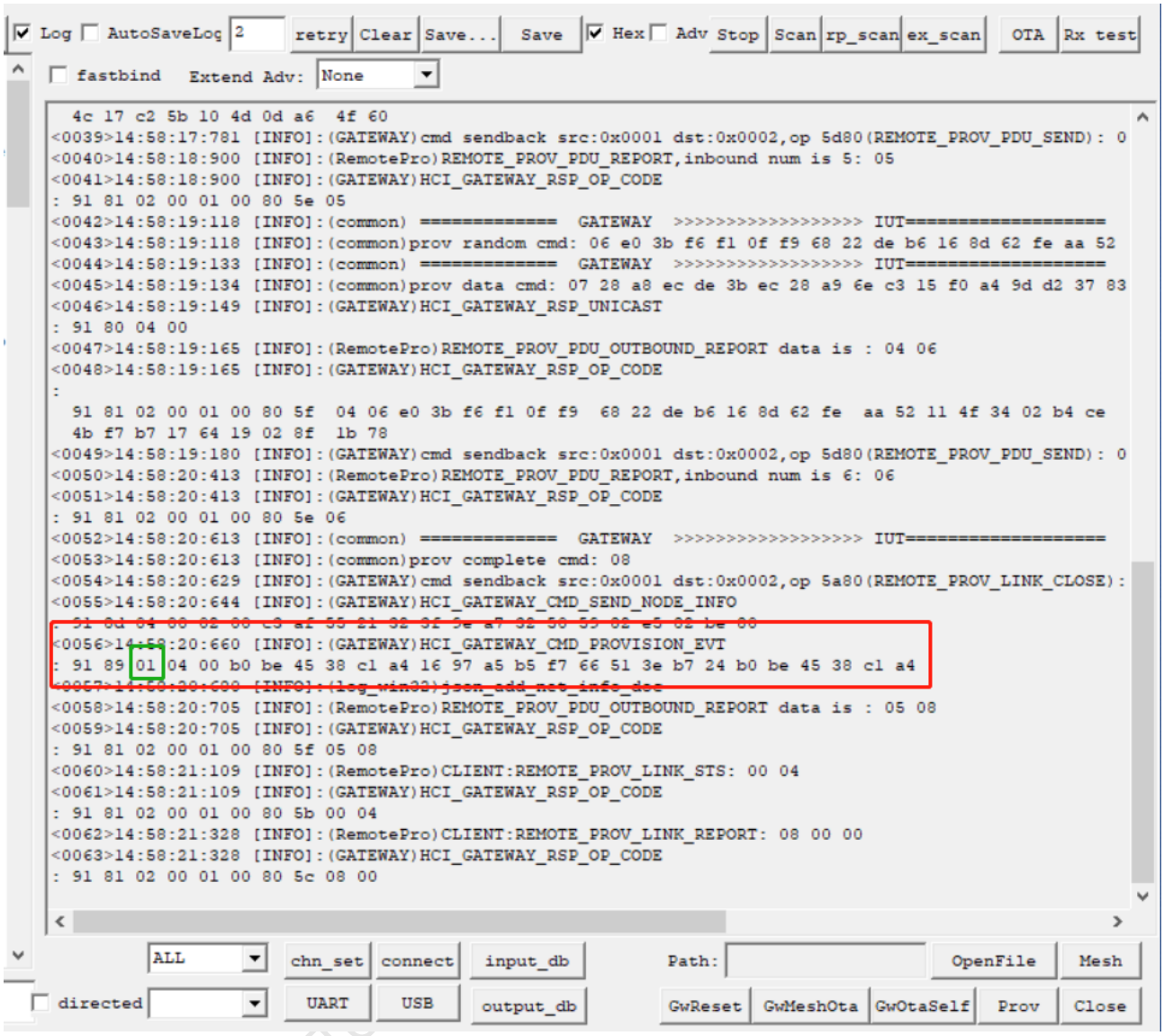


Figure 23.20: Provisioning status

```
typedef struct{
    u8 eve;//1 indicates success
    u16 adr;
    u8 mac[6];
    u8 uuid[16];
}gateway_prov_event_t;
```

(7) Bind app_key

After Provisioning is complete, you also need to bind the app_key for the model. click "bind_all" button to bind the app_key for the model. the process is the same as for non-remote provisioning modes.

a. The corresponding command for bind_all is: HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_START_KEYBIND + fast_bind + app_key index(2 byte)+app_key(16 bytes).

i.e.: e9 ff + 0b + fast_bind + app_key index(2 byte)+app_key(16 bytes).

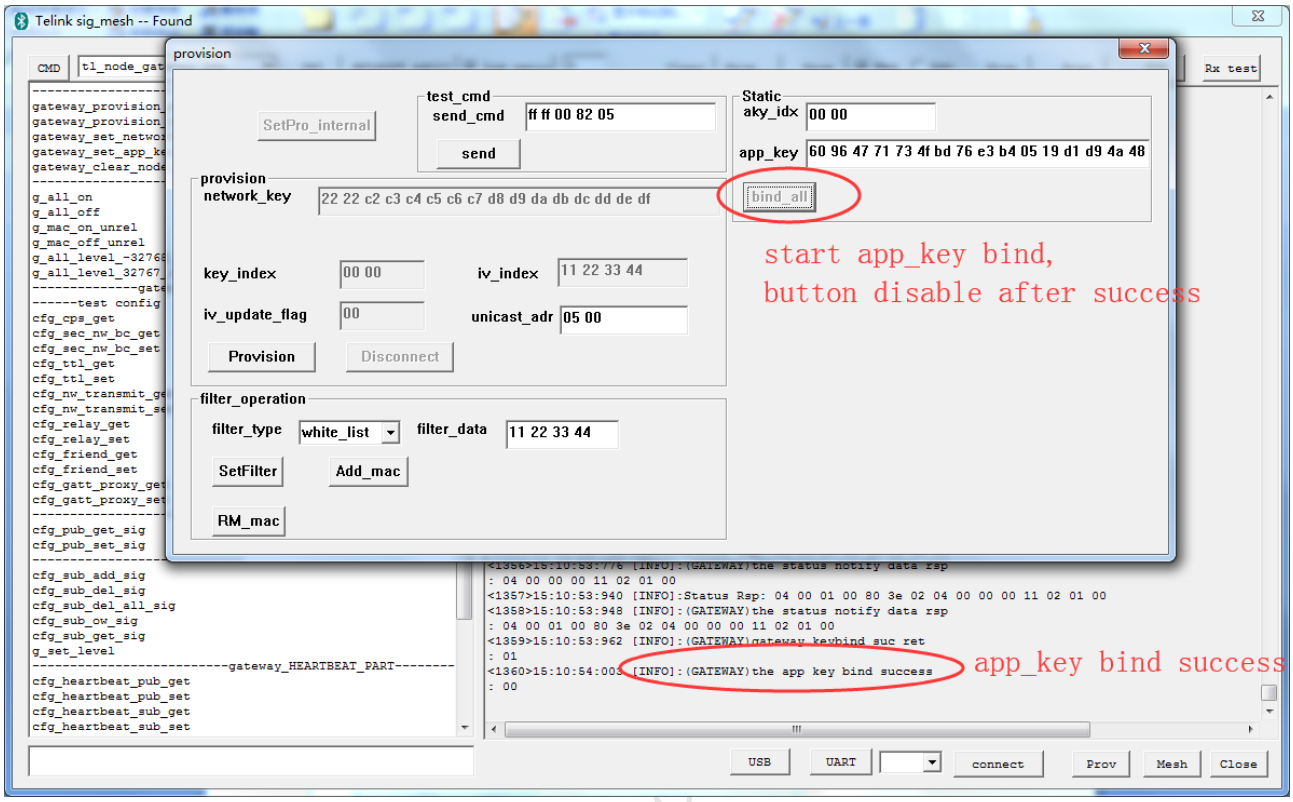


Figure 23.21: Bind app_key

When fast_bind is 1: the gateway will only issue appkey add, the provisioned device needs to turn on the default binding function (PROVISION_FLOW_SIMPLE_EN is set to 1).

When fast_bind is 0: the gateway binds all the model ids by default, in order to save time, users can choose the model ids that need to be bound. open the macro MD_BIND_WHITE_LIST_EN on the gateway side, and the model ids that need to be bound can be seen in the master_filter_list in the Mesh_common.c file. [], users can modify it as needed.

b. App_key bind process gateway will call u8 gateway_model_cmd_rsp(u8 *para,u8 len) to return the status information of the bound model, the format is:TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + Parameters.

i.e.: 91 + 81 + appkey bind status

c. App_key bind completes and returns HCI_GATEWAY_CMD_KEY_BIND_EVT indicating success or time_out. in the format of:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_KEY_BIND_EVT +result.

i.e.: 91 + 8a + result.(1:success 2:time_out).

(8) Repeat steps (1)~(7) to perform remote provision for other nodes one by one until all nodes are networked.

24 Mesh OTA and Guide for Host Computer Development

24.1 Mesh OTA Introduction

24.1.1 Mesh OTA Features and Modes

The mesh OTA, termed mesh DFU (device firmware update) in the mesh spec, is a function that performs firmware upgrades for nodes. This function specifies how OTA is implemented and enables simultaneous firmware upgrades for nodes beyond the RF multi-hop distance.

24.1.2 Introduction to Mesh OTA Modes and Reference Rates

The following modes are supported:

(1) Support to upgrade multiple nodes at the same time by mesh ADV relay. Currently, the upgrade time for Demo SDK 160k firmware is around 60 minutes, and if Telink's Extended Broadcast Packet mode is enabled, the upgrade time is around 4 minutes.

(2) mesh OTA for LPN: the current Demo SDK 130k firmware upgrade time is around 70 minutes, if Telink's extended broadcast packet mode is enabled, the upgrade time is around 4 minutes.

(3) GATT OTA mode for single node (including LPN node): APP will disconnect the current connection, go to connect the upgraded node, and then execute OTA of SIG MESH, the sending and receiving process is the same as the process of upgrading multiple nodes, only the sending and receiving packet interactions are faster, the OTA time is basically the same as that of the OTA defined in the Telink gerneric BLE SDK, and the upgrading time is about 1 minute.

An overview of the functions can also be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/

24.1.3 Mesh OTA Firmware Distribution Method

(1) The host computer or app acts as both Initiator and distributor: The host computer is involved in the process of distributing firmware data to multiple nodes that need to be upgraded. The host has to stay connected to the nodes at all times and cannot be disconnected, a process that can take several tens of minutes.

(2) "GATT master dongle + host computer" or app as Initiator, GATT directly connected node as distributor: in this mode, the host computer only needs to download the new firmware to the directly connected node through GATT in the front stage, and then the directly connected node acts as distributor to manage and execute the later work: distributing the new firmware to other nodes to be upgraded. At this time, the App can disconnect with the node if needed. Gateway does not support this mode for the time being, but directly adopts the mode of the host computer plus the gateway dongle as the Initiator and distributor, because the gateway doesn't need to keep the GATT connection with the nodes.

24.1.4 Three Role Profiles of Mesh OTA

(1) Initiator: A node whose DISTRIBUTOR_UPDATE_CLIENT_EN is set to 1 in the SDK functions as an OTA initiator, e.g., an PC tool sig_mesh_tool.exe, an app, etc. The Initiator fetches new firmware from the local or cloud via http and generates a receiver list, which specifies the nodes to which the OTA update is to be initiated. Then it sends the new firmware and receiver list to the Distributor, the corresponding process is in mesh_ota_initiator_proc ().

(2) Distributor: Nodes with DISTRIBUTOR_UPDATE_SERVER_EN set to 1, such as gateway dongle and Mesh Light nodes with distributor turned on (not supported by default, need to enable DISTRIBUTOR_UPDATE_SERVER_EN). Function: Receive the firmware from Initiator, or download the new firmware through the http URL sent by Initiator, and store it temporarily. Then distribute the new firmware to the nodes belonging to the receiver list. Distributor corresponds to the sending process in mesh_ota_master_proc () and receiving process in mesh_ota_master_rx ().

(3) Updating Node: The node that receives the OTA new firmware and updates the old firmware to the new firmware. That is, the node belongs to the receiver list.

24.1.5 Mesh OTA Silent Upgrade Mode

After distributing the firmware completely, the upgraded node will not apply the new firmware until receive firmware update apply command. So app or host can choose a suitable time to send firmware update apply command after transmitting firmware to the upgraded node to achieve silent upgrade mode.

24.1.6 Mods for Mesh OTA

(1) BLOB Transfer(Binary Large Object Transfer) server: this model is used to receive large block data transfer, including but not limited to receiving firmware data and contains commands such as BLOB_INFO_GET, BLOB_TRANSFER_START, BLOB_BLOCK_START, BLOB_CHUNK_TRANSFER and so on. BLOB_INFO_GET, BLOB_TRANSFER_START, BLOB_BLOCK_START, BLOB_CHUNK_TRANSFER, and so on. There are two modes, Push mode and Pull mode (see definition of MESH_OTA_TRANSFER_MODE_SEL), Pull mode is only used for LPN nodes.

- Push mode is where the host actively sends firmware data to the upgraded node, which has the advantage of being fast, but requires the node to be listening for mesh messages all the time.
- Pull mode is when the upgraded node is in receive mode, requesting firmware data from the host. applicable to LPN devices, the OTA time is relatively long, and only one node is being upgraded at the same time.

  The LPN supports both Push mode and Pull mode by default. the LPN can use Push mode only when it is in the GATT connection state.

(2) Firmware update server: This mod is used for firmware process control and contains commands such as FW_UPDATE_METADATA_CHECK, FW_UPDATE_START, FW_UPDATE_APPLY and so on.

(3) Firmware Distribution server: This model is mainly used to receive new firmware and receiver list from Initiator. It downloads the new firmware to local storage by receiving commands such as FW_DISTRIBUT_UPLOAD_START, FW_DISTRIBUT_RECEIVERS_ADD, BLOB_TRANSFER_START, etc.; and distributes the new firmware by receiving commands such as FW_DISTRIBUT_START,

BLOB_TRANSFER_START, etc. BLOB_TRANSFER_START commands to download the new firmware to local storage; distribute the new firmware by receiving FW_DISTRIBUT_START, BLOB_TRANSFER_START commands.

(4) BLOB Transfer client：Corresponds to the sender side of the server model.

(5) Firmware update client ： Corresponds to the sender side of the server model.

(6) Firmware Distribution client ： Corresponds to the sender side of the server model.

24.2 Test Mesh OTA with App

To test the mesh OTA using the App, please refer to the "4.4 Mesh OTA" section of the chapter Android and iOS APP User Guide.

24.3 Gateway Mesh OTA

The node to be upgraded is a non-LPN node.

24.3.1 Test and Command Sending and Receiving Process

24.3.1.1 Code Configuration

Code configuration test conditions: 8258 dongle 1 (burn 8258_mesh_gw.bin), 8258 dongle 2 (burn 8258_mesh.bin)

(1) Open MD_MESH_OTA_EN.

config_8278.h (vendor\mesh_provision) | app_mesh.h (proj_lib\sig_mesh) | mesh_config.h (vendor\common) | mesh

```
559:     #define CERTIFY_BASE_ENABLE     0
560:     #else
561:     #define CERTIFY_BASE_ENABLE     0
562:     #endif
563: #endif
564:
565: #ifndef MD_MESH_OTA_EN
566: #if (DEBUG_CFG_CMD_GROUP_AK_EN)
567: #define MD_MESH_OTA_EN              1   // just for internal test.
568: #elif (__PROJECT_MESH_PRO__)   // app & gateway
569:     #if WIN32
570: #define MD_MESH_OTA_EN              1
571:     #else // gateway
572: #define MD_MESH_OTA_EN              1   // default disable before released by SIG.
573:     #endif
574: #else
575:     #if ((MESH_USER_DEFINE_MODE == MESH_MI_ENABLE) || (LIGHT_TYPE_SEL == LIGHT_TYPE_PANEL) || SPIRIT_PI
576: #define MD_MESH_OTA_EN              0   // must 0
577:     #elif (MESH_USER_DEFINE_MODE == MESH_IRONMAN_MENLO_ENABLE)
578: #define MD_MESH_OTA_EN              0   //
579:     #elif (AIS_ENABLE)
580: #define MD_MESH_OTA_EN              0   // default disable
581:     #else
582: #define MD_MESH_OTA_EN              1   // dufault disable before released by SIG.
583:     #endif
584: #endif
585: #endif
```

gw

mesh

Figure 24.1: Open MD_MESH_OTA_EN

(2) To shorten the mesh ota time, the macro EXTENDED_ADV_ENABLE can be set to 1 to support the extended broadcast packet mode, which should be noted is not a spec-defined mode.

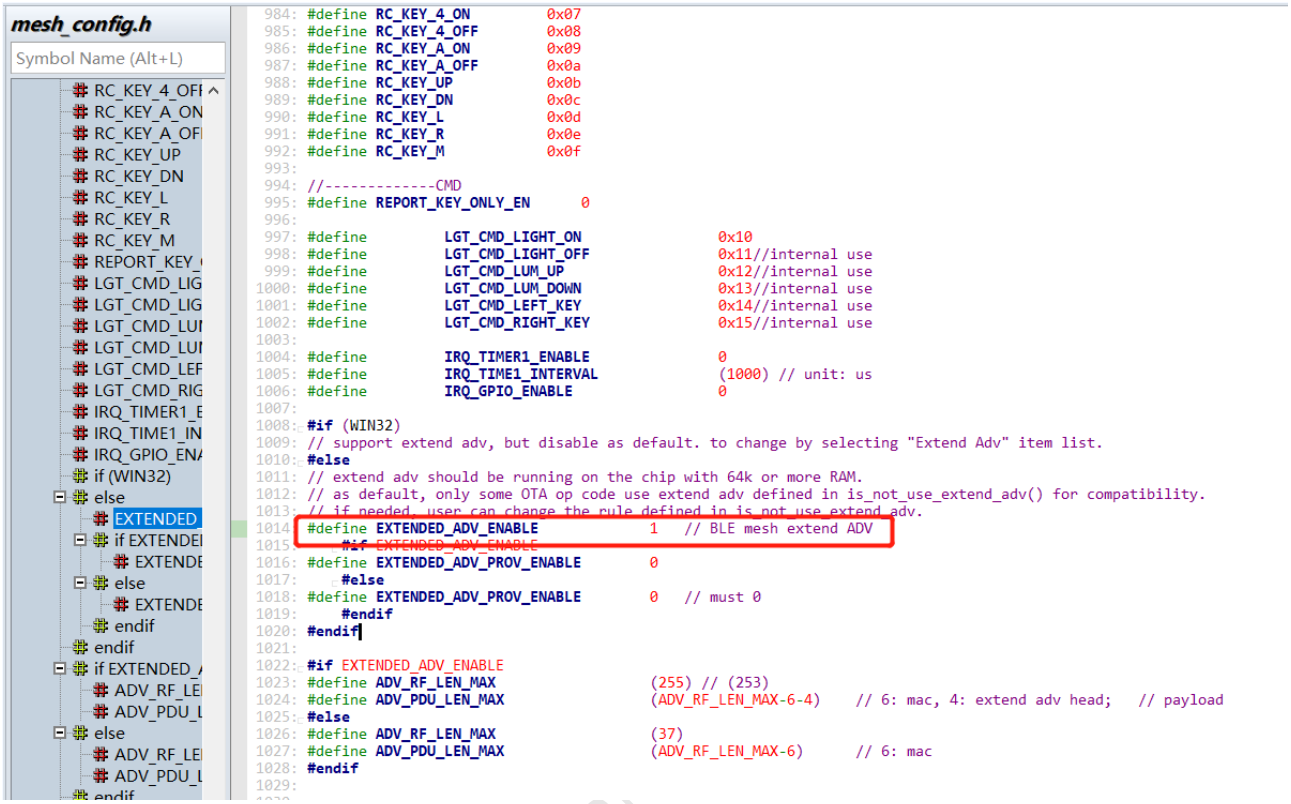


Figure 24.2: Open EXTENDED_ADV_ENABLE

(3) The host computer can set Extend Adv to enable the extended broadcast packet mode.

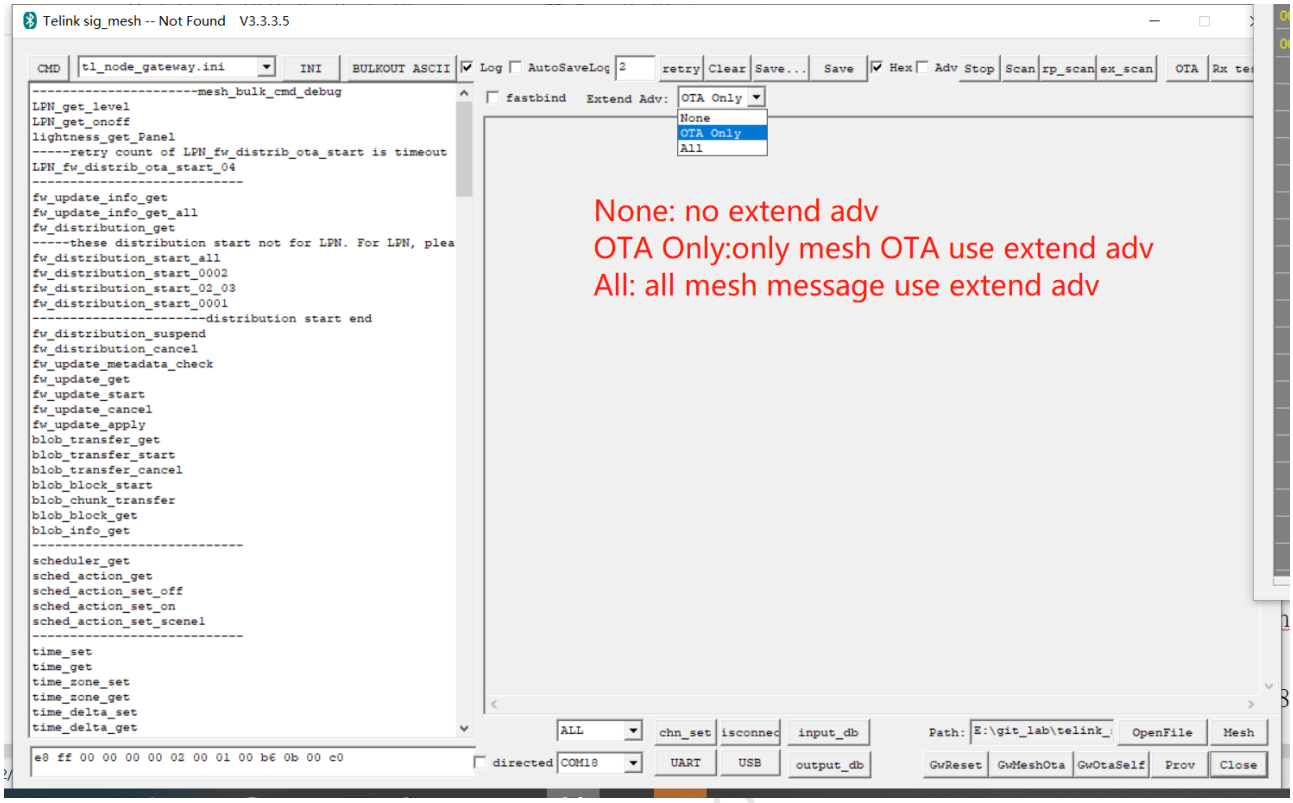


Figure 24.3: The host computer opens the extend_adv

24.3.1.2 Networking Nodes

Connect the pc through gw dongle, open tools tool sig_mesh_tool.exe, select the ini file corresponding to gw as following figure 1-1, and then network the 2 8258 dongles sequentially. After successful networking, click "MESH" button to open the mesh control interface, and automatically send LIGHTNESS_GET all command to get all the current nodes (you can also send ONOFF_GET).

HCI_CMD_GATEWAY_CMD + netkey index(2 bytes) + appkey index(2 bytes) + retry cnt + rsp_max + gateway addr + op + par

i.e.: e8 ff + netkey index + appkey index + retry cnt + rsp_max + gateway addr + 82 02 00 00

> Note: Unless otherwise specified, the ini format is hexadecimal in this document.

Make sure that all nodes are displayed in the UI, because if you want to do mesh OTA via fw_distribution_start_all later, the unicast address of the corresponding node is obtained from inside this UI, which is the source of the address list in the parameter area of FW_DISTRIBUT_START.

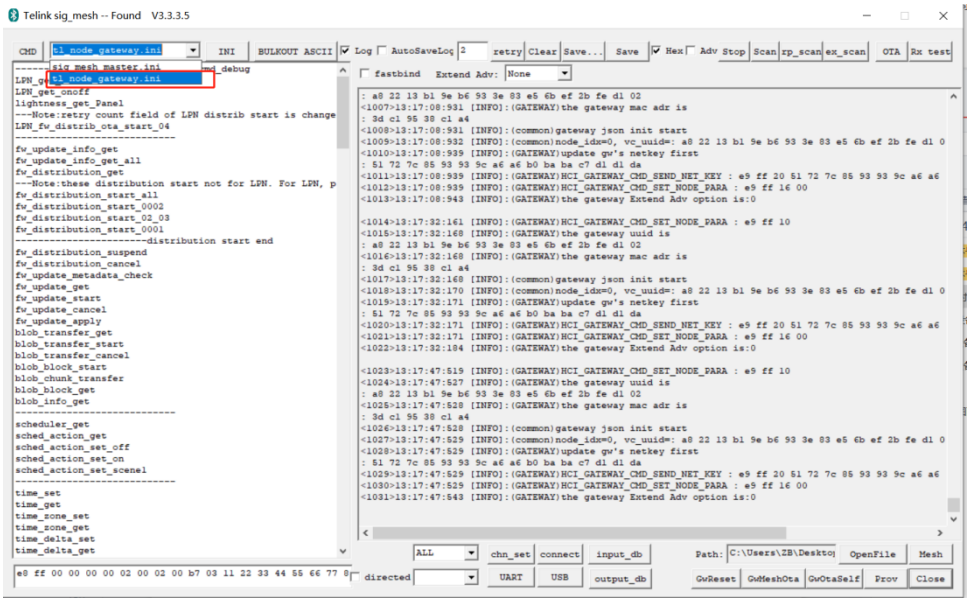


Figure 24.4: Network node

24.3.1.3 Select New Firmware

Click on the open file button in the figure, select the target firmware, and click confirm.

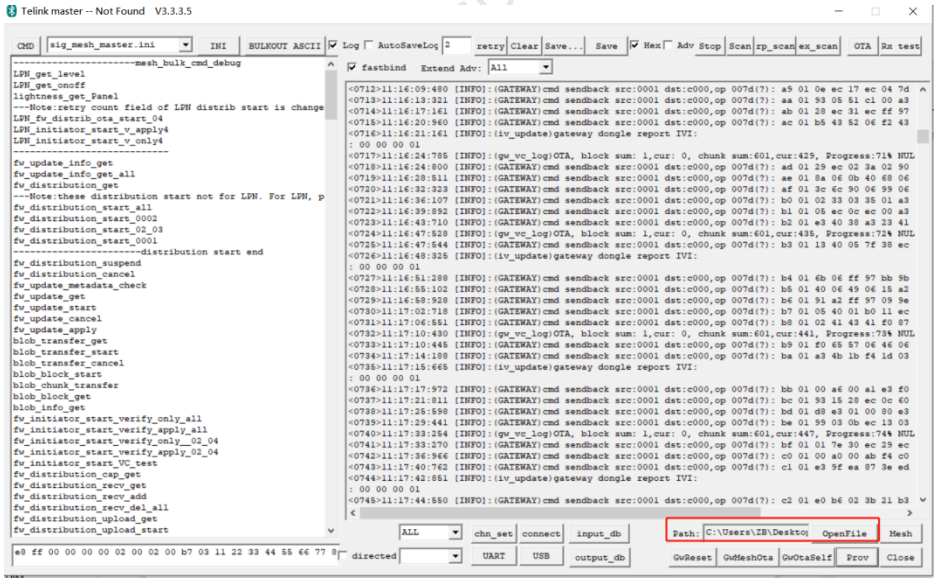


Figure 24.5: Select new firmware

24.3.1.4 Download New Firmware to Local Gateway Dongle

Click the "GwMeshOta" button, the host computer loads the selected new firmware from the target path to the gateway dongle for storing locally, so as to prepare for the subsequent mesh OTA. After the button is clicked, the host computer calls OnBnClickedGatewayOta to send firmware to the gateway, the corresponding ini commands are as follows:

(1) Set the ota type to GATEWAY_OTA_MESH in the format: HCI_CMD_MESH_OTA + MESH_OTA_SET_TYPE + GATEWAY_OTA_MESH. i.e.: eb ff + 01 + 00.

> Note: ota type GATEWAY_OTA_SLEF (01) is the upgrade gateway itself.

(2) Send the ota start command to wait for the ota area erase to complete. the default wait is 5 seconds. in the format: HCI_CMD_GATEWAY_OTA + len + CMD_OTA_START. i.e.: ea ff 02 01 ff.

(3) Send the firmware package in the format: HCI_CMD_GATEWAY_OTA + len + ota_index(2 bytes) + ota_payload(16 bytes) + crc16. i.e.: ea ff + 14 + ota_index + ota_packet(16 bytes) + crc16. Where pkt index is the index value of the ota packet, the size of each ota payload is 16 bytes, and the crc16 value is the crc16 checksum of ota_index and ota_payload.

(4) After the firmware is transferred, send the ota end command in the format: HCI_CMD_GATEWAY_OTA + len + CMD_OTA_END + index_max + ~(index_max). i.e.: ea ff + 6 + 02 ff + index_max + ~(index_max). Where index_max = (firmware_total_len + 15)/16 - 1, is the maximum index value of the ota packet, which is used to receive the ota end command whether to collect all or not.

Note:

> New firmware is temporarily stored in the pending OTA area of the gateway dongle (0x0000 or 0x40000), once the gateway dongle is rebooted, it will be cleared, and you need to perform the loading action again. Otherwise, the mesh ota cannot be performed subsequently.

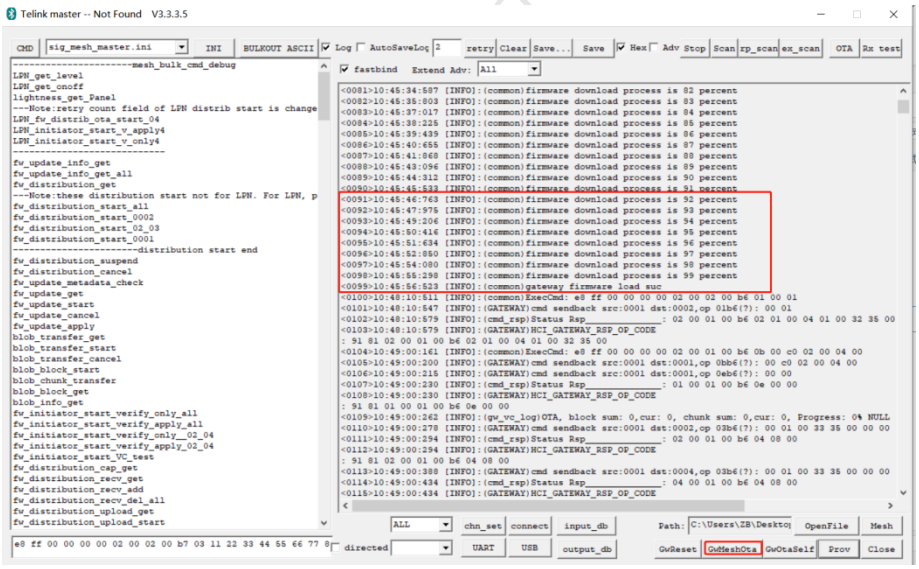


Figure 24.6: Download new firmware to local cache

24.3.1.5 Get the Version Information of the Nodes Currently on the Network

Users can query the version information of the nodes currently on the network via fw_updata_info_get, as shown in the following figure 02 The version number of the address device is displayed 41 00 (i.e., the ASCII code of 4.1 version "4", "1").

HCI_CMD_GATEWAY_CMD + netkey index(2 bytes) + appkey index(2 bytes) + retry cnt + rsp_max + dst addr + op + par

i.e.: e8 ff + netkey index + appkey index + retry cnt + rsp_max + dst addr + 83 08 00 01

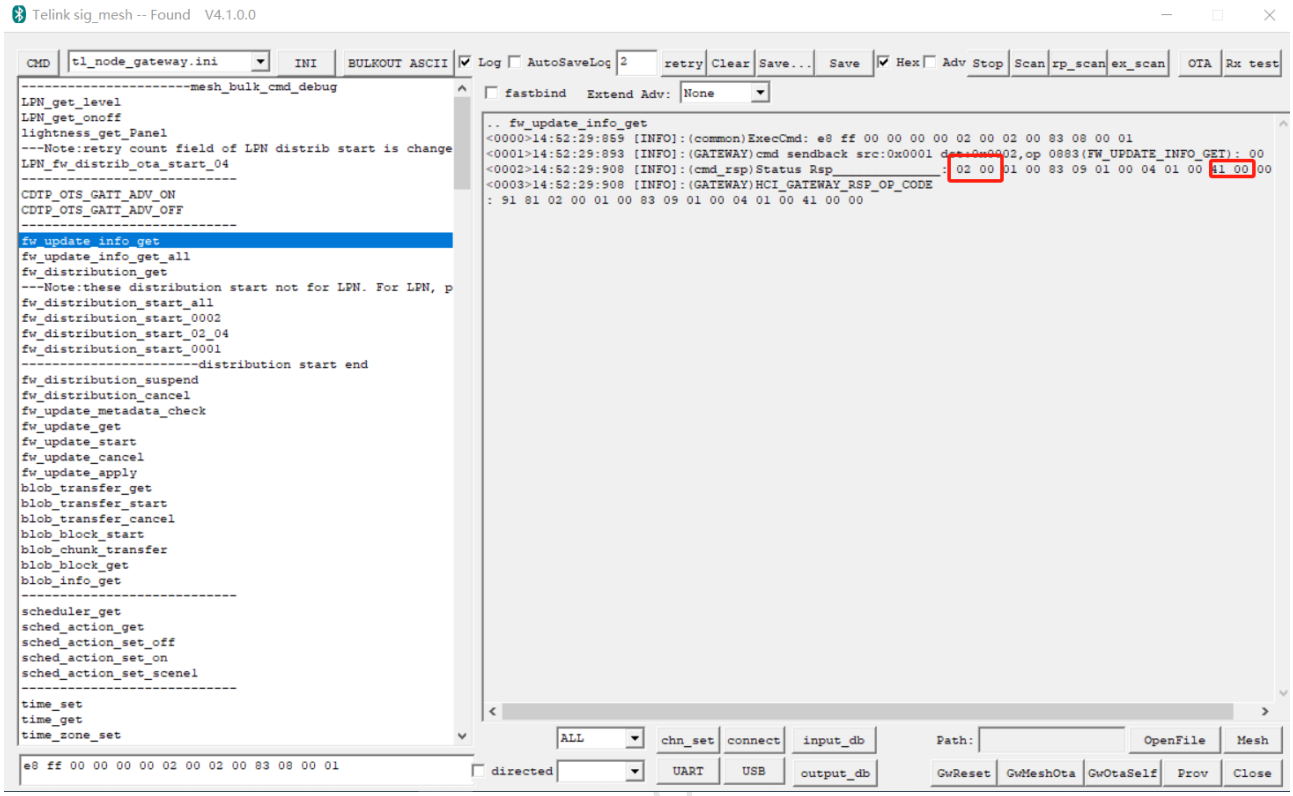


Figure 24.7: Get node information

24.3.1.6 Send fw_distribution_start_all Command

Double-click the fw_distribution_start_all command in the ini list, the host computer(SIG Mesh Tool on PC) will automatically read the unicast address of all current nodes from inside the UI list at vc_distribute_all_proc(), command format:

HCI_CMD_GATEWAY_CMD + netkey index(2 bytes) + appkey index(2 bytes) + retry cnt + rsp_max + gateway addr + op + par

i.e.: e8 ff + 00 00 + 00 00 + 02 + 00 + gateway_addr + 83 19 + group_addr + device_addr _list.

where gateway_addr is the gateway address, and the gateway will do mesh ota to the device specified by device_addr_list after adding it to group group_addr.

24.3.1.7 OTA Progress Reporting

The gateway receives distribution_start and starts the mesh ota process, mesh_cmd_sig_fw_distribute_start()->mesh_ota_master_proc(), which sends the firmware that is temporarily stored in the gateway to the updating node.

Gateway ota progress is currently reported in string format through the APP_RefreshProgressBar interface shown as below:

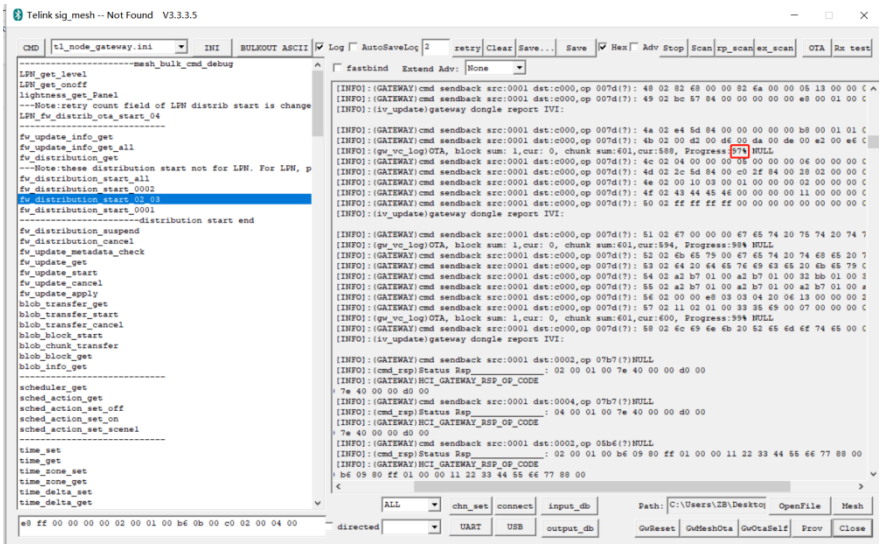


Figure 24.8: OTA progress reporting

24.3.1.8 Mesh OTA Completion Display Page

After ota has finished, report ota results via gateway_upload_mesh_ota_sts in the format:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_SEND_MESH_OTA_STS + fail_num + fail_list.

i.e.: 91 + 98 + fail_num + fail_list.

Where a fail_num of 0 indicates the number of failed upgrades, and 0 indicates all successes, the print page will show mesh OTA success as shown below.

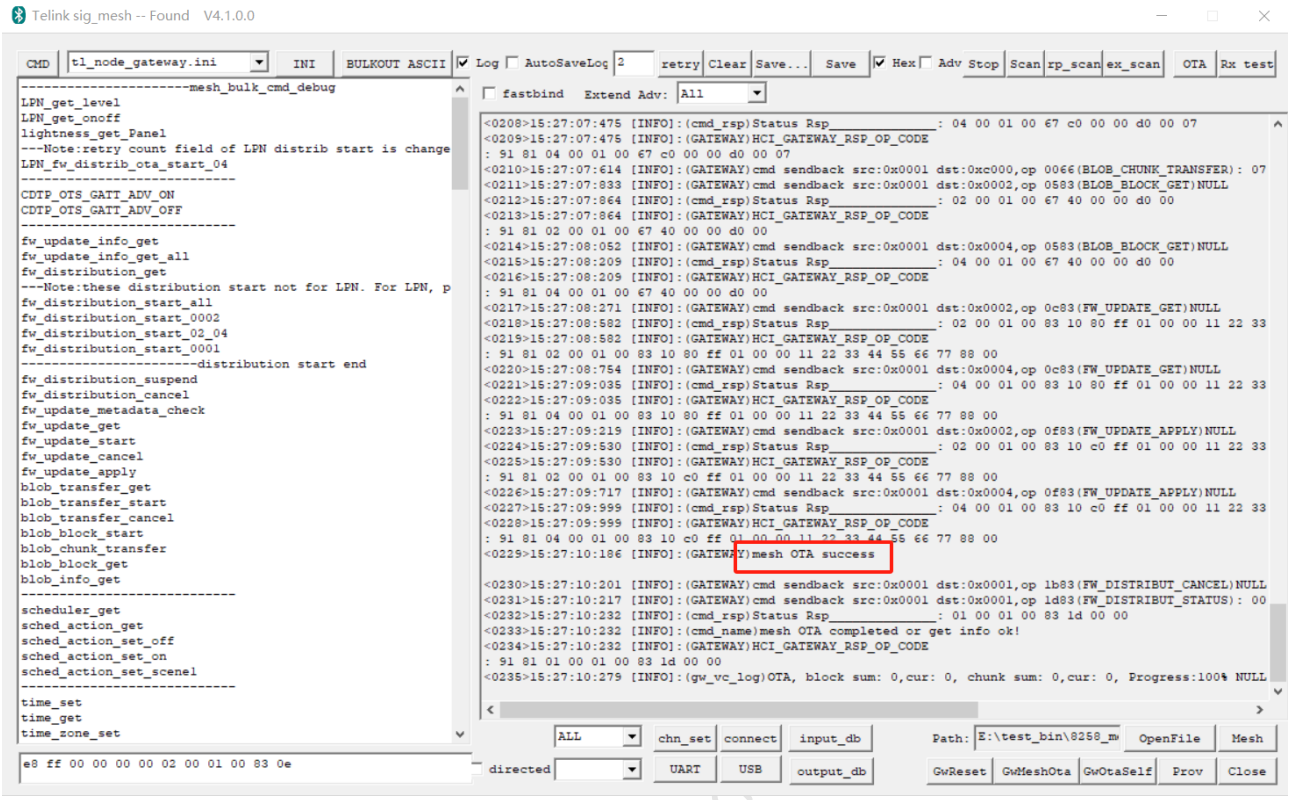


Figure 24.9: Mesh OTA completed

24.3.1.9 Device Flashes 6 Seconds Slowly

All devices will flash slowly for 6 seconds, and then reboot automatically to take effect new firmware. after reboot, you can query the version through step 5 above to confirm whether the version is upgraded successfully again, as shown below.

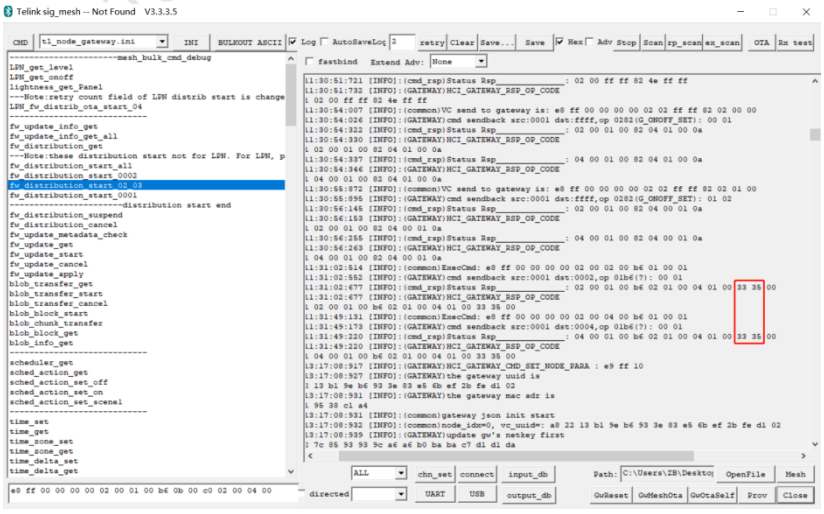


Figure 24.10: ota reboot

24.3.2 OTA Code Flow Summary

(1) The host computer loads fireware to the gw node.

(2) The host computer sends FW_DISTRIBUT_START to notify gw of the start of ota. The unicast address list which need to be OTA and the group address for OTA is included in parameters of FW_DISTRIBUT_START.

(3) gw receives the FW_DISTRIBUT_START, then callback to the corresponding function: mesh_cmd_sig_fw_distribut_s -> mesh_ota_master_proc() -> (Start to send firmware data)

24.3.3 Gateway OTA Flowchart

Gateway's processing flowchart in mesh_ota_master_proc() is shown below:

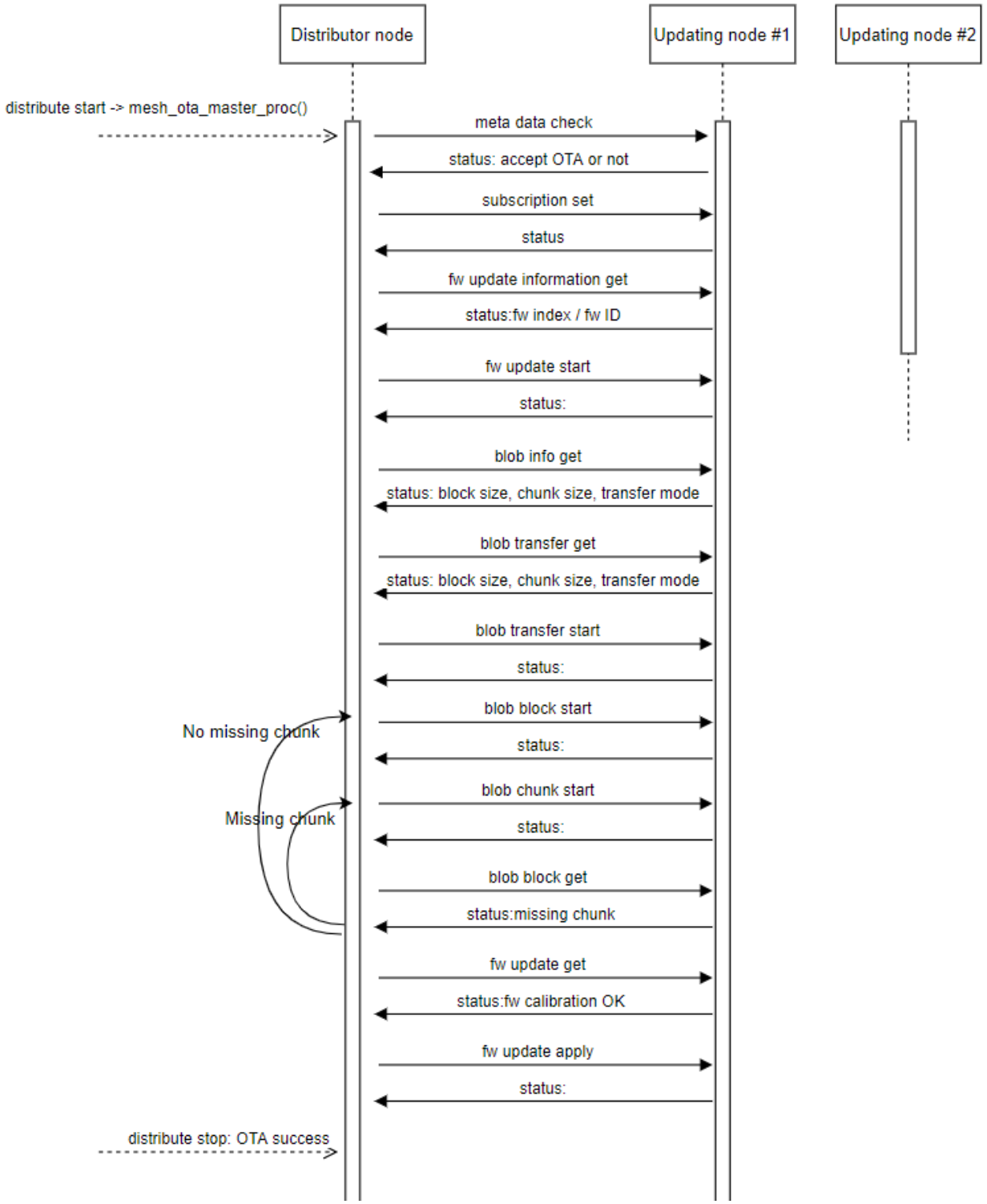


Figure 24.11: Gateway ota flowchart

24.3.4 Mesh OTA Related Commands

The structure corresponding to the INI command is mesh_bulk_ini_vc_t:

```
typedef struct{
    u16 nk_idx;
    u16 ak_idx;
    u8 retry_cnt;   // only for reliable command // for op "distribute start" of gateway mesh
↪   OTA, it is reliable retry interval for LPN. // retry_intv_for_lpn_100ms
    u8 rsp_max;     // only for reliable command
    u16 adr_dst;
    u8 op;
    u8 par[MESH_CMD_ACCESS_LEN_MAX];
}mesh_bulk_cmd_par_t;

typedef struct{
    u16 flag;
    mesh_bulk_cmd_par_t cmd;
}mesh_bulk_ini_vc_t;
```

The following is a description of the commands used in the flowchart Gateway ota flowchart. Please refer to the "messages" section in the spec "MshDFU_v1.0.pdf" and "MshMBT_v1.0.pdf" for the parameters of each command.

24.3.4.1 FW_DISTRIBUT_START

The initiator (host) sends this command to the distributor, which receives it and starts executing the distribution of firmware.

Note

> These distributor starts in the INI are private commands.

```
fw_distribution_start_all   =a3 ff 00 00 00 00 00 00 01 00 83 19 00 c0
```

The opcode "83 19" is the spec-defined distributor start opcode, but when the first two bytes of the following argument are a group address, it is recognised in private format. See is_par_distribute_start_tlk() in mesh_cmd_sig_fw_distribut_start() for more details, in order to be compatible with the earlier command format. When is_par_distribute_start_tlk() returns ture, the corresponding parsing format of the parameter area is:

```
typedef struct{
    u16 adr_group; // Destination address to be used when sending firmware data.
    u16 update_list[MESH_OTA_UPDATE_NODE_MAX]; // The unicast address list of the node to be
↪   upgraded
}fw_distribut_start_tlk_t;
```

The update_list of fw_distribution_start_all is empty, which means it needs to be auto-populated with the node list of the host computer's "Mesh" window.

If the update_list for fw_distribution_start is not empty, the example is as follows:

```
CMD-fw_distribution_start_0002  =a3 ff 00 00 00 00 00 01 00 83 19 00 c0 02 00
CMD-fw_distribution_start_02_04 =a3 ff 00 00 00 00 00 01 00 83 19 00 c0 02 00 04 00
```

The unicast address list of the node to be upgraded is specified by the INI command and does not need to be auto-populated. However, it should be noted that when the number of addresses to be populated exceeds the default value of MESH_OTA_UPDATE_NODE_MAX, you need to modify MESH_OTA_UPDATE_NODE_MAX.

Also, since when doing a Mesh OTA for an LPN, only one LPN node can be upgraded at a time, it is used:

```
LPN_fw_distrib_ota_start_04 =a3 ff 00 00 00 00 32 00 01 00 83 19 00 00 04 00
```

The corresponding parsing format of the parameter area is also fw_distribut_start_tlk_t, except that the adr_group should be set to 0, and update_list[0] is the unicast address of the LPN. In addition, the reliable retry count is "32". "For LPN_fw_distrib_ota_start, it is not the reliable retry count, but the reliable retry interval in unit 100ms, so 0x32 here means 5000ms, please see the comment of member retry_cnt of mesh_bulk_cmd_par_t. _cnt.

24.3.4.2 FW_UPDATE_METADATA_CHECK

Distributor sends this command to the node to be upgraded, which contains the firmware id, which reads the contents of the second to fifth bytes of the new firmware, corresponding to pid (product id) and vid (version id), see get_fw_metadata() for details. When the node to be upgraded receives this command, it checks the received firmware id and its own pid vid for comparison, when the pid is the same, it replies with the value of METADATA_CHECK status as success, which suggests that the distributor can carry out OTA, and if the pid is not the same, it replies with the value of not allowed to carry out OTA, for more details, please refer to mesh_ cmd_sig_fw_update_metadata_check() of mesh_ota_slave_need_ota() for more details, if you want to change to other rules, please modify this function ota_is_valid_pid_vid().


```
/**
 * @brief       This function check if new firmware has a valid PID(product ID) and VID(Version
 ↪  IS).
 * @param[in]   p_fw_id     - firmware ID
 * @param[in]   gatt_flag   - 1: it is GATT OTA.
 * @return
 * @note        for both GATT and MESH ADV OTA
 */
_USER_CAN_REDEFINE_ int ota_is_valid_pid_vid(fw_id_t *p_fw_id, int gatt_flag)
{
    #if (OTA_ADOPT_RULE_CHECK_PID_EN)
    // user can change this policy
    int accept = 0;
    if(p_fw_id->pid == fw_id_local.pid){
        #if OTA_ADOPT_RULE_CHECK_VID_EN
        sw_version_big_endian_t *p_new = (sw_version_big_endian_t *)p_fw_id;
```

```
        sw_version_big_endian_t *p_local = (sw_version_big_endian_t *)&fw_id_local.pid;
        u16 ver_new_little = get_little_end_version(p_fw_id->pid);
        u16 ver_local_little = get_little_end_version(fw_id_local.pid);
        if(ver_new_little > ver_local_little){
            accept = 1;
        }
        #else
        accept = 1;
        #endif
    }
    return accept;
    #else
    return 1;
    #endif
}
```

24.3.4.3 CFG_MODEL_SUB_ADD

Corresponding flowchart Gateway ota flowchart The comment inside is "subscription set".

Distributor sends this command to the pending upgrade. When the pending upgrade node receives this command, it subscribes to the group number for SIG_MD_BLOB_TRANSFER_S, so that when the OTA sends the firmware data in the future, it can use the group address as the destination address, and send the firmware data to all of the pending upgrade nodes at the same time.

24.3.4.4 FW_UPDATE_INFO_GET

Distributor gets the firmware information of the node to be upgraded, which mainly contains information such as firmware id.

24.3.4.5 FW_UPDATE_START

Distributor sends this command to the node to be upgraded to indicate that firmware update is to be started.

24.3.4.6 BLOB_INFO_GET

Distributor sends this command to get information of the node to be upgraded. include block size，chunk size，transfer mode, etc.

24.3.4.7 BLOB_TRANSFER_START

Distributor sends this command to the node to be upgraded to inform the node of the size of the new firmware and the block size and chunk size parameters to be used, and to start the BLOB data sending process.

24.3.4.8 BLOB_BLOCK_START

The distributor sends this command to the node to be upgraded, informing the node which block data is about to be sent, etc.

24.3.4.9 BLOB_CHUNK_TRANSFER

Distributor sends this command to the node to be upgraded to transmit firmware data.

24.3.4.10 BLOB_BLOCK_GET

After the distributor transmitting firmware data is completed, send this command to query all the nodes to be upgraded to see if there is any packet loss, if there is, it will re-send BLOB_BLOCK_START and BLOB_CHUNK_TRANSFER to make up for the packet loss until all the nodes have finished receiving.

24.3.4.11 FW_UPDATE_GET

After the distributor confirming that all the nodes to be upgraded have received the firmware data, send this command and the nodes to be upgraded will perform CRC checksums and return the checksum value.

24.3.4.12 FW_UPDATE_APPLY and FW_UPDATE_CANCEL

If the returned result is successful, the distributor sends FW_UPDATE_APPLY to the node to notify it to reboot and enable the new firmware. If the result is a failure, the distributor sends FW_UPDATE_CANCEL to the node, informing it to reboot and discarding the firmware data it just received.

24.4 Gatt master dongle mode mesh OTA (kma_dongle)

The node to be upgraded is a non-LPN node.

Test conditions：1 x 8269 dongle (burn 8269kma_master_dongle), 2 x 8258 dongles.

24.4.1 Code Configuration

The default mesh ota function is not enabled on the node side, the way to enable it:change MD_MESH_OTA_EN from 0 to 1, as follows：

```
#if (__PROJECT_MESH_PRO__)    // app & gateway
#define MD_MESH_OTA_EN              1
#else
    #if ((MESH_USER_DEFINE_MODE == MESH_MI_ENABLE) || (LIGHT_TYPE_SEL == L
#define MD_MESH_OTA_EN              0    // must 0
    #elif (AIS_ENABLE)
        #if DUAL_VENDOR_EN
#define MD_MESH_OTA_EN              1    // decrease firmware size
        #else
#define MD_MESH_OTA_EN              1    // enable for genius
        #endif
    #else
#define MD_MESH_OTA_EN              1    // dufault disable before released by SIG.
    #endif
#endif
```

Figure 24.12: Open MD_MESH_OTA_EN

If ota upgrade selects the directly connected node as the distributor's mode, you need to change DISTRIBUTOR_UPDATE_SERVER_EN from 0 to 1, as shown below:

```
#if (MD_MESH_OTA_EN)
    #if(__PROJECT_MESH_PRO__ || __PROJECT_MESH_GW_NODE__)
#define DISTRIBUTOR_UPDATE_CLIENT_EN    1
#define DISTRIBUTOR_UPDATE_SERVER_EN    DISTRIBUTOR_UPDATE_CLIENT_EN
        #if (WIN32 && DISTRIBUTOR_UPDATE_CLIENT_EN)
#define INITIATOR_CLIENT_EN             1   // only app or VC enable
        #endif
    #else
#define DISTRIBUTOR_UPDATE_CLIENT_EN    0   // must 0
        #if (__PROJECT_MESH__)              // lpn, switch should not enable
#define DISTRIBUTOR_UPDATE_SERVER_EN    1   // include distribute server, update client and blob transfer
        #endif
    #endif
#endif
```

Figure 24.13: Enable server distributor

24.4.2 Networking Nodes

Through the master dongle connection pc, open tools tool for two 8258 dongle in turn for networking, networking success, connect one of them, click on the "MESH" button to open the mesh control interface, and automatically get all the current nodes, to confirm that all the nodes are displayed in the UI (because the subsequent mesh OTA node's unicast addr is based on the UI inside the access, that is, the source of the address list in the parameter area of the FW_DISTRIBUT_START).

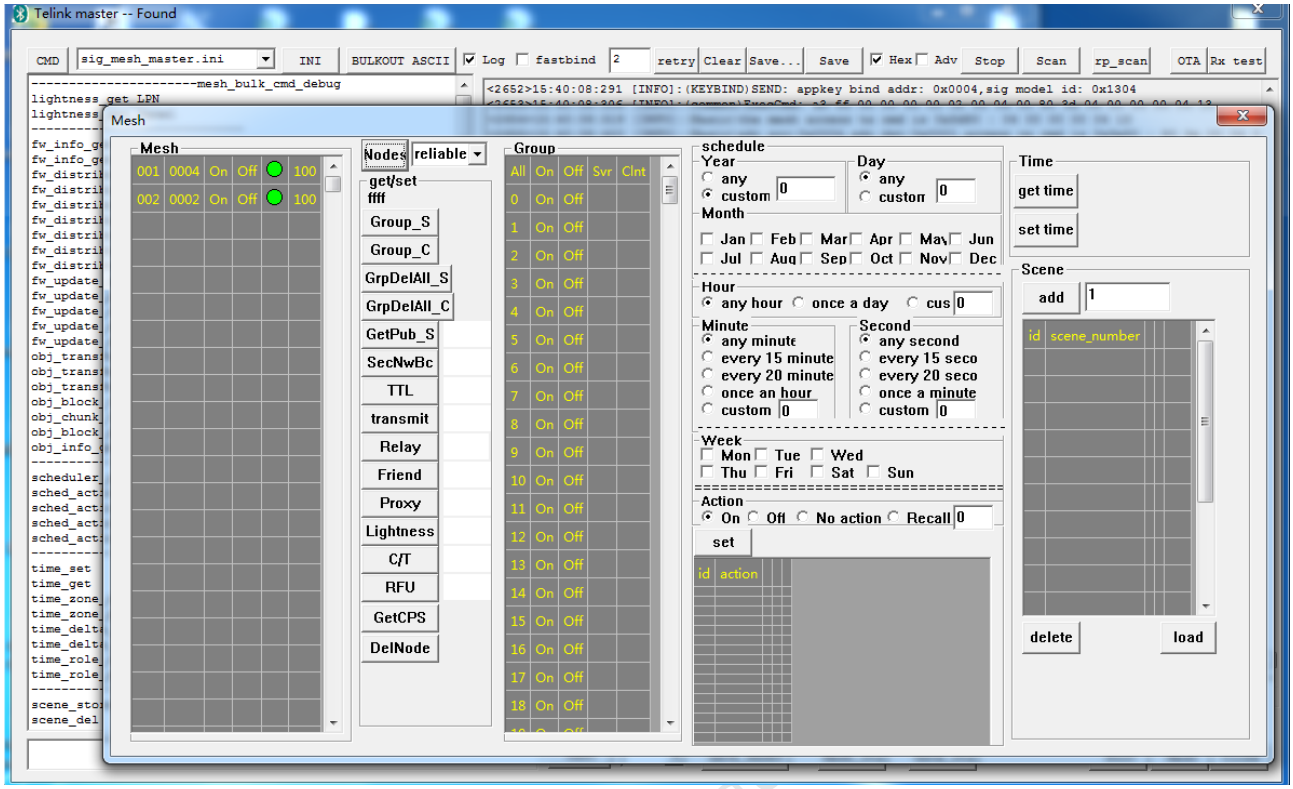


Figure 24.14: mesh_UI

24.4.3 Select New Firmware

Close the mesh page to return to the home page, as shown in the following figure, click on the search file button in the figure, select the target firmware, click on the confirmation, after the confirmation as shown in mark 2.

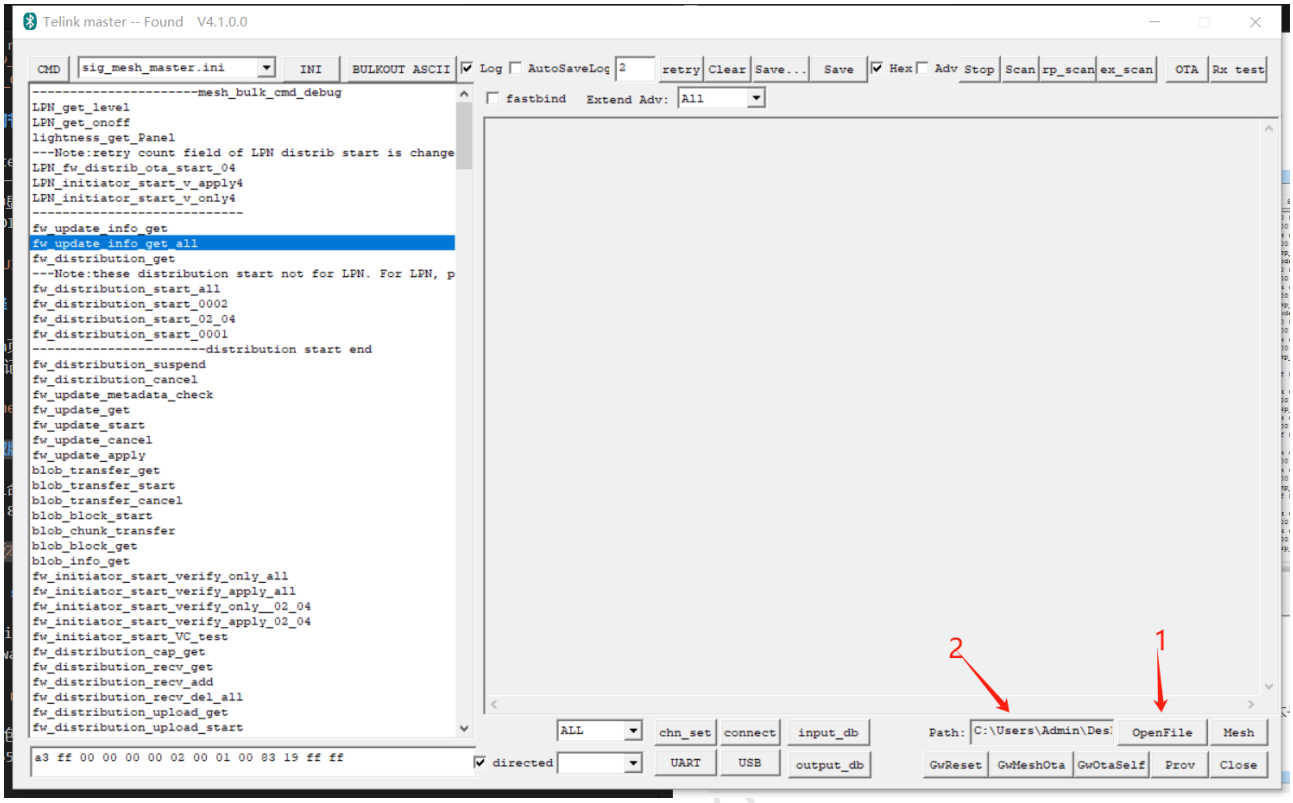


Figure 24.15: Select new firmware

24.4.4 Get Version

Click the ini command :fw-update-info-get-all to get the version of the devices currently on the network, as shown in the following figure, the version number of the two devices shows 32 38 (i.e., version 2.8), for details, please refer to the corresponding description of the gateway mesh ota.

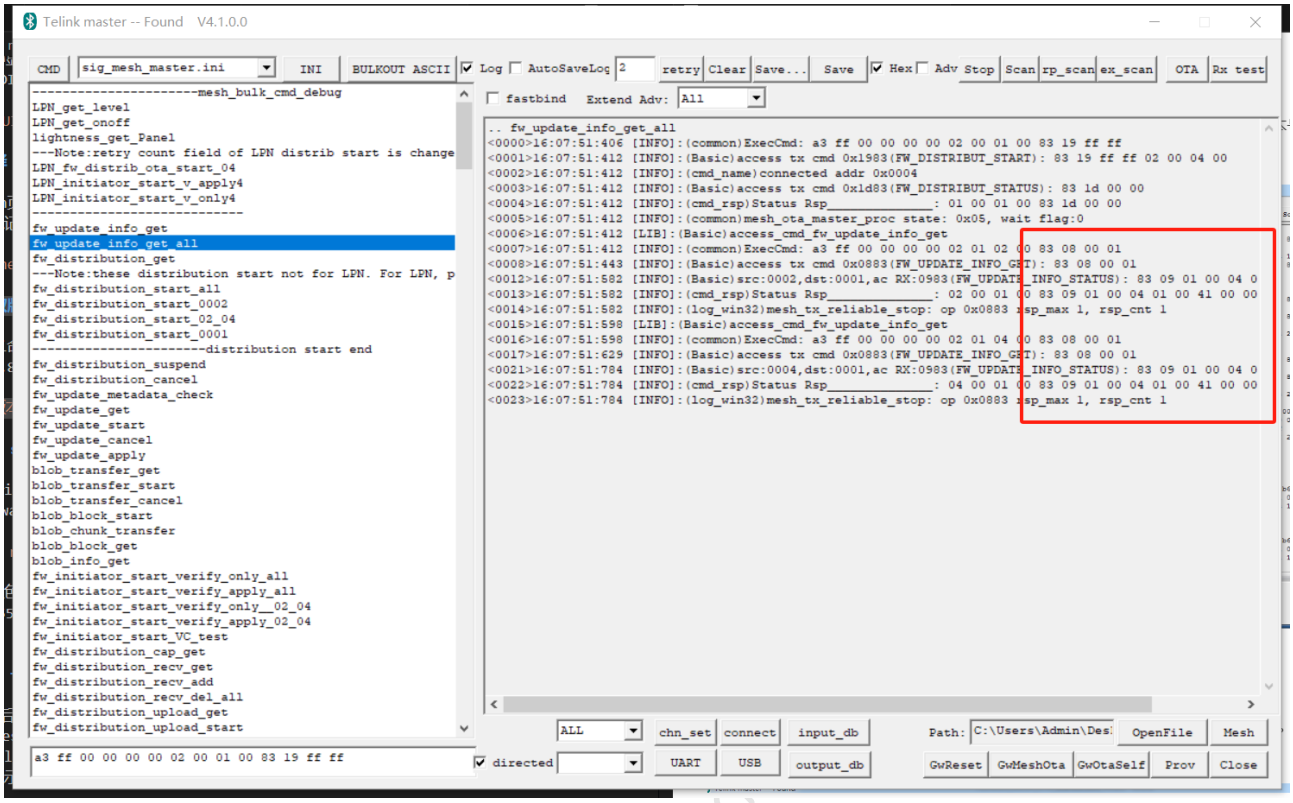


Figure 24.16: Get version

24.4.5 OTA Start

When testing, choose one of the following 3 ways to test.

(1) Upper computer as distributor mode

The host will transmit firmware data directly to the target node. This mode requires the master dongle and the GATT connected node to be in GATT connection state at all times.

Double-click the ini command :CMD-fw_distribution_start_all to start the mesh OTA.

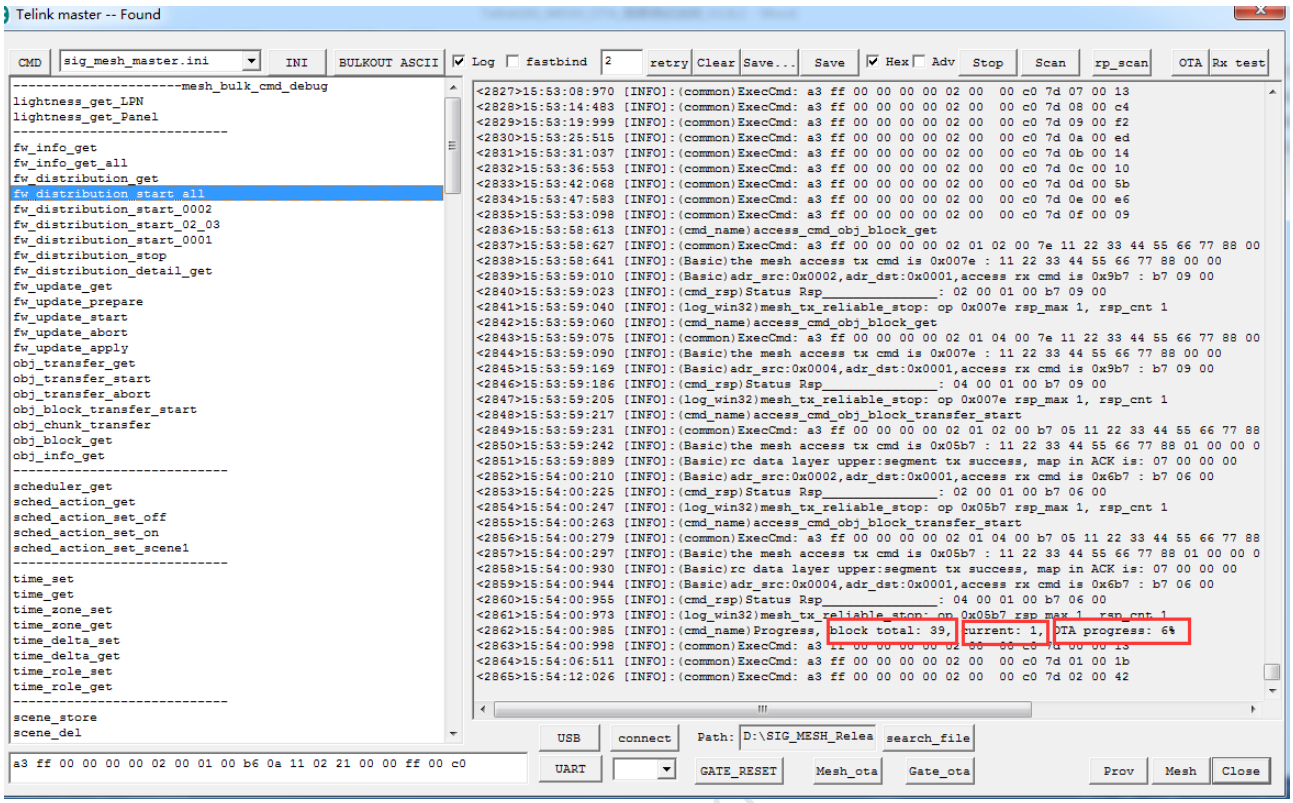


Figure 24.17: start mesh OTA

(2) The directly connected node acts as distributor and selects verify and apply mode.

Instead of directly transmitting the firmware data to the node to be upgraded, the host computer first quickly transmits the firmware to the directly connected node through GATT mode, and then the GATT connected node transmits the firmware to the node to be upgraded. during the transmission to the node to be upgraded, the host computer can disconnect the GATT connection.

Since verify and apply mode is selected, the directly connected node will send the firmware update apply command to the node to be upgraded after distributing the firmware and confirming that it has been received, so that the node to be upgraded can reboot and take effect of the new firmware.

Double-click the ini command:CMD-fw_initiator_start_verity_apply_all to start the mesh OTA. Command Format:(refer to structure fw_initiator_start_t)

HCI_CMD_MESH_OTA_INITIATOR_START + distribute adress + distribute appkey index + distribute ttl + timeout + distribute transfer mode + group + upload ttl + upload timeout + upload blob id

i.e.: ab ff + 00 00 + ff ff + ff + ff ff + 05 + 00 00 + 00 c0 + ff + ff ff + 61 62 63 64 65 66 67 68

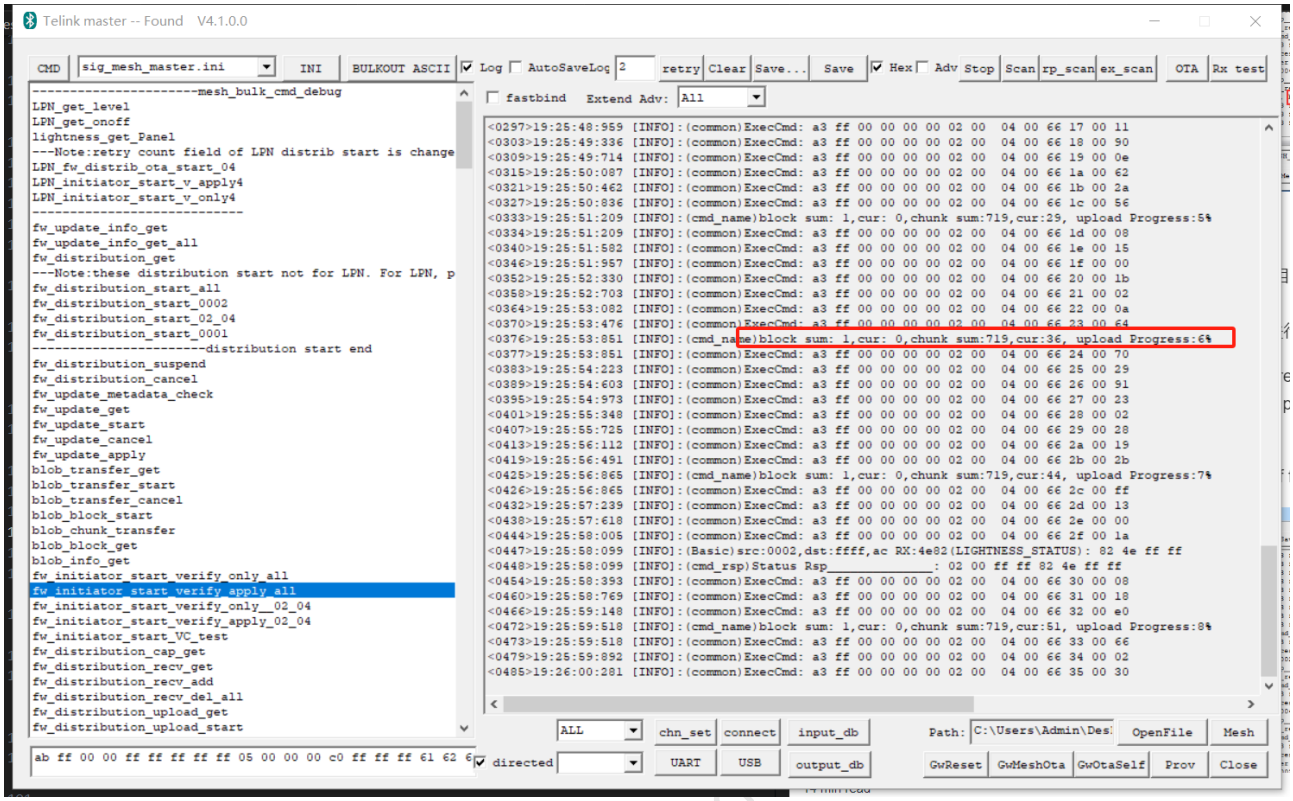


Figure 24.18: verify apply

(3) The directly connected node acts as distributor and selects verify only mode.

The difference between this mode and the "select verify and apply mode" is that the directly connected node will not send the distribution apply command to the node to be upgraded after distributing the firmware and confirming that it has been received. Instead, it waits for the host to send the distribution apply command before the node to be upgraded takes effect with the new firmware.

Currently, for ease of use, when the host computer receives a report from the distributor (directly connected node) that the distribution is complete, the host computer will automatically trigger the distribution apply command to be sent to the distributor. if you want to cancel this automatic sending, you can modify the code of " mesh_ota_initiator_proc() –> case INITIATOR_OTA_ST_DISTR_PRE_APPLY". For example, change it to nothing, and wait until the apply command is sent manually, and then set it to INITIATOR_OTA_ST_DISTR_GET state.

Double-click the ini command:CMD-fw_initiator_start_verity_only_all to start mesh OTA. Command Format: (The difference from verify apply mode is that the value of distribute transfer mode is not the same.)

HCI_CMD_MESH_OTA_INITIATOR_START + distribute adress + distribute appkey index + distribute ttl + timeout + distribute transfer mode + group + upload ttl + upload timeout + upload blob id

i.e.: ab ff + 00 00 + ff ff + ff + ff ff + 01 + 00 00 + 00 c0 + ff + ff ff + 61 62 63 64 65 66 67 68

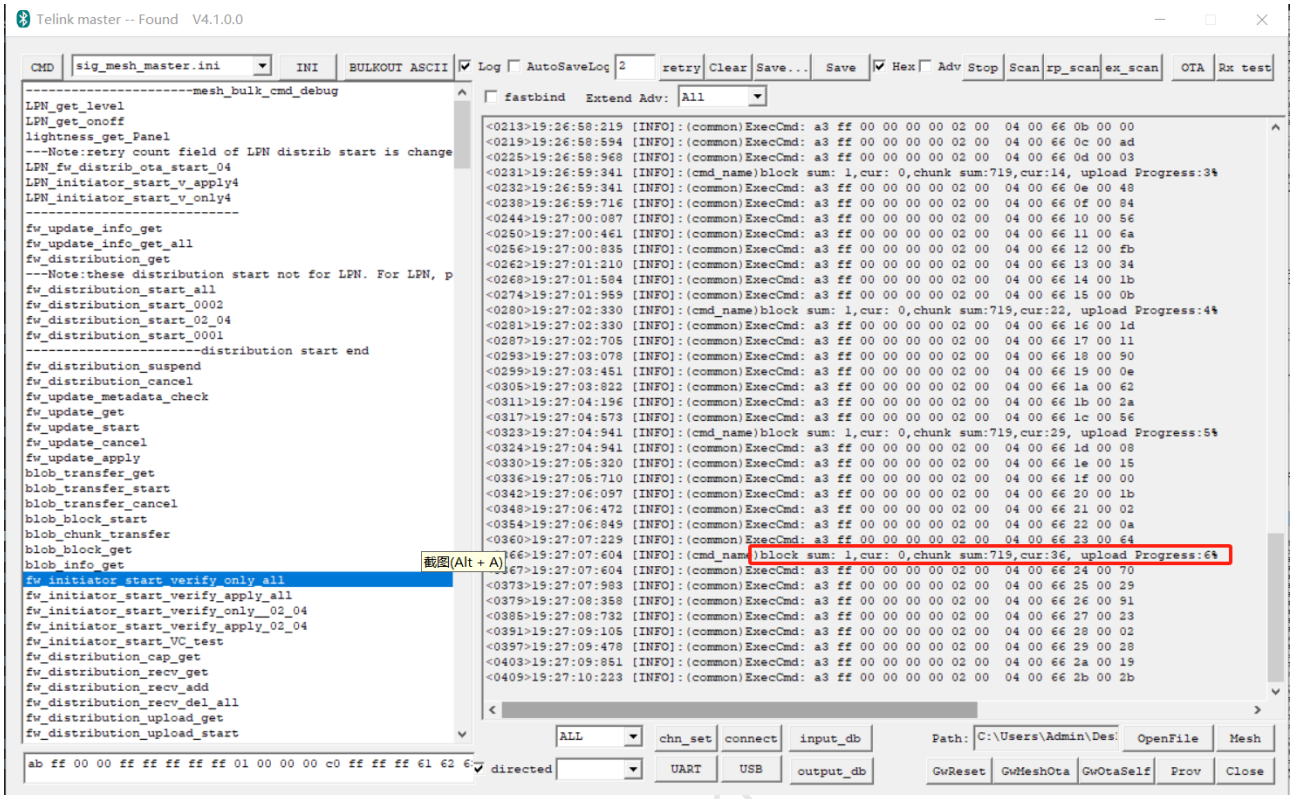


Figure 24.19: verify only

The red box in the above figure is the progress reference value: cur:1 indicates that the current is the 1st bolck; every 256k is 1 block, 147k firmware is 147/256 (such as block total:1 in the figure), total chunk:719 indicates that a total of 719 chunks are to be transmitted, cur:4 indicates that the current transmitted chunk is the 4th, OTA process:7% indicates that the current progress of ota is 7%.

24.4.6 OTA Finish

OTA is completed, the two devices will be applied, the print page will appear distribution completed; flow completed, the VC tool page to stop printing, the tick in front of the log will be removed to close the log printing, at the same time will pop up a small prompt box log disable now, indicating that the OTA is complete. log off is In order to avoid the log being flushed out during the upgrade process, it is convenient to check the log and save it, as shown in the following figure.

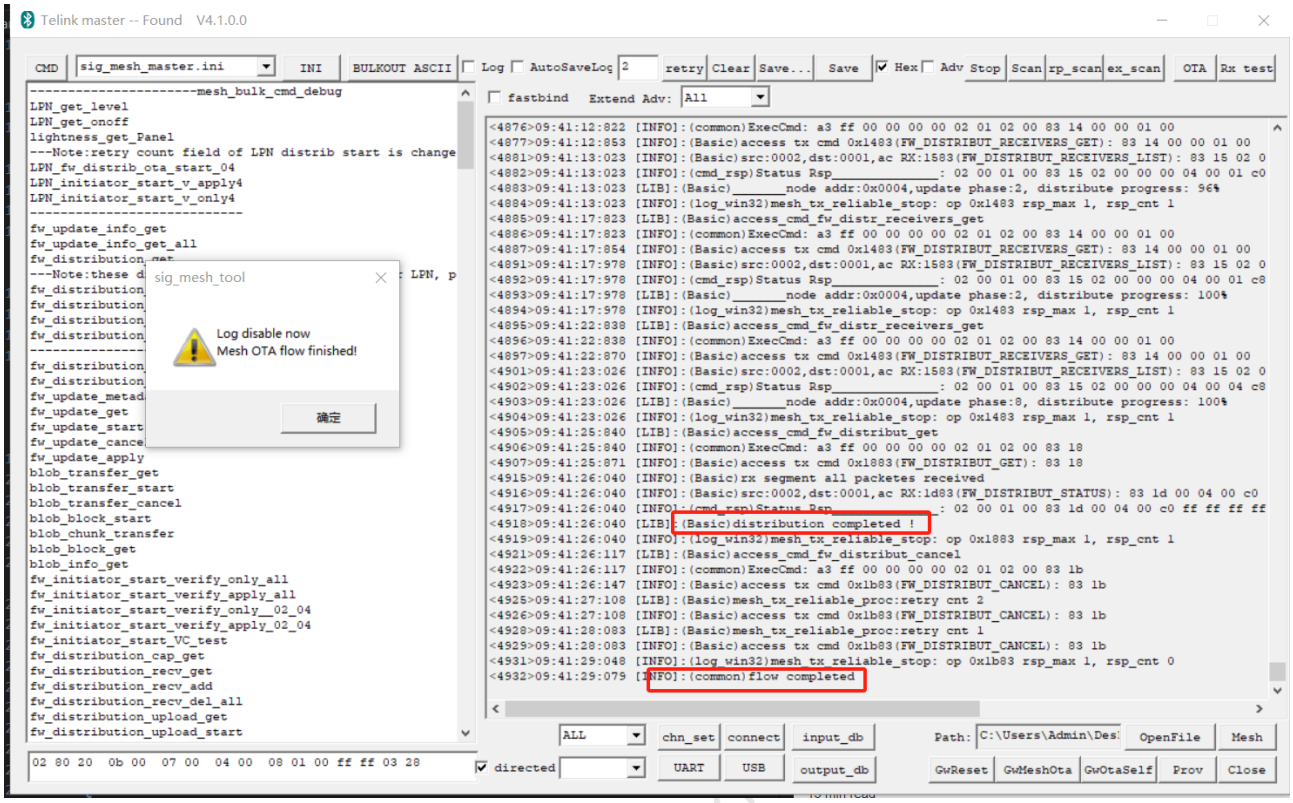


Figure 24.20: kma ota finish

24.4.7 Recover Log

All devices will flash slowly for 6 seconds, and then it will reboot automatically, after reboot, confirm the log, if necessary, you can tap the "Save" button to save the log, VC tools "log" control check box, so that the print is back to normal.

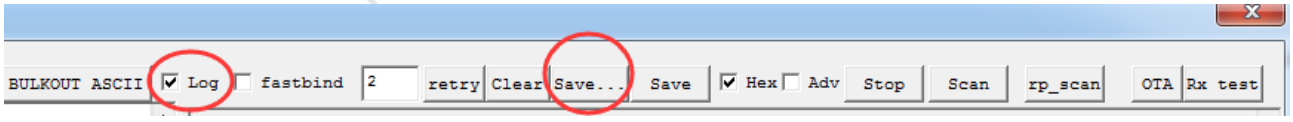


Figure 24.21: Recover log

24.4.8 Check for Success

Reconnect to the network, you can check the version through the above step 3 to confirm whether the version is upgraded successfully, as shown below.

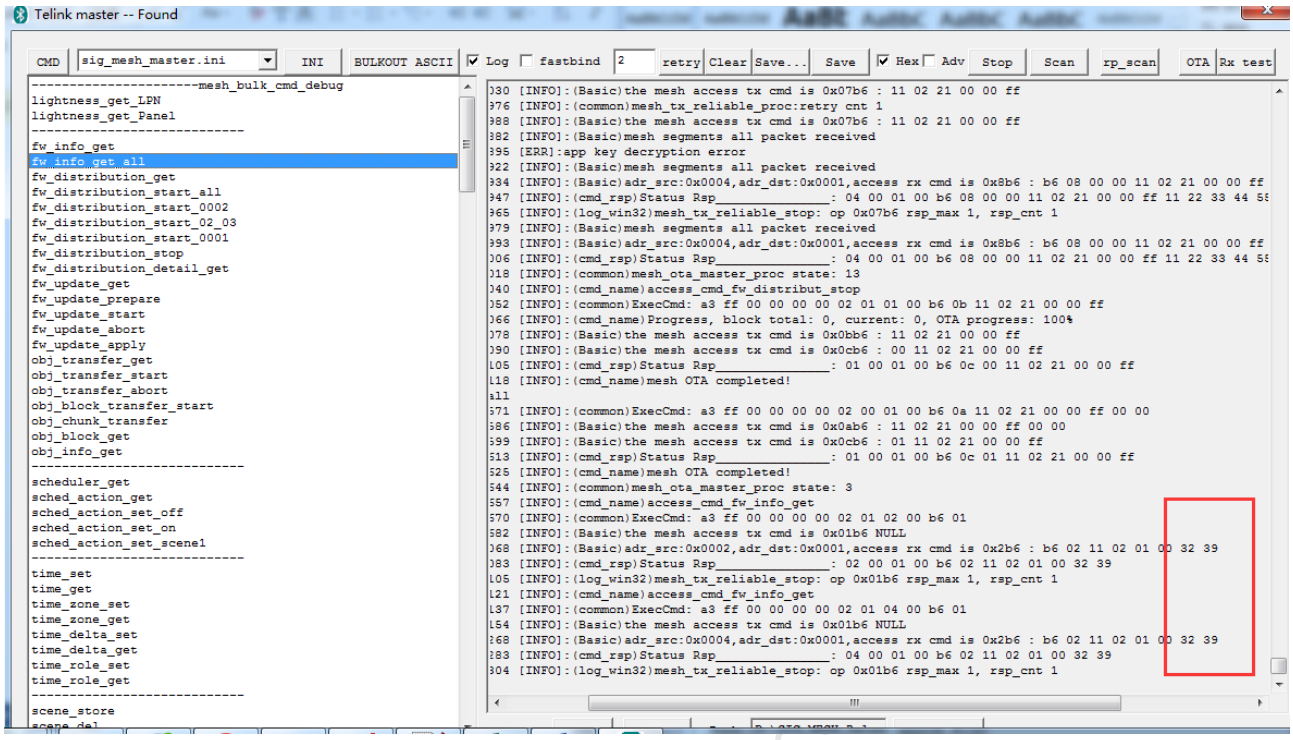


Figure 24.22: Check for success

24.5 LPN Mesh OTA

24.5.1 LPN Mesh OTA Gateway Mode Operation Procedure

Test conditions: 1 x 8258 dongle (burning 8258_mesh_gw.bin), 2 x 8258 dongles (burning 8258_mesh_LPN.bin)

24.5.1.1 Code Configuration

(1) Open MD_MESH_OTA_EN

(2) To shorten the mesh ota time, the macro EXTENDED_ADV_ENABLE can be set to 1 to support the extended broadcast packet mode, which should be noted is not a spec-defined mode.

(3) If the gateway node is not selected as a friend, the gateway node should have FEATURE_FRIEND_EN set to 0.

```
#define FEATURE_LOWPOWER_EN     0
#define FEATURE_PROV_EN         1
#define FEATURE_RELAY_EN        (0 || SWITCH_ALWAYS_MODE_GATT_EN)
#define FEATURE_PROXY_EN        1
#else
    #if DU_ENABLE
#define FEATURE_FRIEND_EN       0
    #else
#define FEATURE_FRIEND_EN       0   // WIN 32 should be support disable
    #endif
```

Figure 24.23: Switch off the friend function

24.5.1.2 Networking Nodes

Refer to the Gateway mesh ota networking process.

Note:

> If the LPN node is not shown in the UI after the networking is completed, you can click this INI command to get it (you need to pay attention to whether the destination address is correct, the default is 0x0004), CMD-LPN_get_onoff. Another way is that the power down the LPN node and then re-power-up operation, the LPN will actively send the current status once, and the UI interface will display the LPN node.

24.5.1.3 Select New Firmware

Refer to the Gateway mesh ota selecting new firmware process.

24.5.1.4 Get Version

Refer to the Gateway mesh ota selection to obtain the version process.

24.5.1.5 OTA Start

Click LPN_fw_distrib_ota_start_04, then modify the last two bytes of the command window below to the unicast address of the LPN node. and then click "Enter" in keyboard of PC to send this INI command.

i.e. 02 00, and hit enter to send this command.

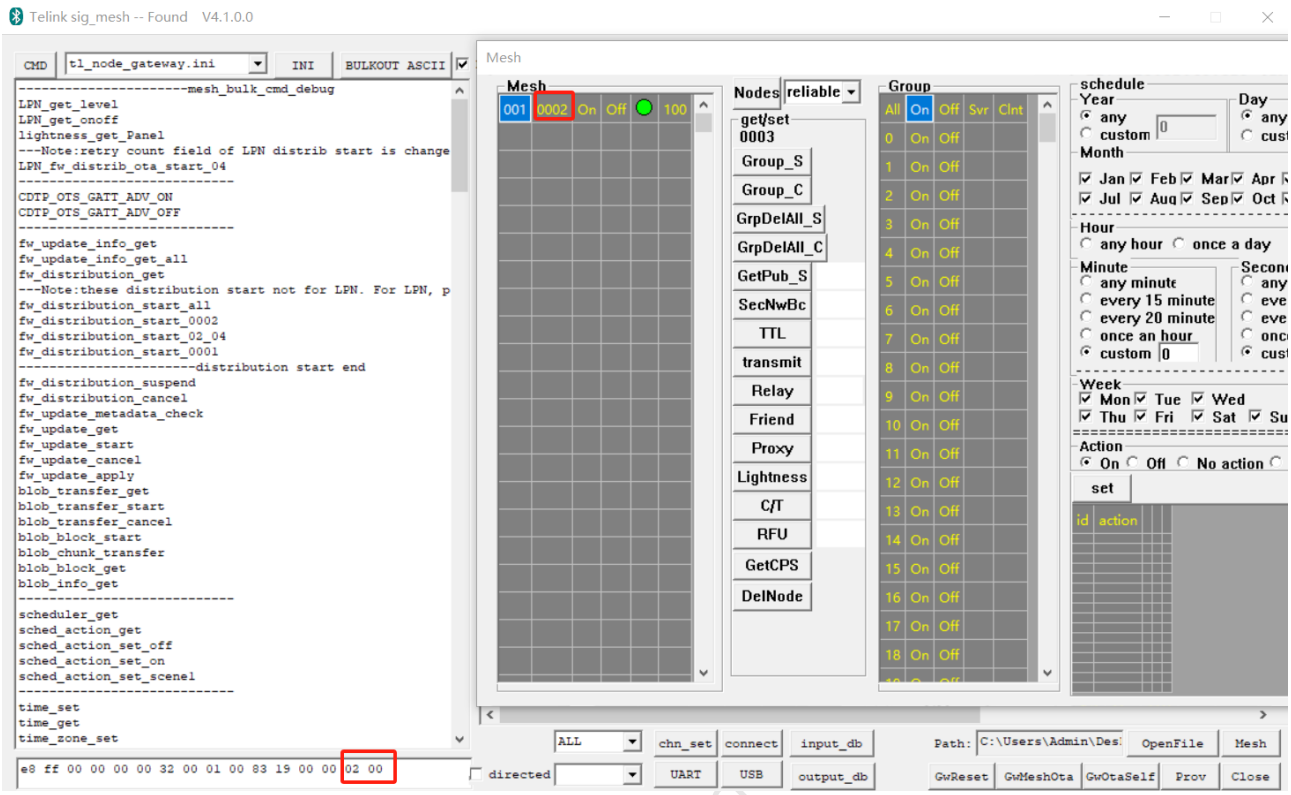


Figure 24.24: LPN mesh ota start

24.5.1.6 OTA Finish

The log after successful ota is shown below.

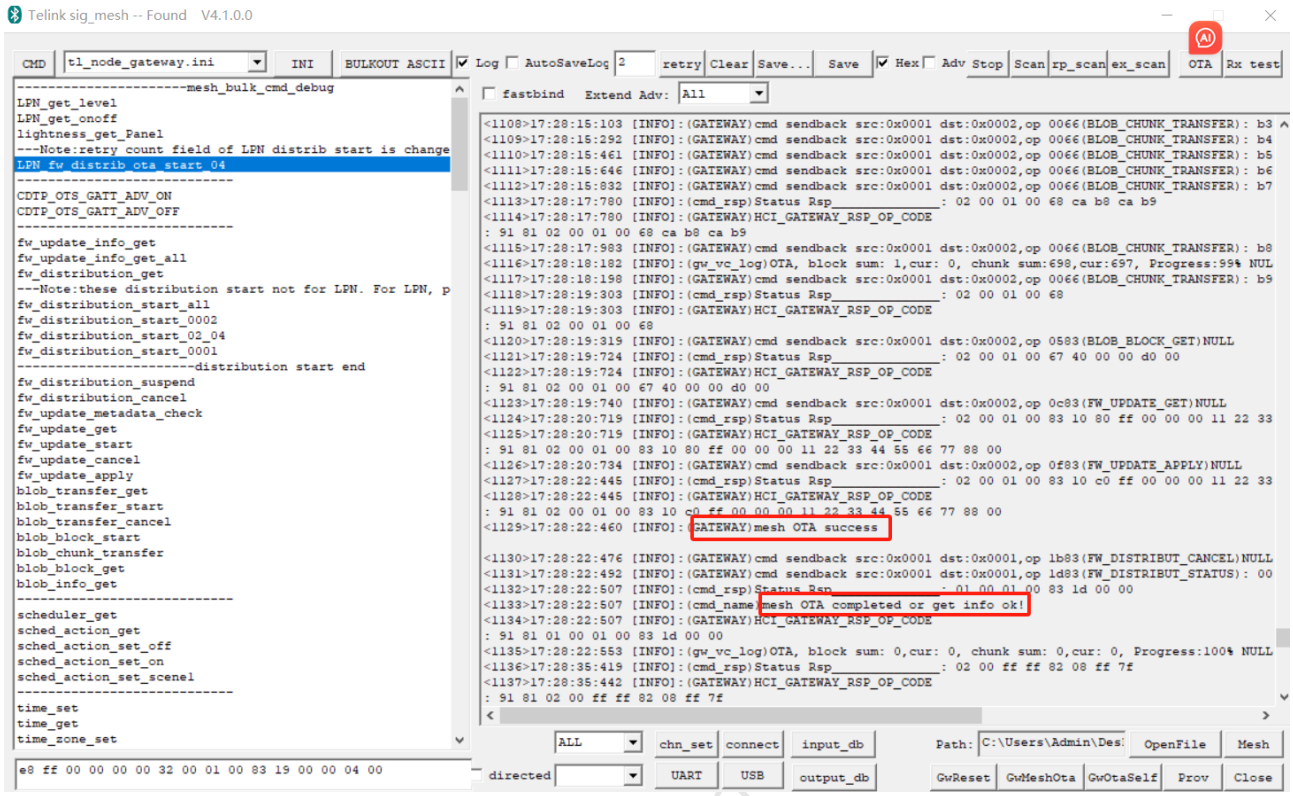


Figure 24.25: ota success

24.5.2 LPN Mesh OTA Gatt Master Dongle Mode

Test condition: 8269 dongle 1 (burning 8269kma_master_dongle), 8258 dongle 3 (two burning 8258_mesh.bin, one burning 8258_mesh_LPN.bin)

24.5.2.1 Code Configuration

(1) Open MD_MESH_OTA_EN.

(2) To shorten the mesh ota time, the macro EXTENDED_ADV_ENABLE can be set to 1 to support the extended broadcast packet mode, which should be noted is not a spec-defined mode.

(3) If the ota upgrade selects the directly connected node as the distributor mode, you need to change DISTRIBUTOR_UPDATE_SERVER_EN from 0 to 1.

24.5.2.2 Networking Nodes

Refer to the gatt master dongle mesh ota Selecting a Networking Node Procedure.

Note:

> If the LPN node is not shown in the UI after the networking is completed, you can click this INI com-

> mand to get it (you need to pay attention to whether the destination address is correct, the default is 0x0004):

CMD-LPN_get_onoff

Another way is that the power down the LPN node and then re-power-up operation, the LPN will actively send the current status once, and the UI interface will display the LPN node.

24.5.2.3 Select New Firmware

Refer to gatt master dongle mesh ota select new firmware process.

24.5.2.4 Get Version

Refer to gatt master dongle mesh ota selection to get the version process.

24.5.2.5 OTA Start

(1) Upper computer as distributor mode

Click on LPN_fw_distrib_ota_start_04,then modify the last two bytes of the command window below to be the unicast address of the LPN node, i.e., 02 00, and then click on the Enter key to send this INI command.

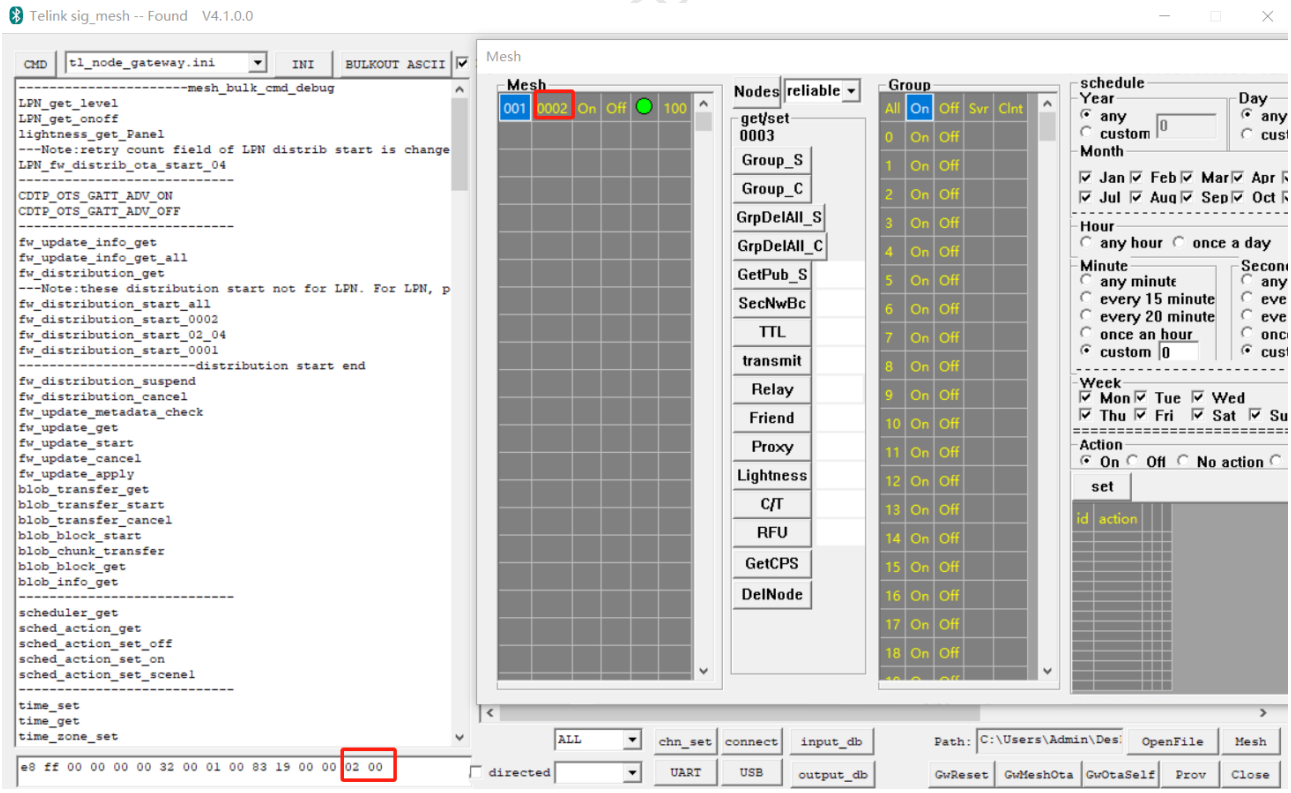


Figure 24.26: LPN mesh ota start

(2) The directly connected node acts as a distributor and the verify apply mode is selected.

Click LPN_initiator_start_v_apply4,then modify the last two bytes of the command window below to be the unicast address of the LPN node, i.e., 02 00, and then hit enter.

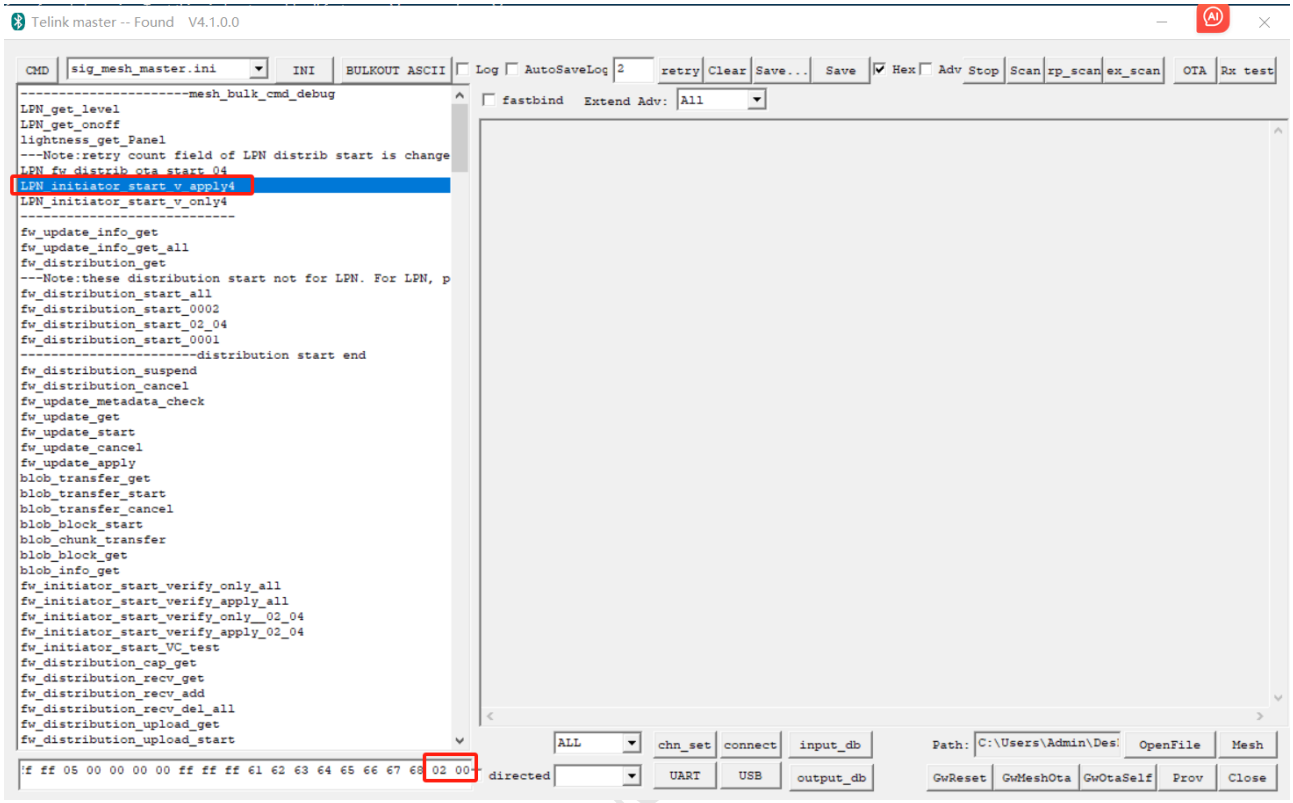


Figure 24.27: LPN mesh ota verify apply mode

(3) Directly connected node as distributor and verify only mode selected.

Click LPN_initiator_start_v_only4,then modify the last two bytes of the command window below to be the unicast address of the LPN node, i.e., 02 00, and then click Enter.

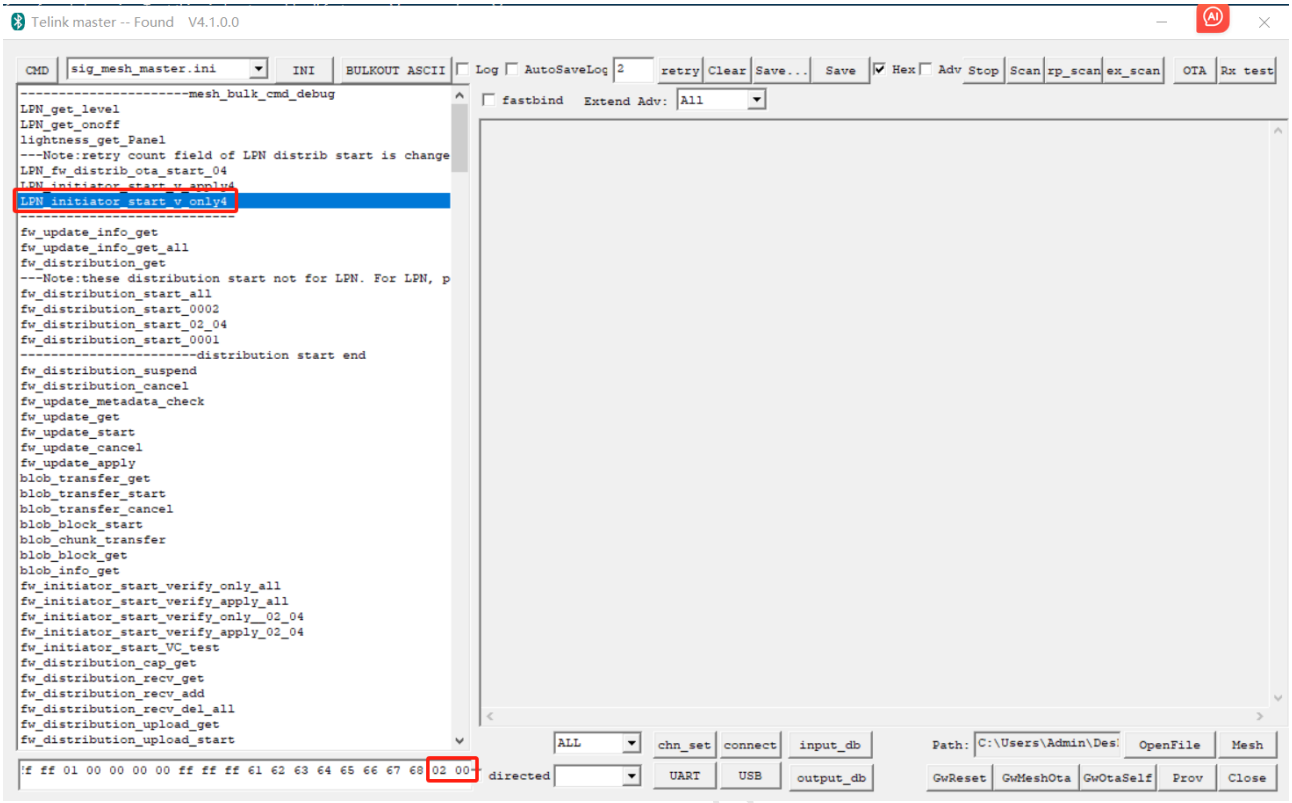


Figure 24.28: LPN mesh ota verify only mode

24.5.2.6 OTA Finish

The log after successful ota is shown below.

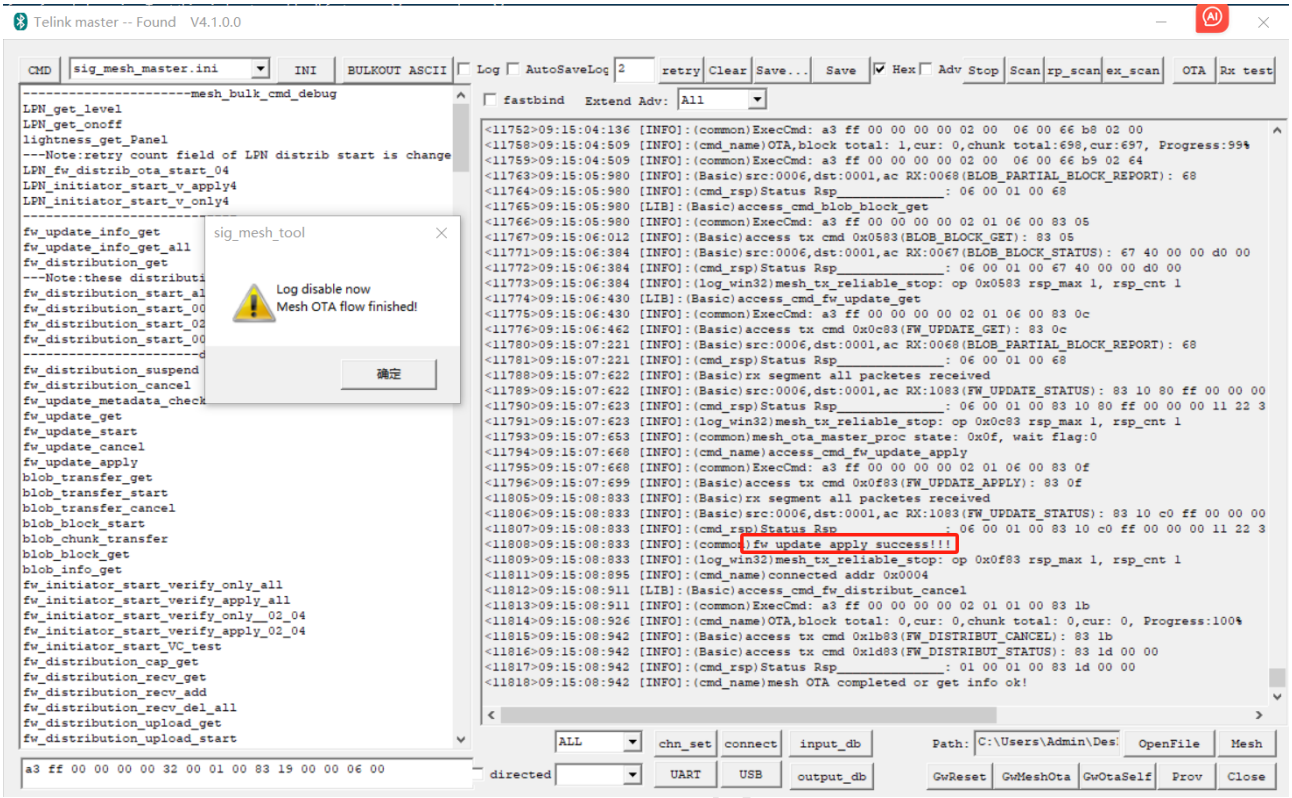


Figure 24.29: ota success

24.6 QA

24.6.1 What's the Best Way to Distinguish Between Different Equipment Types for OTA?

To initiate a mesh OTA on the VC host, you click the distribute start command, which has the parameter format:

```
typedef struct{
    u16 adr_group;
    u16 update_list[MESH_OTA_UPDATE_NODE_MAX];
}fw_distribut_start_t;
```

Figure 24.30: fw_distribution_start

(1) Method 1: Normally, you need to put all the node addresses of all the nodes that need to be upgraded inside the update_list array. Those that are not in the update list will not be OTA.

(2) Method 2: By default, the device side will judge whether the PID(Product ID) of the new firmware and the current PID are the same or not, if not, it will reject the current OTA request. The corresponding function to judge is mesh_ota_slave_need_ota()->ota_is_valid_pid_vid().

So you can modify the update list parameter of the distribution start command to determine which node to initiate OTA for, for example, "fw_distribution_start_02_04" is to initiate OTA for 0x0002 and 0x0004:

CMD-fw_distribution_start_02_04 =e8 ff 00 00 00 00 00 00 01 00 83 19 00 c0 02 00 04 00

Introduction to the fw_distribution_start_all Command:

```
fw_distribution_start_all   =e8 ff 00 00 00 00 00 00 01 00 b6 0b 00 c0
```

Figure 24.31: fw_distribution_start_all

On the VC UI, we don't want to have to manually add the node address to the distribute start command in order to initiate an OTA for ease of operation, but rather have a common command to start executing the OTA, so we have made a special marking, i.e., when the length of the update list is 0, we assume that all nodes displayed in this UI are added to the update list. As shown in the figure below, clicking "fw_distribution_start_all" will perform OTA on nodes 0x0004 and 0x0007.

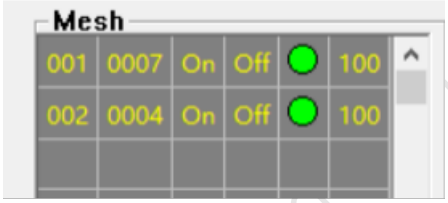


Figure 24.32: start_all_UI

24.6.2 Ways to Differentiate between Different Devices?

We are currently using the PID (Product ID) to determine them.

24.6.3 Is it Possible to Confirm the Version before OTA?

On the originating side (master side), it is identified by reading the PID and CID of the composition data inside the Jason file.

On the node side (the upgraded side), mesh_ota_slave_need_ota() is used to judge the pid cid from the meta data to determine whether to upgrade or not, and returns 0 if no upgrade is needed. By default, our Demo SDK is to let the master decide which to upgrade, and the node side only check PID, if equal, return 1 to allow OTA.

24.6.4 Can I Revert to a Previous Version?

The default is yes, which means that OTA downgrades are allowed. If you wish to disallow downgrades, just set OTA_ADOPT_RULE_CHECK_VID_EN to 1.

24.6.5 What Needs to Be Done in FW in order to Differentiate between Device Types for Separate OTAs?

On the node side (upgraded side), ota_is_valid_pid_vid() is used to judge whether the product ID passed from the meta data is the same as the product ID of the new firmware to judge whether to upgrade or not, if the product ID is not the same, it returns 0 to indicate that it cannot be upgraded; if it is the same, it returns 1 to indicate that it can be upgraded.

24.6.6 What Needs to Be Done in FW in order to Distinguish FW Version Information for OTA?

On the node side (the upgraded side), ota_is_valid_pid_vid() is used to judge the version information passed from the meta data to determine whether to upgrade or not, and returns 0 if the upgrade is not needed, or 1 if the upgrade is needed.

24.7 Appendix Log

Gateway Mesh OTA Upgrade for Nodes with Unicast Addresses 0x0002 and 0x0004

(To save OTA time, the following log is based on enabling the private extended advertising packet mode, i.e., EXTENDED_ADV_ENABLE is set to 1.)

<0106>15:26:57:370 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 01 00 83 19 00 c0

<0107>15:26:57:418 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0001,op 1983(FW_DISTRIBUT_START): 00 c0 02 00 04 00

<0108>15:26:57:433 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0001,op 1d83(FW_DISTRIBUT_STATUS): 00 00

<0109>15:26:57:449 [INFO]:(cmd_rsp)Status Rsp_____: 01 00 01 00 83 1d 00 00

<0110>15:26:57:449 [INFO]:(cmd_name)mesh OTA completed or get info ok!

<0111>15:26:57:449 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 01 00 01 00 83 1d 00 00

<0112>15:26:57:480 [INFO]:(gw_vc_log)OTA, block sum: 0,cur: 0, chunk sum: 0,cur: 0, Progress: 0% NULL

<0113>15:26:57:496 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0a83(FW_UPDATE_METADATA_CHE 00 01 00 32 38 00 00 00 00

<0114>15:26:57:512 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 0b 08 00

<0115>15:26:57:512 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 0b 08 00

<0116>15:26:57:620 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0a83(FW_UPDATE_METADATA_CHE 00 01 00 32 38 00 00 00 00

<0117>15:26:57:666 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 0b 08 00

<0118>15:26:57:666 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 0b 08 00

<0119>15:26:57:823 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 1b80(CFG_MODEL_SUB_ADD): 02 00 00 c0 00 14

<0120>15:26:57:963 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 80 1f 00 02 00 00 c0 00 14

<0121>15:26:57:965 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 80 1f 00 02 00 00 c0 00 14

<0122>15:26:58:043 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 1b80(CFG_MODEL_SUB_ADD): 04 00 00 c0 00 14

<0123>15:26:58:244 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 80 1f 00 04 00 00 c0 00 14

<0124>15:26:58:249 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 80 1f 00 04 00 00 c0 00 14

<0125>15:26:58:260 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0883(FW_UPDATE_INFO_GET): 00 01

<0126>15:26:58:306 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 09 01 00 04 01 00 41 00 00

<0127>15:26:58:306 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 09 01 00 04 01 00 41 00 00

<0128>15:26:58:461 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0883(FW_UPDATE_INFO_GET): 00 01

<0129>15:26:58:538 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 09 01 00 04 01 00 41 00 00

<0130>15:26:58:538 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 09 01 00 04 01 00 41 00 00

<0131>15:26:58:661 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0d83(FW_UPDATE_START): ff 00 00 11 22 33 44 55 66 77 88 00 01 00 32 38 00 00 00 00

<0132>15:26:58:942 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 10 40 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0133>15:26:58:942 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 10 40 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0134>15:26:59:112 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0d83(FW_UPDATE_START): ff 00 00 11 22 33 44 55 66 77 88 00 01 00 32 38 00 00 00 00

<0135>15:26:59:419 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 10 40 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0136>15:26:59:419 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 10 40 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0137>15:26:59:591 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0083(BLOB_TRANSFER_GET)NULL

<0138>15:26:59:885 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 03 00 01 11 22 33 44 55 66 77 88

<0139>15:26:59:885 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 03 00 01 11 22 33 44 55 66 77 88

<0140>15:27:00:054 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0083(BLOB_TRANSFER_GET)NULL

<0141>15:27:00:315 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 03 00 01 11 22 33 44 55 66 77 88

<0142>15:27:00:315 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 03 00 01 11 22 33 44 55 66 77 88

<0143>15:27:00:502 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0683(BLOB_INFO_GET)NULL

<0144>15:27:00:768 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 07 12 12 ed 04 d0 00 00 00 03 00 7c 01 01

<0145>15:27:00:768 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 07 12 12 ed 04 d0 00 00 00 03 00 7c 01 01

<0146>15:27:00:940 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0683(BLOB_INFO_GET)NULL

<0147>15:27:01:203 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 07 12 12 ed 04 d0 00 00 00 03 00 7c 01 01

<0148>15:27:01:203 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 07 12 12 ed 04 d0 00 00 00 03 00 7c 01 01

<0149>15:27:01:403 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0183(BLOB_TRANSFER_START): 40 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01

<0150>15:27:01:683 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 03 40 02 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01 01

<0151>15:27:01:683 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 03 40 02 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01 01

<0152>15:27:01:872 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0183(BLOB_TRANSFER_START): 40 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01

<0153>15:27:02:167 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 03 40 02 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01 01

<0154>15:27:02:167 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 03 40 02 11 22 33 44 55 66 77 88 c4 11 00 00 12 7c 01 01

<0155>15:27:02:352 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0483(BLOB_BLOCK_START): 00 00 d0 00

<0156>15:27:02:398 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 67 00 00 00 d0 00

<0157>15:27:02:398 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 67 00 00 00 d0 00

<0158>15:27:02:552 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0483(BLOB_BLOCK_START): 00 00 d0 00

<0159>15:27:02:598 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 67 00 00 00 d0 00

<0160>15:27:02:598 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 67 00 00 00 d0 00

<0161>15:27:02:785 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 0, Progress: 5%
NULL

<0162>15:27:02:800 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
00 00 26 80 01 00 32 38 5d 01 4b 4e 4c 54 60 00 88 00 ae 80

<0163>15:27:03:000 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 1, Progress: 9%
NULL

<0164>15:27:03:015 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
01 00 08 58 10 50 04 b1 04 b2 f8 87 00 a0 1a 09 1b 0a 91 02

<0165>15:27:03:217 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 2, Progress:14%
NULL

<0166>15:27:03:232 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
02 00 70 07 c0 46 00 65 00 f6 00 fe 07 0b 18 40 07 0b 18 40

<0167>15:27:03:448 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 3, Progress:18%
NULL

<0168>15:27:03:463 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
03 00 10 6d c0 46 43 06 80 00 b8 00 80 00 ba 00 80 00 00 65

<0169>15:27:03:649 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 4, Progress:23%
NULL

<0170>15:27:03:665 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
04 00 f9 c9 01 a2 04 0b 1a 40 34 a0 80 a1 ff 97 78 9f 01 60

<0171>15:27:03:867 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 5, Progress:27%
NULL

<0172>15:27:03:882 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
05 00 02 a1 ff 97 15 9f 02 a0 a2 a1 ff 97 11 9f 27 a0 00 a1

<0173>15:27:04:084 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 6, Progress:32%
NULL

<0174>15:27:04:099 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
06 00 c1 87 01 a3 63 40 d1 87 60 00 80 00 04 04 04 04 40 0c

<0175>15:27:04:284 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 7, Progress:36%
NULL

<0176>15:27:04:314 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 8, Progress:41%
NULL

<0177>15:27:04:330 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
08 00 d1 87 bd a0 ff 97 bc 9d 01 ec a1 03 09 f6 09 fe bd a0

<0178>15:27:04:486 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur: 9, Progress:45%
NULL

<0179>15:27:04:502 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER):
09 00 9c 02 5c c1 0a a9 00 c1 45 82 28 a9 00 c1 29 83 0d a9

<0180>15:27:04:701 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:10, Progress:50%
NULL

<0181>15:27:04:716 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0a 00 54 e9 af 0a a4 e8 22 48 13 00 0b 03 23 40 30 6d 02 ac

<0182>15:27:04:903 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:11, Progress:54% NULL

<0183>15:27:04:919 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0b 00 15 82 81 a5 ad f0 cf a3 00 a1 8b 87 05 a9 00 c1 69 81

<0184>15:27:05:120 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:12, Progress:59% NULL

<0185>15:27:05:135 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0c 00 00 c1 01 82 28 a9 00 c1 5e 81 08 a5 3f a3 00 a1 20 87

<0186>15:27:05:320 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:13, Progress:63% NULL

<0187>15:27:05:336 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0d 00 fc a3 00 a1 be 86 09 a9 00 c1 df 80 04 a9 00 c1 a3 81

<0188>15:27:05:521 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:14, Progress:68% NULL

<0189>15:27:05:537 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0e 00 40 a1 57 86 82 a5 6d f0 cf a3 10 a1 52 86 a7 0d fc a3

<0190>15:27:05:742 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:15, Progress:72% NULL

<0191>15:27:05:757 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 0f 00 00 a1 ef 85 25 ec f3 a3 00 a1 eb 85 74 0d fc a3 00 a1

<0192>15:27:05:946 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:16, Progress:77% NULL

<0193>15:27:05:961 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 10 00 88 85 43 0d fc a3 02 a1 84 85 c4 a5 ad f0 fc a3 02 a1

<0194>15:27:06:149 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:17, Progress:81% NULL

<0195>15:27:06:164 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 11 00 c1 a5 ad f0 cf a3 10 a1 1c 85 25 ec f3 a3 04 a1 18 85

<0196>15:27:06:367 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:18, Progress:86% NULL

<0197>15:27:06:382 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 12 00 cf 99 f2 a0 08 a1 ff 97 cb 99 04 a0 00 a1 ff 97 7f 9b

<0198>15:27:06:586 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:19, Progress:90% NULL

<0199>15:27:06:602 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 13 00 67 99 88 a0 ff 97 44 99 04 ec 87 a0 ff 97 40 99 1f a1

<0200>15:27:06:789 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:20, Progress:95% NULL

<0201>15:27:06:804 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 14 00 00 a3 23 50 3b 40 42 06 13 40 04 6c 90 06 f0 6d 66 00

<0202>15:27:06:991 [INFO]:(gw_vc_log)OTA, block sum: 1,cur: 0, chunk sum:22,cur:21, Progress:99% NULL

<0203>15:27:07:007 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 15 00 11 50 a5 a1 19 0a 11 40 01 b2 13 40 10 6d 18 08 ff 97

<0204>15:27:07:195 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0583(BLOB_BLOCK_GET)NULL

<0205>15:27:07:226 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 67 c0 00 00 d0 00 07

<0206>15:27:07:226 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 67 c0 00 00 d0 00 07

<0207>15:27:07:396 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0583(BLOB_BLOCK_GET)NULL

<0208>15:27:07:475 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 67 c0 00 00 d0 00 07

<0209>15:27:07:475 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 67 c0 00 00 d0 00 07

<0210>15:27:07:614 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0xc000,op 0066(BLOB_CHUNK_TRANSFER): 07 00 10 65 0c f6 24 fe a1 f0 23 f1 19 03 21 03 09 f6 09 fe

<0211>15:27:07:833 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0583(BLOB_BLOCK_GET)NULL

<0212>15:27:07:864 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 67 40 00 00 d0 00

<0213>15:27:07:864 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 67 40 00 00 d0 00

<0214>15:27:08:052 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0583(BLOB_BLOCK_GET)NULL

<0215>15:27:08:209 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 67 40 00 00 d0 00

<0216>15:27:08:209 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 67 40 00 00 d0 00

<0217>15:27:08:271 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0c83(FW_UPDATE_GET)NULL

<0218>15:27:08:582 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 10 80 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0219>15:27:08:582 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 10 80 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0220>15:27:08:754 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0c83(FW_UPDATE_GET)NULL

<0221>15:27:09:035 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 10 80 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0222>15:27:09:035 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 10 80 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0223>15:27:09:219 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0002,op 0f83(FW_UPDATE_APPLY)NULL

<0224>15:27:09:530 [INFO]:(cmd_rsp)Status Rsp_____: 02 00 01 00 83 10 c0 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0225>15:27:09:530 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 02 00 01 00 83 10 c0 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0226>15:27:09:717 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0004,op 0f83(FW_UPDATE_APPLY)NULL

<0227>15:27:09:999 [INFO]:(cmd_rsp)Status Rsp_____: 04 00 01 00 83 10 c0 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0228>15:27:09:999 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE: 91 81 04 00 01 00 83 10 c0 ff 01 00 00 11 22 33 44 55 66 77 88 00

<0229>15:27:10:186 [INFO]:(GATEWAY)mesh OTA success

<0230>15:27:10:201 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0001,op 1b83(FW_DISTRIBUT_CANCEL)NULL

<0231>15:27:10:217 [INFO]:(GATEWAY)cmd sendback src:0x0001 dst:0x0001,op 1d83(FW_DISTRIBUT_STATUS): 00 00

<0232>15:27:10:232 [INFO]:(cmd_rsp)Status Rsp_____: 01 00 01 00 83 1d 00 00

<0233>15:27:10:232 [INFO]:(cmd_name)mesh OTA completed or get info ok!

<0234>15:27:10:232 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE : 91 81 01 00 01 00 83 1d 00 00

<0235>15:27:10:279 [INFO]:(gw_vc_log)OTA, block sum: 0,cur: 0, chunk sum: 0,cur: 0, Progress:100% NULL

25 Subnet Bridge

25.1 Function Introduction

A Bluetooth network contains one or more subnets, and the subnets are usually isolated from each other using different network key encryption. In mesh 1.0, only devices that are on the same subnet and use the same network key can communicate with each other; devices between different subnets cannot communicate with each other. Secure isolation using subnets is a powerful feature: as shown in the figure below, in a hotel where each room is isolated from each other by subnets, devices in one room will not interfere with devices in another room.

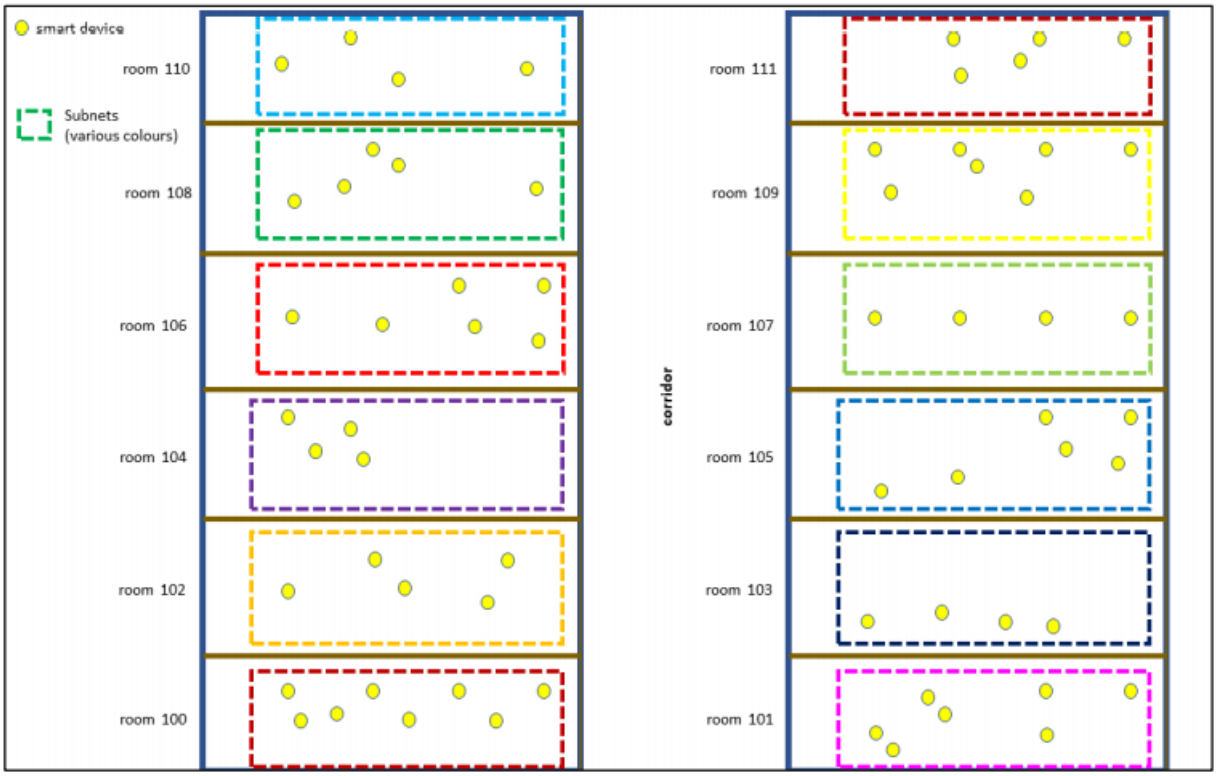


Figure 25.1: Hotel subnet map

In some specific cases, there is a need for messages to be able to be transmitted between subnets. For example, the need for a guest in a hotel room to press the room cleaning service button to request service from the housekeeping team. Since the mesh key of each room cannot be the same (based on permission control considerations, the guest's mobile phone console can only have the key of the current room, not the key of other rooms, including the housekeeping team's key), in mesh 1.0, it is necessary to add a non-mesh gateway (e.g., TCP/IP) for each room to report the message to achieve this, and it is not possible to report the message to the housekeeping team through the current mesh network.

The Subnet bridge feature now enables communication between devices on different subnets, even if they do not share a common subnet and network key. By selecting a mesh node that supports two keys as a bridge point, and by configuring the bridge table of the bridge node, it is possible to achieve that after a guest presses the room clean button, the message can reach the housekeeping team through subnet

bridging, but other devices in the room that do not have a bridge table configured cannot communicate with devices outside of this room.

Field	Size (bits)	Description
Directions	8	Allowed directions for the bridged traffic
NetKeyIndex1	12	NetKey index of the first subnet
NetKeyIndex2	12	NetKey index of the second subnet
Address1	16	Address of the node in the first subnet
Address2	16	Address of the node in the second subnet

Figure 25.2: Bridging tables

In the bridging table shown above,

"Directions" specifies whether bi-directionality is supported.

"NetKeyIndex1, NetKeyIndex2" specifies between which two netkeys the bridge is established.

"Address1, Address2" specifies the two addresses between which the bridge is to be established.

When the network layer receives the message, it will check the bridge table to decide whether to bridge or not. If the bridge conditions are met, the message is re-encrypted and forwarded using another mesh network's NetKey specified in the bridge table.

An overview of the functions can also be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/, this SIG official website description.

25.2 Subnet Bridging Principles

The subnet bridging feature forwards messages to specific subnets at the network layer by configuring a bridging table for nodes with multiple subnets.

When a node receives a message, if the source and destination addresses of the message are in the bridge table and the message is encrypted using the NetKey of the source subnet, it will decrypt the message using the NetKey of the source subnet and then re-encrypt the message with the NetKey of the target subnet before forwarding it to the specified subnet; if the source and destination addresses of the message are not in the bridge table, the message will not be forwarded to any other subnet.

25.3 Configuration

(1) Set the macro MD_SBR_CFG_SERVER_EN to 1 in the mesh_config.h file.

```
911: #define MD_SCHEDULE_EN              MD_TIME_EN  // because both of them save in same flash s
912: #define MD_PROPERTY_EN              0
913: #define MD_LOCATION_EN              0   // location,sensor,battery use same flash addr, but
914: #define MD_BATTERY_EN               0
915:     #if (__PROJECT_MESH_LPN__||__PROJECT_MESH_SWITCH__||__PROJECT_SPIRIT_LPN__)
916: // not support directed forwarding model and subnet bridge model.
917:     #else
918: #define MD_DF_CFG_SERVER_EN         0   // directed forwarding server model.
919: #define MD_DF_CFG_CLIENT_EN         0
920: #define MD_SBR_CFG_SERVER_EN        0   // subnet bridge server model.
921: #define MD_SBR_CFG_CLIENT_EN        0
922:     #endif
923: #define MD_SAR_EN                   0
924: #define MD_ON_DEMAND_PROXY_EN       0
925: #define MD_OP_AGG_EN                0
926: #define MD_LARGE_CPS_EN             0
927: #define MD_SOLI_PDU_RPL_EN          MD_ON_DEMAND_PROXY_EN
928:     #if __PROJECT_MESH_SWITCH__
929: #define MD_SERVER_EN                0   // SIG and vendor models
```

Figure 25.3: Open MD_SBR_CFG_SERVER_EN

(2) Compile the 8258_mesh project.

(3) Burn at least 3 dongle nodes: light-room 1, light-room 2, cleaning unit.

25.4 Function Display

Scene display:

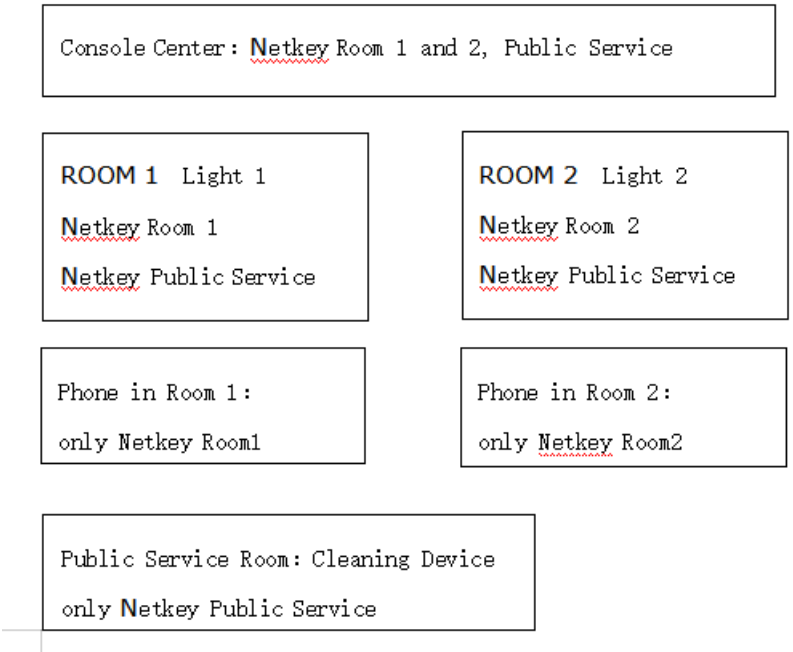


Figure 25.4: Scene display

The Subnet Bridge feature allows a node with multiple subnets to be configured with a bridge table to forward messages to specific subnets.

Please refer to the "Subnet Bridge Setting" section of the chapter Android and iOS APP User Guide for detailed operation instructions.

Note: The sig_mesh_tool on PC does not have a UI to configure the node's subnet bridge parameters at this time.

26 Direct Forwarding

Direct Forwarding reduces the number of packets forwarded over the air by participating in the forwarding of commands at specified path nodes (routing tables). Direct Forwarding focuses on improving network utilisation, not on increasing transmission rates.

Managed flood: It is the propagation of mesh messages from the source outwards, similar to a stone thrown into water producing ripples that spread in all directions. The range of transmission is controlled through ttl. Whether the relay feature of a node is enabled determines whether the node will relay the message. This transmission mode is called managed flooding.

Managed flooding does not control the direction of message delivery and wastes bandwidth on parts of the network that are not related to the message. For example, if there are 2 switches in the middle of a large conference room that control the podium and the lights in the back row, when controlling the lights at the podium, messages are also retransmitted between the back row light nodes.

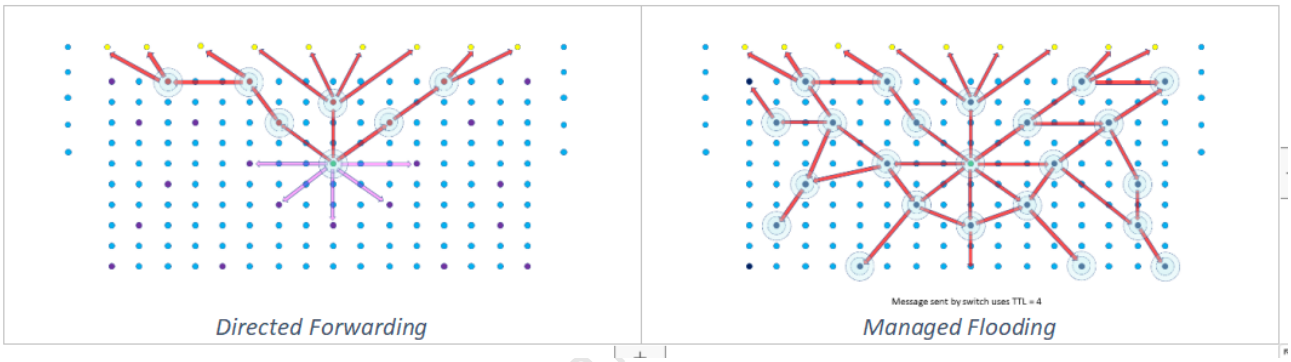


Figure 26.1: Directed Forwarding & Managed Flooding schema

Routing table: A routing refers to a path identifier that specifies all nodes that a message transmission path passes through from the starting point to the endpoint, and only nodes on the path can forward the message. The end point can be unicast, multicast and virtual address. Each routing node will save all path through it. These all paths are called routing table. Select several nodes in the network to form a path, and a routing may have one or more paths.

26.1 Routing Principles

When a message is sent in Direct Forwarding, it will check if the path exists first, and if it does, it will be sent as routed. Otherwise it will be sent as flooding and routing establishment will be triggered automatically. Messages sent by flooding are encrypted with a network key, and messages sent by routing are encrypted with a directed key (derived from the network key).

When a routing node receives a mesh message encrypted by a directed key, it will look up whether there is a corresponding path in the routing table according to the source and destination addresses. If the corresponding path is found, the message is forwarded by routing, otherwise the message will not be relayed. This achieves the purpose of forwarding messages along the specified route.

An overview of the functions can also be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/, this SIG description from the official website.

In DF_TEST_MODE_EN mode, nodes flash lights when forwarding messages encrypted with a DIRECTED key. The "path establishment" message is encrypted with the DIRECTED key, so all nodes flash their lights, indicating that all nodes forwarded the "path establishment" message. After the path is established, only the nodes on the path flash lights, indicating that only the nodes on the path forwarded this message.

26.2 Routing Table Types

Routing is divided into two ways: fixed routing and non-fixed routing:

(1) Fixed routings are configured and managed by provisioner. after the network has been built, routing nodes are selected based on their location in the network. Usually, this requires professional personnel to install and configure.

(2) Non fixed routings are automatically created and maintained by the sender which is the starting point of the path.

26.2.1 Test Firmware Configuration

Open MD_DF_CFG_SERVER_EN and DF_TEST_MODE_EN and compile the 8258_mesh project.

26.2.2 Fixed Routing

The fixed routing is configured and managed by the provisioner to forward with nodes on a specified path. So it is necessary to configure the following in advance on the app:

Go to page of Direct Forwarding–Direct Toggles, then open Direct Forwarding(Main), Direct Relay, Direct Proxy, Direct Friend to the nodes on the path. Note: If you don't enable Direct forwarding(main), the bottom will prompt "(relay) check direct forwarding first".

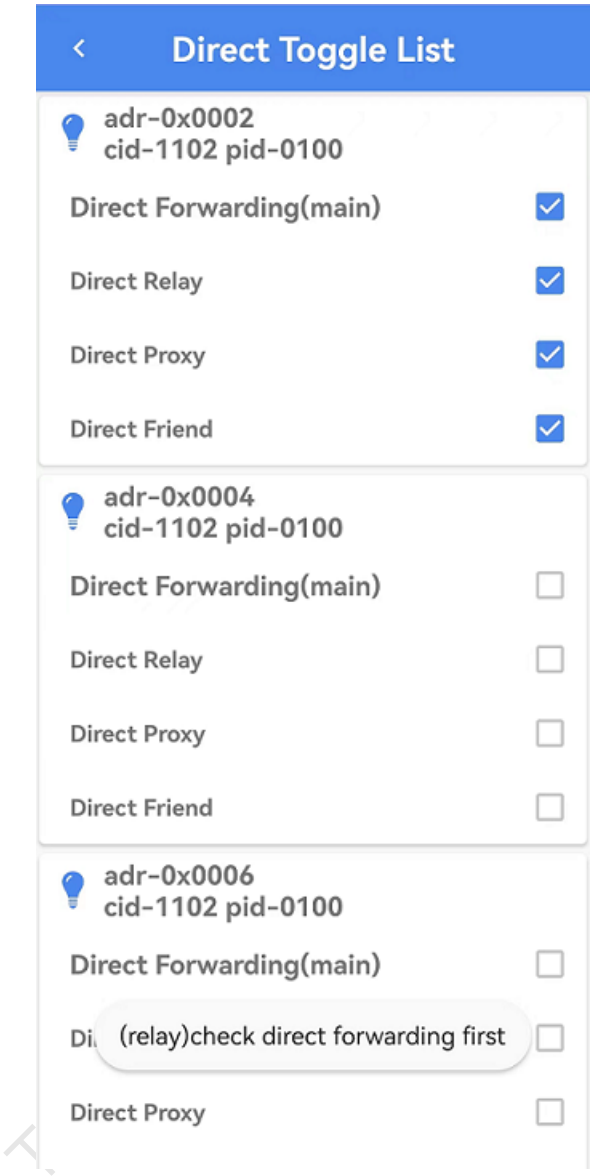


Figure 26.2: Fixed routing directangle toggle list interface

(DF means Direct Forwarding below)

(a) Direct Forwarding(main) - This is the main switch for DF. if disable, all DF features will be disable, include Direct Relay/Proxy/Friend.

(b) Direct Relay - if disable the DF messages will not be relayed to other nodes by current node which received the DF message. DF message is encryption by DF key.

(c) Direct Proxy - if disable, the message send from App can not be sent to other node by rounting, and will be sent by flooding.

(d) Direct Friend - due to LPN(low power node) do not support DF function, because it is a low power node which is not listenning the ADV all the time. and LPN receive message only from the Friend node which has establish friendship with current LPN. so if other node want to send message to a LPN by rounting, we need to enable "Direct Friend" function. then the message can be send to the Friend node by rounting, the Friend node will cache the message and then sent to the LPN when LPN wakeup.

The Direct forwarding interface allows you to add a fixed-routing path to the mesh network by clicking the Add Table button at the bottom. A path contains a start point and an end point, as well as the nodes through which the path passes. When a message is routed from the start point, all nodes on the path participate in forwarding the message; nodes not on the path ignore the message.

Note that commands sent from the phone are generally not used in fixed routing mode, but in Non fixed routing mode, because the mobile app location is not fixed, and the node with which it makes a GATT connection is also not fixed.

So the starting address of a path configuration for a fixed routing is usually a certain lamp node, for example as follows: the start of the path is 0x0006, the node on the path is 0x000e, and the node at the destination address is 0x0016.

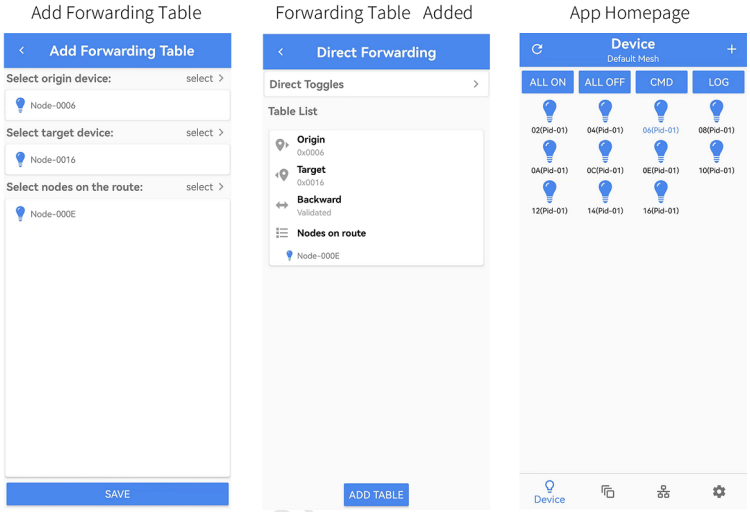


Figure 26.3: Adding a fixed routing

Pressing the SW2 button on node 0x0006 will issue the Generic Onoff command, and since test mode is turned on, i.e., DF_TEST_MODE_EN is turned on, the source address of the command is the node itself (0x0006), and the destination address is the destination address inside the path list of the first fixed routing, i.e., 0x0016.

After the command is issued, we can see that the red LEDs of nodes 0x06, 0x0e and 0x16 on the path are blinking, and the LEDs of other nodes that are not on the path are not blinking, and the nodes that are not blinking indicate that they will not forward the message, which means that the routing function has been realized.

26.2.3 Non-fixed Routing

The Non-fixed routing do not require Direct Forwarding in the APP.

When the command initiator is the mobile app, the routing table will be created automatically and maintained by the node which is GATT connected with the mobile app. the GATT connected node proxy messages in a controlled flooding manner first, and then triggers routing establishment.

When the command initiator is not a mobile app, such as a time gateway, etc., the command sender (path origin) creates and maintains the routing information.

The path establishment message is network key encrypted, so all nodes flash their lights to indicate that the node forwarded the path establishment message. After the path is established, Directed Key was used for encryption, so only the nodes on the path will flash their lights, indicating that only the nodes on the path forwarded the command.

Non-fixed routing establishment rules: The figure below shows the schematic diagram of the two paths from PO (Path Origon) to PT (Path Target), where the shortest path which RSSI is greater than RSSI thread-hold are selected.

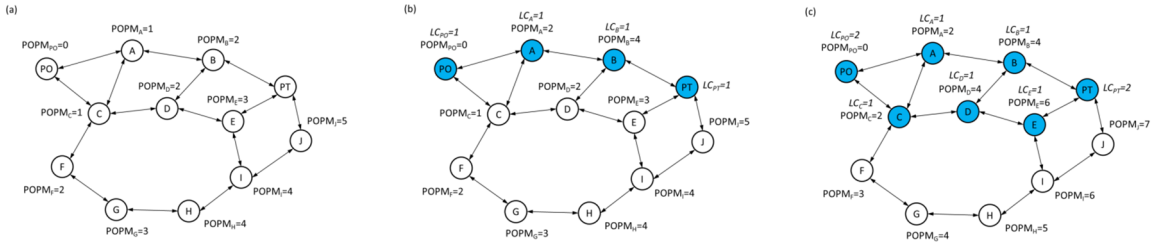


Figure 26.4: Non-fixed routing establishment rules

The test example uses the App to send commands as follows:

(1) Add the node to the network using the app.

(2) The mobile app is in the home page and automatically connects to a node in an initial state where no dynamic paths have been created yet.

(3) The mobile app sends the Generic Onoff command to any non-directly connected node, for example, node 0x0016, which is in flood mode, so all nodes forward the command, that is, all nodes blink the red LED. At this time, it is detected that there is no path for the source and destination addresses, and the path establishment is triggered automatically.

(4) After 5 seconds, the establishment of the path is completed. The established paths can be set to more than one, and in the test mode, only the optimal one path is selected.

(5) The mobile APP sends a light on/off command again to the same destination address 0x0016, at which time only the nodes that are on the same path (including the start and end of the path) will blink at a frequency of 2Hz for 2 seconds.

(6) The mobile app sends the Generic Onoff command to the node again, and only the node for the path from the previous step is blinking. If no messages are sent for a period of time (in test mode it is 12 minutes by default, in non-test mode it is 24 hours by default), the path is deleted. Sending the onoff light again will go to step 3 to re-establish the path.

27 Private-beacon

An overview of the functions can be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/, This SIG official website description.

Please refer to "4.2.44 Mesh Private Beacon", "4.4.11 Mesh Private Beacon Server model" and "4.4.12 Mesh Private Beacon Client model" in "MshPRT_v1.1.pdf" for the corresponding chapters of the spec. " can also be retrieved by typing private in the bookmark bar of the spec.

27.1 Application Background

In some scenarios, such as wearable and other devices that need to be mobile, if the mesh beacons sent out by such devices have plaintext static data, then these messages could be tracked and the location of that device could be tracked. So private-beacon is defined to solve such problems because in private-beacon mode, these beacon data will always change and be encrypted so that they cannot be tracked.

27.2 Function Introductions

The private function ensures that static information in beacon messages is not visible to devices outside the network because it has been encrypted with a network key, increasing security and privacy.

27.2.1 Mesh Private Beacon

As shown in the following figure, Mesh Private beacon is added to the beacon, the decrypted content of this beacon is the same as that of the secure network beacon, and the function is also the same.

So the mesh private beacon is also sent after successful networking, unlike the secure network beacon, the private beacon's ivi index and flag appear encrypted, and the address of the adv is non-reslovable.

The unprovisioned device beacon does not have a corresponding private mode because the node is not yet networked and is not involved in being tracked.

Value	Definition
0x00	Unprovisioned Device beacon
0x01	Secure Network beacon
0x02	Mesh Private beacon
0x03–0xFF	Reserved for Future Use

Figure 27.1: Value of different beacon packages

The following figure describes each field of the private beacon, as well as the calculation process. The SDK corresponding function is mesh_tx_sec_privacy_beacon():

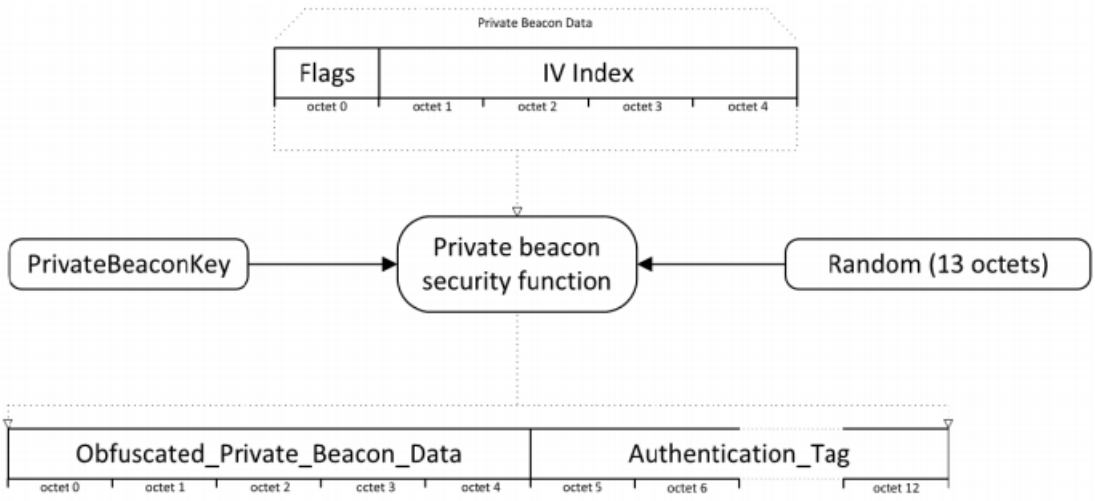


Figure 27.2: Segment of private_ivi

27.2.2 Private Network Identity and Private Node Identity

As shown in the following figure, two new types of connectable broadcast packets have been added，which are Private network identity and Private Node Identity.

After decrypted, Private network identity is equivalent to Network ID and Private Node Identity is equivalent to Node Identity.

Type Value	Description
0x00	Network ID type
0x01	Node Identity type
0x02	Private Network Identity type
0x03	Private Node Identity type
0x04–0xFF	Reserved for Future Use

Figure 27.3: Identification type values

The following figure describes whether the device should send Node Identity or Private Node Identity when the node sends Node Identity state in the following combinations, where the first column "Node Identity state" and the second column "Private Node Identity state" are the conditions, and the third column "Advertising" is the packets that need to be sent.

Node Identity state	Private Node Identity state	Advertising
0x00	Does Not Exist	No Identity Advertising
0x00	Disable (0x00)	No Identity Advertising
0x00	Enable (0x01)	Private Node Identity
0x01	Does Not Exist or Disable (0x00)	Node Identity
0x02	Does Not Exist or Not Supported (0x02)	No Identity Advertising

Figure 27.4: Type value of private

The following figure describes whether the device should send Network ID or Private network identity when the node is in the transitive state of sending Network ID in the following combinations, where the first column of "Node Identity state" and the second column of "Private Node Identity state" are the conditions, and the third column of "Advertising" is the packets that need to be sent.

GATT Proxy state	Private GATT Proxy state	Advertising
0x00	Does Not Exist	No Proxy Advertising
0x00	Disable (0x00)	No Proxy Advertising
0x00	Enable (0x01)	Private Network Identity
0x01	Does Not Exist or Disable (0x00)	Network ID
0x02	Does Not Exist or Not Supported (0x02)	No Proxy Advertising

Figure 27.5: Type value of private

Private-beacon maintains state through two models: private-beacon server model, private-beacon client model.

In the SDK, the corresponding judgement function is mesh_get_identity_type().

27.2.3 Introduction to Opcode

Please refer to "4.3.12 Mesh Private Beacon Messages" in "MshPRT_v1.1.pdf" for the description of the corresponding section of the spec.

PRIVATE_BEACON_SET: Enable or disable the sending of private-beacon.

PRIVATE_GATT_PROXY_SET: Enable or disable the sending of private gatt proxy, i.e., control the private node identity and the private network identity.

PRIVATE_NODE_IDENTITY: Enable or disable the sending of private node identity.

Element	Model ID	State	Message	Rx	Tx
Mesh Private Beacon Main (Primary)	0xTBD	Mesh Private Beacon	PRIVATE_BEACON_GET	M	–
			PRIVATE_BEACON_SET	M	–
			PRIVATE_BEACON_STATUS	–	M
		Private GATT Proxy	PRIVATE_GATT_PROXY_GET	M	–
			PRIVATE_GATT_PROXY_SET	M	–
			PRIVATE_GATT_PROXY_STATUS	–	M
		Private Node Identity	PRIVATE_NODE_IDENTITY_GET	M	–
			PRIVATE_NODE_IDENTITY_SET	M	–
			PRIVATE_NODE_IDENTITY_STATUS	–	M

Figure 27.6: Private beacon opcode

27.3 Test Steps

The firmware SDK turns on MD_PRIVACY_BEA and PRIVATE_PROXY_FUN_EN.

For the test procedure using the App, please refer to section "33.4.5 Private beacon" of the the chapter Android and iOS APP User Guide.

28 Minor Mesh Enhancements

An overview of the functions can also be found in https://www.bluetooth.com/mesh-feature-enhancements-summary/, this SIG description from the official website。

28.1 Opcodes Aggregator Server Model

28.1.1 Application Background

For example, when networking, many key bind commands can be aggregated and packaged into one command to save networking time.

28.1.2 Function

The opcode aggregator allows different opcode messages under the same server model to be packaged into a single OPCODES_AGGREGATOR_SEQUENCE message type to be sent via the LTV structure (length,Opcode,Parameters).

The receiver node get all the opcode and parameters of the AGGREGATOR message by parsing the LTV structure, and packs all the response status into an OPCODES_AGGREGATOR_STATUS through the LTV structure to responde.

An opcode aggregator reduces interaction, processing and response time by compressing a series of messages into one.

Please refer to "4.4.19 Opcodes Aggregator Server model" and "4.4.20 Opcodes Aggregator Client model" in "MshPRT_v1.1.pdf" for the corresponding chapters of the spec.

28.1.3 Test Steps

- Firmware SDK Turn on MD_OP_AGG_EN.
- Use the sig mesh app to network nodes.

When in the app bind process, the app compresses all the bind messages into a single message, reducing the interaction, processing and response time. This is shown in the following figure:

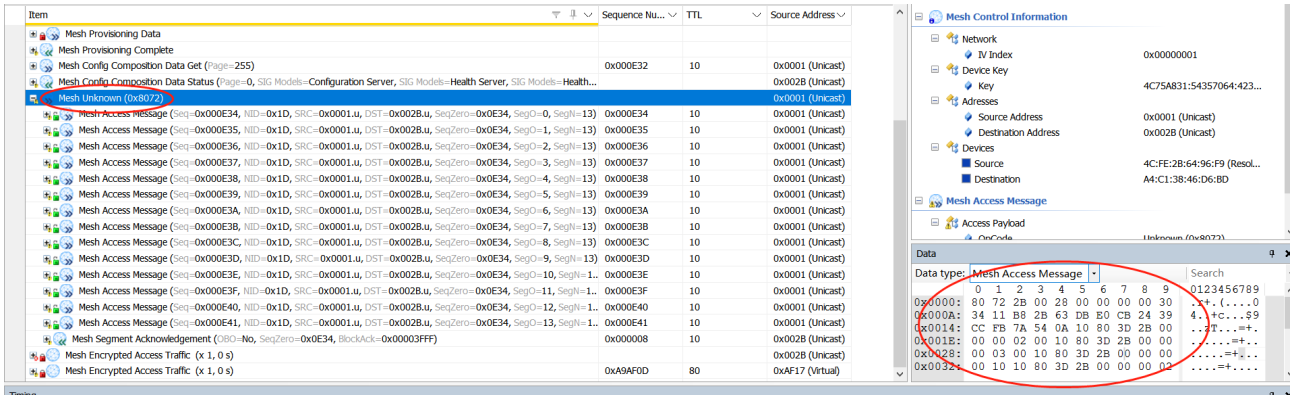


Figure 28.1: Bind compression

28.2 Large Composition Data Models

28.2.1 Application Background

Some devices require a large amount of variable data to describe their composition, configuration data and other attributes. When the node has a lot of elements, for example, 100, then the length of the composition data will be very long, more than 380byte (a mesh message can only send a maximum of 380byte), then it is not possible to send the composition data status by a single message, then you need to segment to get the composition data status.

28.2.2 Function

Composition data consists of a series of pages, each of which is a composition state, where page 0 defines the elements of the node composition as well as the supported models. LARGE_COMPOSITION_DATA_GET can be read starting from a byte of the specified page, as well as specifying the read length for segmented reads.

Please refer to "4.4.21 Large Composition Data Server model" and "4.4.22 Large Composition Data Client model" in "MshPRT_v1.1.pdf" for the corresponding chapters of the spec.

28.2.3 Test Steps

- Firmware SDK Open MD_LARGE_CPS_EN.
- Send LARGE_COMPOSITION_DATA_GET for testing via the INI command.

28.3 SAR Configuration Models

28.3.1 Application Background

When devices from different vendors use different default values for packet grouping behaviour for segment packets, such as retry interval, retry count, timeout time, etc., this may lead to inefficient mesh

message transmission within the mesh network. Unified configuration of SAR behaviours through the SAR Configuration Server model improves performance.

28.3.2 Function Description

SAR means: Segmentation And Reassembly.

Packetisation and reorganisation parameters can be configured through SAR Configuration Server model related commands and can help to improve the efficiency of segment sending and receiving, especially if there are devices from multiple manufacturers in the network.

The SAR Transmitter contains parameters for sending sub-packets: the time interval between sub-packets, and parameters related to the retransmission interval and number of times.

The SAR Receiver contains parameters for receiving the sub-packet: parameters of responding seg ack and timeout.

Please refer to "4.4.15 SAR Configuration Server model" and "4.4.16 SAR Configuration Client model" in "MshPRT_v1.1.pdf" for the description of the corresponding sections of the spec.

28.3.3 Test Steps

- Firmware SDK Open MD_SAR_EN.
- Send SAR-related commands for testing via the INI command.

28.4 EPA(Enhanced Provisioning Authentication)

28.4.1 Application Background

Adding another algorithm further enhances the security of data authentication when networking, especially when networking in static OOB mode.

- Each time a network is configured, the provisioner and provisionee will ask to regenerate the public key and private key for the network.
- The length of oob (including static oob / output oob / input oob) is changed from 16byte to 32byte.
- Use sha256 algorithm.

28.4.2 Function Description

The old algorithm name is BTM_ECDH_P256_CMAC_AES128_AES_CCM, and the new algorithm name is BTM_ECDH_P256_HMAC_SHA256_AES_CCM.

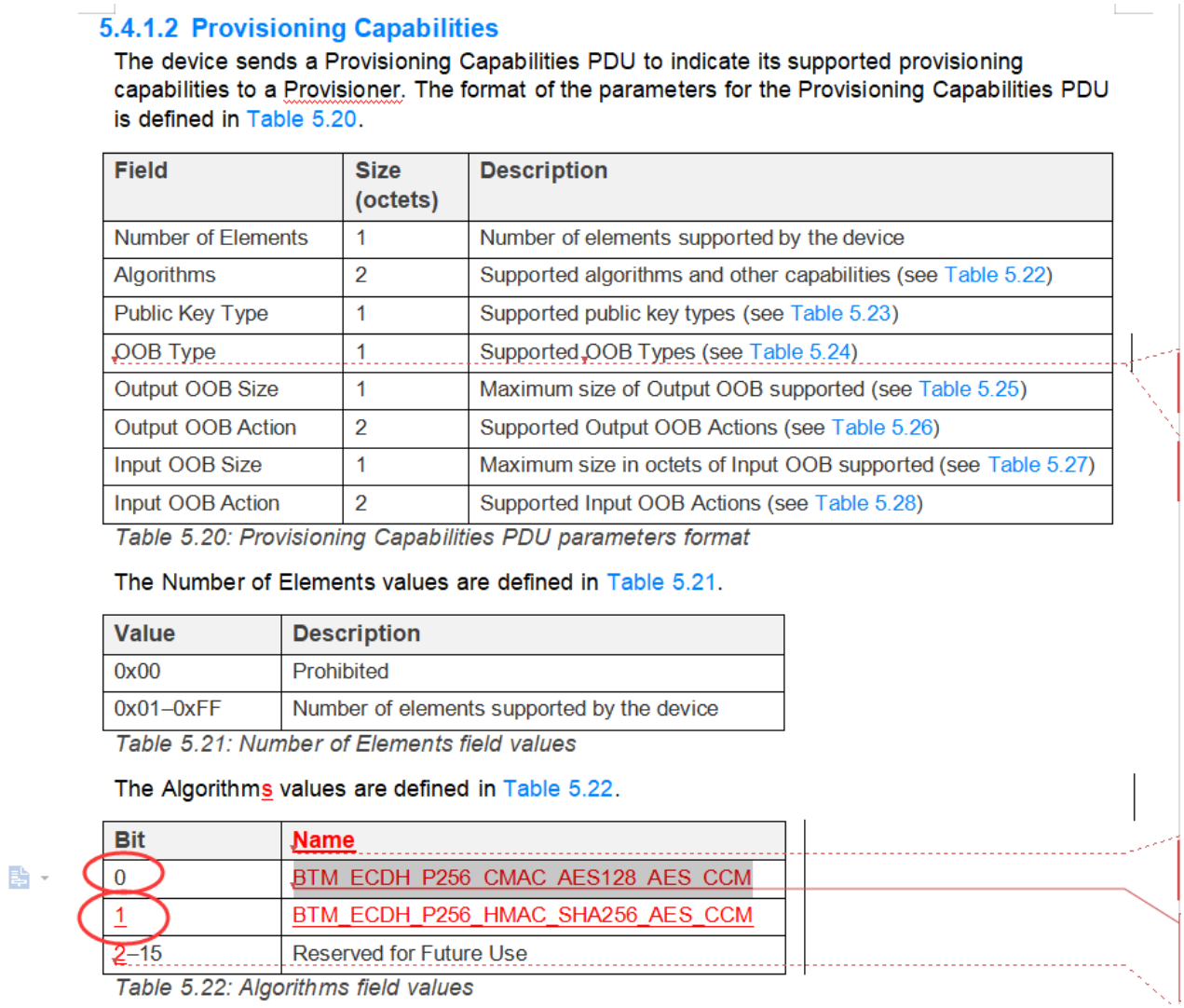
The main difference with spec V1.0 is:

- Each time a network is formed, the provioner and provisionee ask to regenerate the public key and the private key for the network to ensure that retransmitted messages is invalid.

- New Algorithm:EPA(BTM_ECDH_P256_HMAC_SHA256_AES_CCM,) uses the longer sha256 algorithm (the original algorithm was AES128), making it impossible to use a mainframe computer to perform traversal operations, etc.
- The length of oob (including static oob / output oob / input oob) in the new algorithm has been changed from 16byte to 32byte.
- The length of random and confirm in the new algorithm have been changed from 16 bytes to 32 bytes, making it impossible to use a mainframe computer for traversal operations.

Data format:

EPA: Provisioner determines whether a node supports EPA based on the parameter of capability reported by the node, as shown in the Algorithm field in the figure below. In order to be compatible with all providers, the node can choose to set bit0 and bit1 simultaneously to indicate that it supports both the old algorithm and the new algorithm, and then the provider will choose which way to enter the network.



5.4.1.2 Provisioning Capabilities

The device sends a Provisioning Capabilities PDU to indicate its supported provisioning capabilities to a Provisioner. The format of the parameters for the Provisioning Capabilities PDU is defined in Table 5.20.

Field	Size (octets)	Description
Number of Elements	1	Number of elements supported by the device
Algorithms	2	Supported algorithms and other capabilities (see Table 5.22)
Public Key Type	1	Supported public key types (see Table 5.23)
OOB Type	1	Supported OOB Types (see Table 5.24)
Output OOB Size	1	Maximum size of Output OOB supported (see Table 5.25)
Output OOB Action	2	Supported Output OOB Actions (see Table 5.26)
Input OOB Size	1	Maximum size in octets of Input OOB supported (see Table 5.27)
Input OOB Action	2	Supported Input OOB Actions (see Table 5.28)

Table 5.20: Provisioning Capabilities PDU parameters format

The Number of Elements values are defined in Table 5.21.

Value	Description
0x00	Prohibited
0x01–0xFF	Number of elements supported by the device

Table 5.21: Number of Elements field values

The Algorithms values are defined in Table 5.22.

Bit	Name
0	BTM_ECDH_P256_CMAC_AES128_AES_CCM
1	BTM_ECDH_P256_HMAC_SHA256_AES_CCM
2–15	Reserved for Future Use

Table 5.22: Algorithms field values

Figure 28.2: epa description

Please refer to "5.4.1.2 Provisioning Capabilities" in "MshPRT_v1.1.pdf" for the description of the corresponding sections of the spec.

28.4.3 Test Steps

- The firmware SDK turns on PROV_EPA_EN by default.

28.5 On-Demand Proxy Model

28.5.1 Application Background

From a user experience point of view, enabling proxy function on all nodes is the preferred mode, i.e., all nodes are sending connectable broadcast packets, and the mobile APP can connect to the device at any time to control the network. However, when there are a lot of nodes, there will be a lot of connectable broadcast packets in the air, which will affect the available bandwidth, make it harder for the APP and the nodes to make a GATT connection, as well as increase the number of collisions between the subsequent mesh ADV messages and the connectable broadcast packets, which will have a negative impact on the performance of the mesh network. This is where the On-Demand Proxy Model can be used for optimisation.

28.5.2 Function Description

When the On-Demand Proxy Model feature is enabled, the node is in the state of not sending connectable broadcast packets, and then the proxy client (mobile app) sends Solicitation PDUs to the node requesting the node to start broadcasting private beacons for a total of how long it will send private beacons, which is determined by g_ mesh_model_misc_save.on_demand_proxy = ON_DEMAND_PRIVATE_GATT_PROXY_S (default is 30 seconds) to determine. This value can also be set by the APP or gateway via the On-Demand Proxy Model related commands.

After the device side receives the Solicitation PDU, it starts to send the private beacon of duration g_mesh_model_misc_save.on_demand_proxy (it is 30 seconds by default) if all the following 3 conditions are met as follows:

(1) The node's current proxy and private proxy features are both set to support, but are in the disable state.

(2) On-Demand Private GATT Proxy value in 0x01~0xff (set by ON_DEMAND_PRIVATE_PROXY_SET and eventually stored in g_mesh_model_misc_save.on_demand_proxy)

(3) The Solicitation PDU was successfully decrypted using the network key.

Field	Size (octets)	Description
Opcode	2	The message opcode
On-Demand_Private_GATT_Proxy	1	New On-Demand Private GATT Proxy state

Table 4.249: ON_DEMAND_PRIVATE_PROXY_SET message structure

Figure 28.3: On-Demand Proxy descriptions

The corresponding chapters are described in "4.4.13 On-Demand Private Proxy Server Model" and "4.4.14 On-Demand Private Proxy Client Model" of "MshPRT_v1.1.pdf".

Solicitation PDUs have a new sequence number mechanism to prevent replay.

28.5.3 Test Steps

28.5.3.1 Testing with APP

The firmware SDK requires MD_ON_DEMAND_PROXY_EN, PRIVATE_PROXY_FUN_EN, and MD_PRIVACY_BEA to be turned on. for the sake of demonstration, the test was conducted with only one device on the current network.

- After the above functions are configured, the GATT proxy of the node is turned on by the SDK by default, so it does not satisfy the three conditions mentioned in the "Introduction to Functions" subsection of this chapter. Therefore, the node is always in the state of sending connectable broadcast packets. So the mesh app can connect to the device and configure it.
- Go to the app homepage, long press on the node that needs to be configured, then go to settings->private beacon and follow the settings as below to turn off config GATT Proxy and private GATT Proxy. As shown in the following figure:

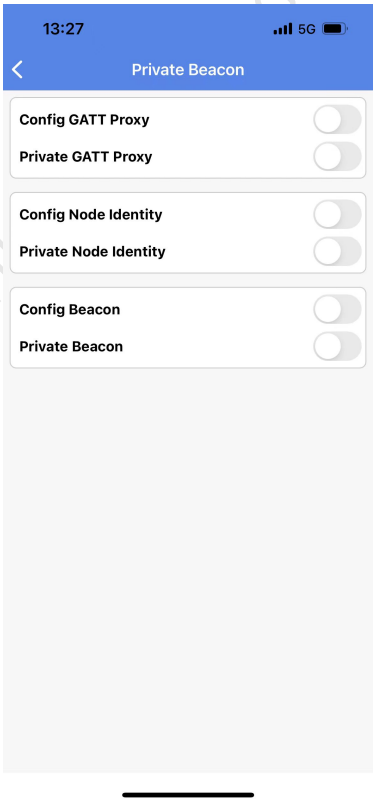


Figure 28.4: set private beacon

Note: If the customer wants to modify the SDK to have GATT proxy turned off for the default node, then change the mesh_global_var_init() inside the

model_sig_cfg_s.gatt_proxy = FEATURE_PROXY_EN ? GATT_PROXY_SUPPORT_ENABLE : GATT_PROXY_NOT_SUPPORT

Change to

model_sig_cfg_s.gatt_proxy = FEATURE_PROXY_EN ? GATT_PROXY_SUPPORT_DISABLE : GATT_PROXY_NOT_SUPPORT

- After going through the above settings, exit the current mesh App and disconnect the app from the node. Scan with a common Bluetooth device scanning App (e.g. Light blue), you will see that after 30 seconds (ON_DEMAND_PRIVATE_GATT_PROXY_S), the node no longer sends broadcast packets, at this time, open the mesh App to reconnect to the node, you can see that the app can't be connected to the node again, as shown in the figure below:

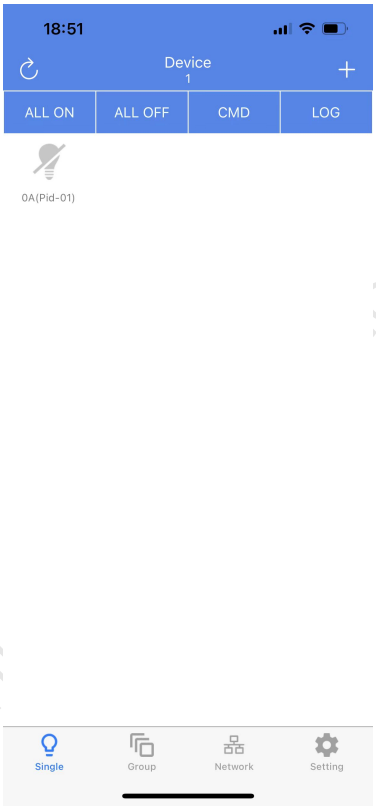


Figure 28.5: Disconnection effects

The main purpose of keeping the connectable broadcast packets sent for a set period of time after disconnection is to consider, for example, the convenience of reconnecting the App after an accidental disconnection.

- If you need to get the node to resend broadcast packets, you can click the Home -> Newtork -> solicitation PDU button as shown in the following figure to get the node to start sending broadcast packets. the destination address of the solicitation PDU is ADR_ALL_PROXY.

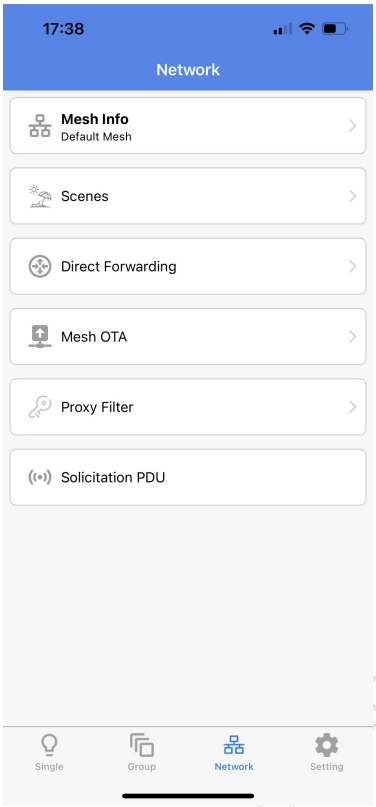


Figure 28.6: Send solicitation PDU

After the app triggers to send solicitation, the app will continuously send solicitation pdu for 10s, which is customised by the app. The solicitation pdu is sent by sending broadcast packet from mobile phone, no need to perform GATT connection, the interval of sending packet is about 100ms. For details of solicitation pdu, please refer to this section solicitation-pdu-rpl-cfg-models

When the node receives the solicitation pdu, it will start sending connectable broadcast packets for the value set to g_mesh_model_misc_save.on_demand_proxy earlier.

During this time you can see that the node is already sending ADV by using a packet grabber or Bluetooth scanner. Then, our APP can automatically initiate a connection to the node and take control of the mesh network.

- The result after reconnecting with the node is shown below:

Figure 28.7: Send solicitation PDU results

If you need to modify the duration that a node sends connectable broadcasts via a command, you can follow the steps below:

- Long-press the node to be configured to enter the interface under settings->device config, click on set under on demand private proxy, and then enter a time value, which is the aforementioned g_mesh_model_misc_save.on_demand_proxy, and this value determines how long it takes for a node to stop sending connectable broadcast packets. The following figure shows this.

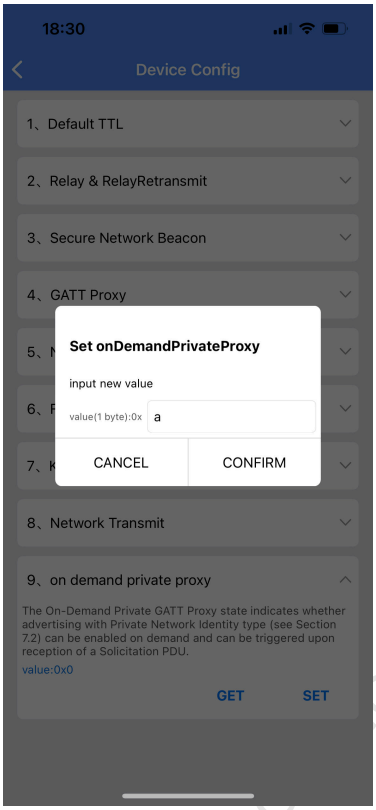


Figure 28.8: set on demand private

- Afterwards, you can click "get" button to get the value you set before, as follows:

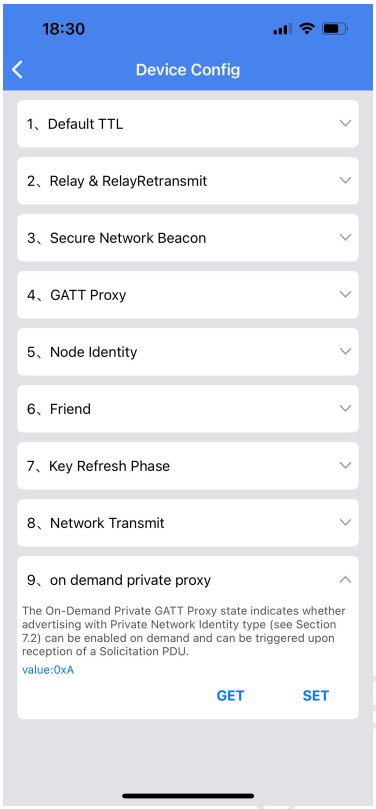


Figure 28.9: get on demand private value

28.6 Solicitation PDU RPL CFG Models

The mobile phone sending a Solicitation command to the node can request the node to start sending private beacons. This Solicitation command is sent in ADV format, and it is not the same format as ordinary network message such as ONOFF. The main difference is that when encrypting and decrypting the solicitation PDU, the iv index is fixed to be 0. The reason is: after sharing the network, APP doesn't know the iv index, and the node doesn't send the private beacon, so there is no way to know the real iv index of the current network, and there is no way to ask the node to start sending connectable ADV through mesh network message, so there is no way to connect and control the nodes through proxy function.

In addition, the solicitation command has an independent sequence number, which is different from the sequence number of ordinary network PDUs, and there is a need to clear the solicitation sequence number cache. Therefore, spec defines Solicitation PDU RPL cfg models to clear the RPL (Relay Protect List) of solicitation PDUs cached on the device.

Field	Size (octets)	Description
Opcode	2	The message opcode
Address_Range	2 or 3	Unicast address range

Table 4.251: SOLICITATION_PDU_RPL_ITEMS_CLEAR message structure

Figure 28.10: PDU_RPL configuration mods

Solicitation PDU RPL cfg models are for On-Demand Proxy Models.

29 Networked Lighting Control(NLC)

A functional overview can be found in the description of the official SIG website https://www.bluetooth.com/mesh-feature-enhancements-summary/ in the section "3. Bluetooth® Mesh Device Profiles".

29.1 Application Background

Bluetooth® Mesh technology provides a rich set of features and options to implement many lighting and sensing applications. This has helped Bluetooth Mesh establish itself as the preferred technology for scalable commercial and industrial applications. However, the optional nature of Bluetooth Mesh features can cause challenges for implementers when they must decide which options to choose for their chosen product segments. If vendors operating in the same product segments choose a different set of options that do not work well with other peer products (e.g., mesh features chosen for light bulbs are not compatible with features selected for light switches), a situation can arise where product ecosystems do not interoperate, which degrades the user experience.

To address this issue, the Bluetooth SIG has come with the concept of Bluetooth Mesh Device Profiles. These profiles are new class of mesh specifications. Device Profiles define which options and features of the mesh specifications are mandatory for a certain kind of end product. The first suite of mesh device profiles is based on the lighting system architecture described in Building a Sensor-Driven Lighting Control System Based on Bluetooth Mesh whitepaper published in 2020. Collectively these profiles are called as Bluetooth Networked Lighting Control (NLC) Profiles, and they are defined as follows:

29.2 All NLC Profiles

29.2.1 NLC Profiles list

- (ALSNLCP) Ambient Light Sensor NLC Profile
- (BLCNLCP) Basic Lightness Controller NLC Profile
- (BSSNLCP) Basic Scene Selector NLC Profile
- (DICNLCP) Dimming Control NLC Profile
- (OCSNLCP) Occupancy Sensor NLC Profile
- (ENMNLCP) Energy Monitor NLC Profile

29.2.2 User Experience when Lights and Sensors work together

The NLC includes the processing linkage of sensor and light, and the main user experience of this linkage is summarized as follows:

- The user experience with ambient light sensor: when there is no ambient light sensor, the light is in the ON state, and the LED will output a maximum brightness of 65535. When there is a ambient light sensor and the light is in the ON state, the LED will output a relatively low brightness based on the current ambient light brightness through the PID algorithm, instead of the maximum brightness of 65536. The purpose is to save energy consumption. (This document will not demonstrate the

effectiveness of the PID algorithm for now, but will first demonstrate the different effects between day and night time.)

- The user experience with occupancy sensor: The light identifies whether there is someone moving by reading the status message sent by the occupancy sensor, if yes it will automatically triggers a change in the light's status, without the need for the app or speaker to receive sensor data first and then send control ONOFF commands to the light.
- The user experience with energy monitor sensor: Detect devices with high energy consumption, and if abnormal energy consumption is found, prompt whether to replace or improve the equipment to achieve the goal of energy conservation.

Note:

> The sensor node sends a sensor status message instead of directly sending control commands such as onoff set and lightness set message, because it is required that many nodes need to listen to the status of this sensor, but the actions they take after receiving it may be different, such as performing light on/off operations or brightness adjustment operations. If the sensor sends a fixed control commands, then this function cannot be implemented.

29.3 Publish_adress Configuration Methods

NLC often uses the publish set command to configure the destination address for messages sent by a button or sensor.

If you want to change the publish adress, you can set it through the ini command of the host computer, or you can configure it through the App. Please refer to "33.2.7 Device Setting (Switch Device)" of the chapter Android and iOS APP User Guide for the configuration method of the App. Device Setting (Switch device)".

The following is an example of the gateway sending the publish set command:

```
CMD-cfg_pub_set_sig     =e8 ff 00 00 00 00 00 00 02 00 03 03 00 bb 00 00 00 ff 00 15 05 12
```

The data "03 00 bb 00 00 00 00 ff 00 15 05 12" corresponds to the following structure. spec V1.1 corresponds to section <4.3.2.16 Config Model Publication Set>:

```
typedef struct{
    u16 ele_adr;
    u16 pub_adr;
    mesh_model_pub_par_t pub_par;
    u16 model_id; // u32 for vendor model
}mesh_cfg_model_pub_set_t;
```

The publish address of the device can be changed by changing the following three variables.

ele_adr: element adress, there is one or more elements on each device, so you need select the elements.

pub_adr: publish adress, the destination address of the publish message.

model_id: Each element has one or more models on it, so you have to select a model by its model id.

The following is the publish adress set by the ini command, the element adress of this setting is 0x0003, the publish ardess is 0x00bb, and the model id is 0x1205 (SIG_MD_SCENE_C).

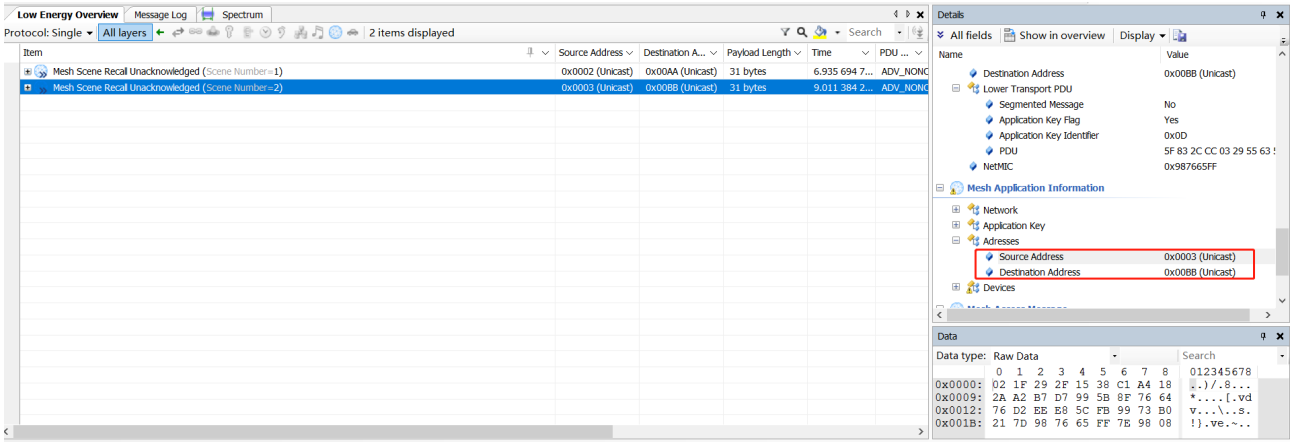


Figure 29.1: pub result

29.4 DICNLCP

29.4.1 Function

The Dimming Control NLC Profile (DIC) 1.0 – represents a wall slider, a dial, or a long-press switch function to dim lights up/down.

The corresponding spec is "DICNLCP_v1.0.pdf", for detailed function description, please refer to this document.

The main purpose of DICNLCP is to define a device that can send Generic Onoff, Delta Level, and Move Level messages and can configure the destination address of the messages.

The demo SDK DICNLCP test uses 825x_switch project of firmware demo SDK, the function is set to support DICNLCP and BSSNLCP at the same time. NLCP_DIC_EN and NLCP_BSS_EN are both equal to 1, customer can switch off NLCP_BSS_EN according to the need. NLCP_DIC_EN and NLCP_BSS_EN are both equal to 1. NLCP_DIC_EN and NLCP_BSS_EN can be switched off according to customer's need. it is at wake up mode as default beacause relay function is needed. If you want to disable relay and enter low power mode, just set SWITCH_ALWAYS_MODE_GATT_EN to 0 is enough. After disabling it, it will enter the low power mode.

29.4.2 nlc_switch Button

For hardware, use Remote control board, see "Switch operation" section of this document for details.

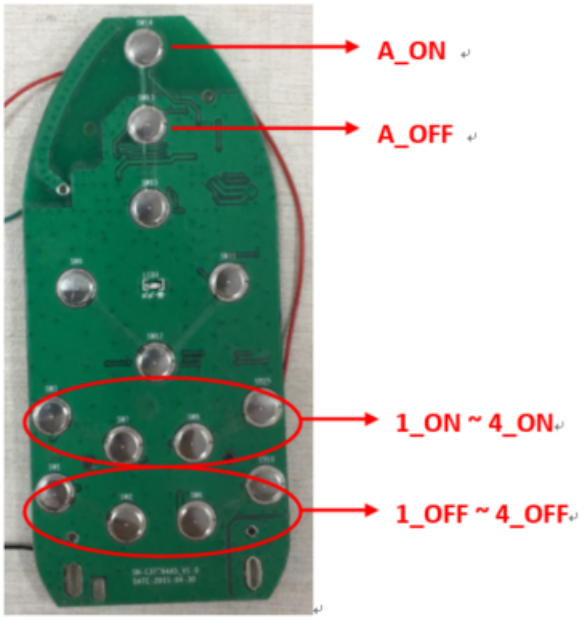


Figure 29.2: Switch button introduction

1_ON corresponds to RC_KEY_1_ON; 1_OFF corresponds to RC_KEY_1_OFF ..... .4_ON corresponds to RC_KEY_4_ON; 4_OFF corresponds to RC_KEY_4_OFF.

A_ON corresponds to RC_KEY_A_ON; A_OFF corresponds to RC_KEY_A_OFF.

29.4.3 Element Address

The 825x_switch project occupies 4 element addresses by default, the first element address is called ele_adr_primary, which is assigned during networking.

29.4.4 nlc_switch Button Functions

The demo SDK tests that DICNLCP only uses RC_KEY_1_ON, RC_KEY_1_OFF, RC_KEY_2_ON, RC_KEY_2_OFF, RC_KEY_3_ON, RC_KEY_3_OFF and RC_KEY_A_ON, RC_KEY_A_OFF to send commands.

When sending commands, RC_KEY_1_ON, RC_KEY_1_OFF use ele_adr_primary as the source address, and 0xC000(NLC_DICMP_GROUP_ADDR_START) as the destination address by default; send Generic Onoff command.

RC_KEY_2_ON, RC_KEY_2_OFF use ele_adr_primary + 1 as source address, default use 0xC001 as destination address; send Generic Onoff command.

RC_KEY_3_ON, RC_KEY_3_OFF use ele_adr_primary + 2 as source address, default use 0xC002 as destination address; send Generic Onoff command.

The up/down/right/left/right buttons (RC_KEY_UP / RC_KEY_DN / RC_KEY_L / RC_KEY_R) are used to do mode selection without sending commands, and the RC_KEY_A_ON and RC_KEY_A_OFF buttons also are used to do mode selection. See the following nlc_switch button onoff Command Mode, nlc_switch button delta_level Command Mode and nlc_switch key move_level command mode for how to do mode selection.

In addition for demo, RC_KEY_4_ON and RC_KEY_4_OFF are the keys to send scene store/recall command as BSSNLCP mode.

The above key function definition, customers can define and modify the key function according to the actual PCBA.

29.4.4.1 nlc_switch button onoff Command Mode

After the remote control board is powered on, the default value of select_pub_model_key in SDK is equal to ONOFF mode. RC_KEY_1_ON, RC_KEY_1_OFF, RC_KEY_2_ON, RC_KEY_2_OFF, RC_KEY_3_ON, RC_KEY_3_OFF send the Generic Onoff command, see "else branch" of dicmp_switch_send_publish_command() below for details:

```
void dicmp_switch_send_publish_command(u32 ele_offset, bool4 onoff, u32 select_pub_model_key)
{
......
    {
#if NLCP_DIC_EN
        if((RC_KEY_UP == select_pub_model_key)||(RC_KEY_DN == select_pub_model_key)){
            s32 delta = DICMP_LEVEL_DELTA_VALUE;
            if(!onoff){
                delta *= -1;
            }
            u16 pub_addr = dicmp_get_publish_addr(ele_offset, SIG_MD_G_LEVEL_C, 1);
            access_cmd_set_delta(pub_addr, 0, delta, CMD_NO_ACK, 0);
        }else if((RC_KEY_L == select_pub_model_key)||(RC_KEY_R == select_pub_model_key)){
            s16 move = DICMP_LEVEL_DELTA_VALUE;
            if(!onoff){
                move *= -1;
            }
            u16 pub_addr = dicmp_get_publish_addr(ele_offset, SIG_MD_G_LEVEL_C, 1);
            access_cmd_set_level_move(pub_addr, 0, move, CMD_NO_ACK, 0);
        }else{ // SIG_MD_G_ONOFF_C
            u16 pub_addr = dicmp_get_publish_addr(ele_offset, SIG_MD_G_ONOFF_C, 0);
            access_cmd_onoff(pub_addr, 0, onoff ? G_ON : G_OFF, CMD_NO_ACK, 0);
        }
#endif
    }
}
```

29.4.4.2 nlc_switch button delta_level Command Mode

Press RC_KEY_UP or RC_KEY_DN to set the value of select_pub_model_key to Delta Level mode. The RC_KEY_1_ON, RC_KEY_1_OFF, RC_KEY_2_ON, RC_KEY_2_OFF, RC_KEY_3_ON, RC_KEY_3_OFF will send the Delta Level command, which controls the level of the target node to increase or decrease by the specified value. For details, see spec "MshMDL_v1.1.pdf", chapter "3.2.2.4 Generic Delta Set".

RC_KEY_1_ON，RC_KEY_1_OFF – The default destination address of the message is: 0xD000 (NLC_DICMP_GROUP_ADDR_START_LEVEL_MODEL)

RC_KEY_2_ON，RC_KEY_2_OFF – The default destination address for messages is: 0xD001

RC_KEY_3_ON, RC_KEY_3_OFF – The default destination address of the message is: 0xD002

29.4.4.3 nlc_switch button move_level Command Mode

Press RC_KEY_L or RC_KEY_R to set the value of select_pub_model_key to Move Level mode. The RC_KEY_1_ON, RC_KEY_1_OFF, RC_KEY_2_ON, RC_KEY_2_OFF, RC_KEY_3_ON, RC_KEY_3_OFF will send Move Level command.

Move level message simply means that the level value of the control node changes from the current value to the maximum value (when the move level parameter of the move command is positive) or to the minimum value (when it is negative) at the specified rate. See spec "MshMDL_v1.1.pdf", chapter "3.2.2.6 Generic Move Set", etc. for details.

The default value for the destination address of a message is the same as the delta level mode.

29.4.4.4 nlc_switch button to Switch to on/off Command Mode

Pressing RC_KEY_A_ON or RC_KEY_A_OFF restores the value of select_pub_model_key to ONOFF mode, which corresponds to sending the Generic Onoff command.

29.4.5 Test Steps

29.4.5.1 SDK Settings

- The firmware SDK uses the 825x_switch project, sets LIGHT_TYPE_SEL to LIGHT_TYPE_NLC_CTRL_CLIENT, and compiles the firmware.

Note that at this point, the configuration related to model enablement uses the configuration information in nlc_ctrl_client_model_config.h instead of mesh_config.h.

29.4.5.2 Add to Network

To add the switch nodes to network, you can refer to section 12.5 Switch Engineering Long Press Handling Logic.

29.4.5.3 Supplement of Group Add Command in App

Take using app to add colour temperature light to the living room as an example. For ease of use, it actually sends 4 group add commands to these 4 models by default: SIG_MD_G_ONOFF_S, SIG_MD_LIGHTNESS_S, SIG_MD_LIGHT_CTL_S, SIG_MD_LIGHT_CTL_TEMP_S.

Since lightness and temperature model are two independent state, and now belong to the same room. if we need to use level control commands such as level delta to control the room, we cannot use 0xC000 as

the destination address to control lightness, because the temperature model will also be controlled, which does not meet expectation.

Therefore when it is needed to use the level command, open the App's "home page" – "Setting" – "Enable Subscription Level Service model Id" function switch, and let the App add the group number 0xD000 to element 0 (element containing the lightness model); add the group number 0xD001 to element 1 (element containing the temperature model).

In this way, level control commands such as level delta can be used to control lightness via group number 0xD000 and colour temperature via group number 0xD001.

- If it is added to "Kitchen room", it corresponds to "0xD010, 0xD011";
- If it is added to "Master bedroom", it corresponds to "0xD020, 0xD021".

29.4.5.4 Key Default Function Test

Step-by-step details can be found in "nlc_switch Button Functions", the 3 modes mentioned were tested.

(1) App mode settings and grouping

- Open the App's "home page" – "Setting" – "Enable Subscription Level Service model Id" function switch.
- Use App to add the first group of the light node to "Living room" (involves group number 0xC000,0xD000,0xD001, see Supplement of Group Add Command in App for details).
- Use the App to add the group part of the light node to the "Kitchen room" (involving group numbers 0xC001,0xD010,0xD011).
- Use the App to add the third group of the light node to the "Master bedroom" (involving group numbers 0xC002,0xD020,0xD021).

(2) Generic on/off test

- Press RC_KEY_1_ON or RC_KEY_1_OFF to switch to onoff mode. (The remote control belongs to onoff mode by default when it is just powered on).
- Pressing the keys RC_KEY_1_ON and RC_KEY_1_OFF controls the onoff state of the nodes belonging to the Living room (0xC000) group.
- Pressing the keys RC_KEY_2_ON and RC_KEY_2_OFF controls the onoff state of the nodes belonging to the Kitchen room (0xC001) group.
- Pressing the keys RC_KEY_3_ON and RC_KEY_3_OFF controls the onoff state of the nodes belonging to the Master bedroom (0xC002) group.

(3) Generic Level Delta test

- Press key RC_KEY_UP or RC_KEY_DN to switch to Delta Level mode.
- The keys RC_KEY_1_ON and RC_KEY_1_OFF controls the increase or decrease of the lightness of the nodes belonging to the Living room (0xC000) group.
- The keys RC_KEY_2_ON and RC_KEY_2_OFF control the increase and decrease of lightness of the nodes belonging to the Kitchen room (0xC001) group.
- The keys RC_KEY_3_ON and RC_KEY_3_OFF control the increase and decrease of lightness of the nodes belonging to the Master bedroom (0xC002) group.

(4) Generic Level Move test

- Press key RC_KEY_L or RC_KEY_R to switch to Move Level mode.

- The keys RC_KEY_1_ON and RC_KEY_1_OFF control the nodes belonging to the Living room (0xC000) group to increase the lightness to the maximum or decrease it to the minimum value.
- The keys RC_KEY_2_ON and RC_KEY_2_OFF control the increase of the lightness to the maximum or the decrease to the minimum value for the nodes belonging to the Kitchen room (0xC001) group.
- The keys RC_KEY_3_ON and RC_KEY_3_OFF control the nodes belonging to the Master bedroom (0xC002) group to increase the lightness to the maximum or decrease it to the minimum value.

29.4.5.5 Configure the Publish Address Test for the Key

Configure the destination address of the command of a key by sending the publish set command. see Publish_adress Configuration Methods for the description of publish.

The following is an example of the publish address for the level delta command that configures RC_KEY_1_ON and RC_KEY_1_OFF:

Method 1. The INI commands actually tested by the host computer are referenced below:

```
CMD-cfg_pub_set_sig = e8 ff 00 00 00 00 00 00 02 00 03 02 00 01 D0 00 00 ff 00 15 03 10
```

This command sets the publish address of the level client model (0x1003) on element address (0x0002) in the remote control to 0xD001.

Method 2. The App is tested as follows

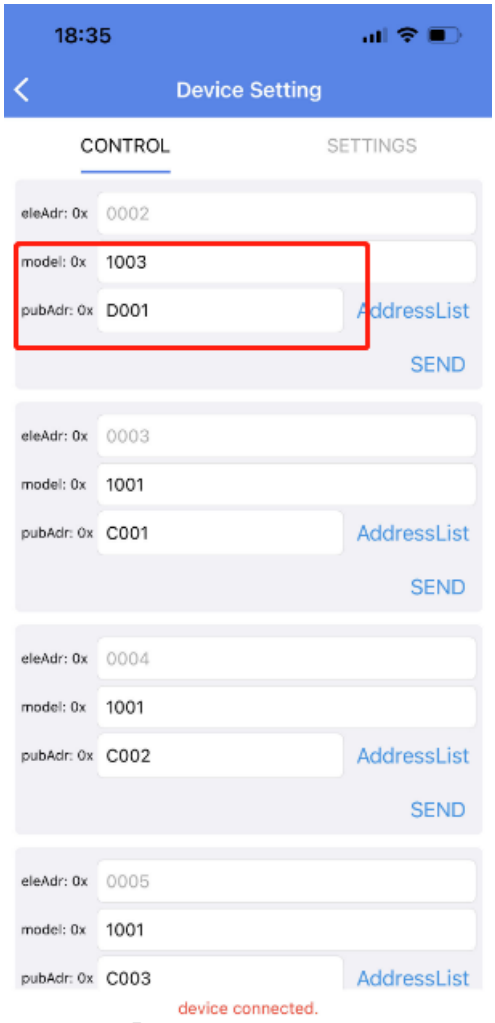


Figure 29.3: app pub set

model: 0x1003 (Generic Level Client), because you want to configure the publish address of the level delta or level move command.

pubAdr: This is set to the destination address of the command sent from this button. Here, for example, it is set to 0xD001 to control the colour temperature of the lamp belonging to "Living room".

29.5 BSSNLCP

29.5.1 Function Description

Basic Scene Selector NLC Profile (BSS) 1.0 – represents a wall switch or a wall station to select lighting scenes or turn the lights on/off.

Please refer to spec "BSSNLCP_v1.0.pdf" for the corresponding chapter introduction and this document for detailed function introduction.

BSSNLCP Defines a device that can send the SCENE STORE/RECALL command and can configure the destination address of the messages.

29.5.2 Hardware Introduction

The firmware SDK uses 825x_switch project, for hardware we use remote control board, see "nlc_switch Button"

29.5.3 Button Functions

The demo SDK DICNLCP test uses 825x_switch project of firmware demo SDK, the function is set to support DICNLCP and BSSNLCP at the same time. NLCP_DIC_EN and NLCP_BSS_EN are both equal to 1, customer can switch off NLCP_BSS_EN according to the need. NLCP_DIC_EN and NLCP_BSS_EN are both equal to 1. NLCP_DIC_EN and NLCP_BSS_EN can be switched off according to customer's need. it is at wake up mode as default beacause relay function is needed. If you want to disable relay and enter low power mode, just set SWITCH_ALWAYS_MODE_GATT_EN to 0 is enough. After disabling it, it will enter the low power mode.

Test BSSNLCP sending command, currently the demo only uses RC_KEY_4_ON, RC_KEY_4_OFF key, other key definition please refer to "nlc_switch-button-functions", customer can redefine according to the actual need.

RC_KEY_4_ON sends SCENE_STORE command and RC_KEY_4_OFF sends SCENE RECALL command.

- RC_KEY_4_ON uses ele_adr_primary as source address and 0xC000(NLC_DICMP_GROUP_ADDR_START) as destination address by default; and it can be modified by sending publish set command.
- RC_KEY_4_OFF uses ele_adr_primary as source address and 0xC001(NLC_DICMP_GROUP_ADDR_START) as destination address by default; and it can be modified by sending publish set command.

See dicmp_switch_send_publish_command()'s "if branch" below for details:

```
void dicmp_switch_send_publish_command(u32 ele_offset, bool4 onoff, u32 select_pub_model_key)
{
#if NLCP_BSS_EN
    if(3 == ele_offset){
        u32 ele_offset_scene = 0;
        if(onoff){  // RC_KEY_4_ON
            ele_offset_scene = 0;
            u16 pub_addr = dicmp_get_publish_addr(ele_offset_scene, SIG_MD_SCENE_C, 0);
            sw_tx_src_addr_offset = ele_offset_scene;
            access_cmd_scene_recall(pub_addr, 0, 1, 0, 0);
        }else{      // RC_KEY_4_OFF
            ele_offset_scene = 1;
            u16 pub_addr = dicmp_get_publish_addr(ele_offset_scene, SIG_MD_SCENE_C, 0);
            sw_tx_src_addr_offset = ele_offset_scene;
            access_cmd_scene_recall(pub_addr, 0, 2, 0, 0);
        }
    }
    ...........
}
```

29.5.4 Test Steps

29.5.4.1 SDK Settings

- Firmware SDK using 825x_switch project, set LIGHT_TYPE_SEL to LIGHT_TYPE_NLC_CTRL_CLIENT.

Note:

> In this case, the configuration related to model enablement uses the configuration information in nlc_ctrl_client_model_config.h instead of the configuration information in mesh_config.h.

29.5.4.2 Add to Network

- To add the switch nodes to network, you can refer to section 12.5 Switch Engineering Long Press Handling Logic.

29.5.4.3 Button Test

- Pressing RC_KEY_4_ON will send SCENE_STORE_NOACK, and pressing RC_KEY_4_OFF will send SCENE_RECALL_NOACK.

For both RC_KEY_4_ON and RC_KEY_4_OFF, the source address is primary address, the default destination address is Living room(0xC000), and the scene ID is 0x0001.

- You can check whether the command was sent successfully by capturing the packet, as shown below:

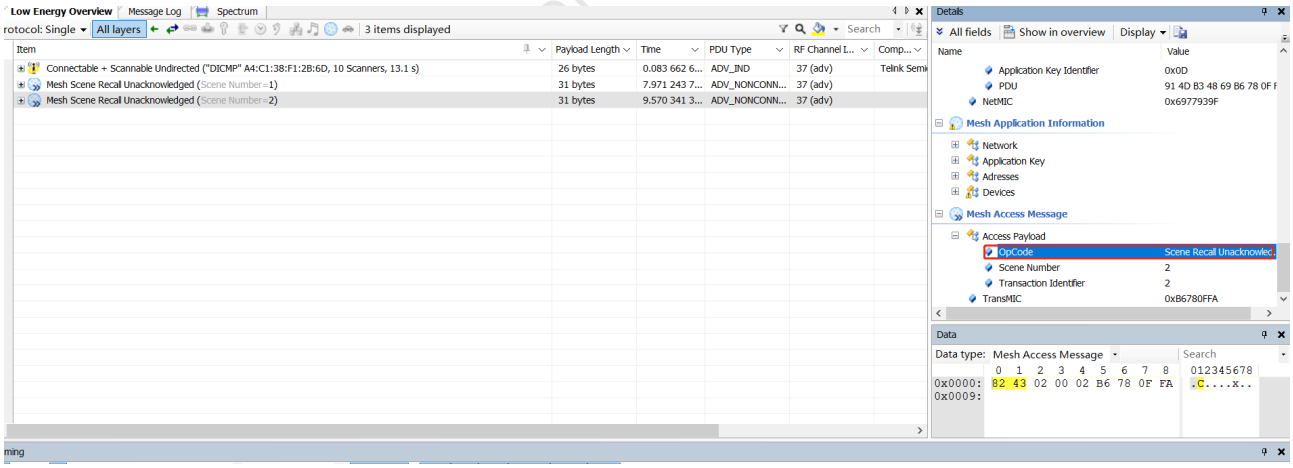


Figure 29.4: scene_recall

Configure the destination address of the key by sending the publish set command. see Publish_adress Configuration Methods for the description of publish.

The following is an example of the publish address of the scene recall command with RC_KEY_4_ON and RC_KEY_4_OFF configured:

Method 1. The INI commands actually tested by the host computer are referenced below:

```
CMD-cfg_pub_set_sig = e8 ff 00 00 00 00 00 00 02 00 03 02 00 02 c0 00 00 ff 00 15 05 12
```

This command sets the publish address of scene client model (0x1205) to Master bedroom (0xC002) on element address (0x0002) in the remote control.

Method 2. The App was tested as follows:

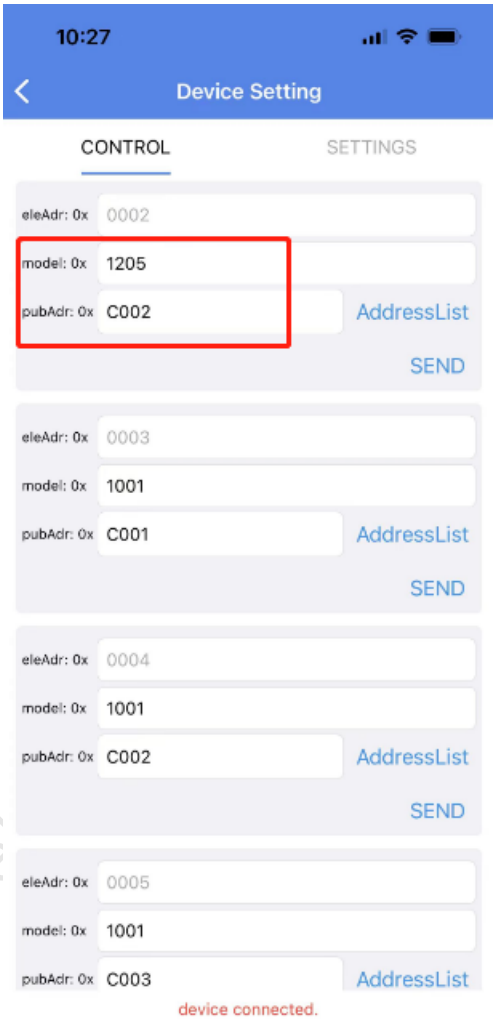


Figure 29.5: app pub set

model: Because you want to configure the publish address of the scene recall command, fill in 0x1205 (Scene Recall Client Model) here.

pubAdr: Set it as the destination address of the command sent by this button, here, for example, scene 1 of the lamp belonging to the Master bedroom is modified to Recall, so it is set to 0xC002.

29.6 BLCNLCP

29.6.1 Function Description

Basic Lightness Controller NLC Profile (BLC) 1.0 – represents a luminaire with an integrated controller.

The corresponding spec is "BLCNLCP_v1.0.pdf", for detailed function introduction, please refer to this document.

BSSNLCP: Define a light type device, the device not only supports common switch, brightness, colour temperature and other controls, but also supports light control model. when light control mode is off, it is an ordinary light, the state of the light is controlled by commands such as generic onoff, lightness set, etc. When light control mode is enabled, the light control server model corresponding to the light node can directly receive the status from the sensor, such as OCCUPANCY sensor(PRESENCE_DETECTED), Ambient light sensor, etc., and then do the corresponding control on the status of the light node itself. Commonly used application scenarios, such as the light in the stairway, when detecting someone moving, it will automatically light up for a period of time, and then automatically fade to the standby brightness state, the standby brightness value is customisable, it can be 0, or it can be a relatively low brightness value.

29.6.2 Hardware Introduction

Hardware uses 825x dongle.

29.6.3 Test Steps

Note that the light control server model also supports the generic onoff model, so when a generic onoff message is received, the light control model is required to occupy an element independently in order to distinguish whether it is controlling the brightness on/off state or light control onoff state. see "Table 6.186: Light LC Setup Server elements and states" in MshMBT_v1.0.pdf for details.

After the light control model is independent as an element, you need to pay attention to the configuration of the element address in the parameter area for the group add command for the light control model. the light control model is on the last element. For color temperature lamp, it is equal to primary address + 2, and for HSL lamp, it is primary address + 3. The same rule for setting LC mode onoff.

29.6.3.1 SDK Settings

Firmware SDK use 825x_mesh project, set NLCP_BLC_EN to 1.

(1) When the node is powered on, the light control mode is off by default, and the method of controlling the brightness and other states of the light is the same as the operation without the light control function.

(2) App Sends the Light LC mode set enable command to the node or through an INI command:

```
light_lc_mode_set       =e8 ff 00 00 00 00 00 00 02 00 82 92 01
```

Note:

> The destination address is not primary address (node address), but primary address +(ELE_CNT_EVERY_LIGHT - 1).

(3) App sends LIGHT_LC_ONOFF_SET on command to the node, or via an INI command:

```
light_lc_onoff_set      =e8 ff 00 00 00 00 00 00 02 00 82 9a 01 00
```

Note:

> The destination address is not primary address (node address), but primary address +(ELE_CNT_EVERY_LIGHT - 1).

The nodes change brightness according to the following curve.

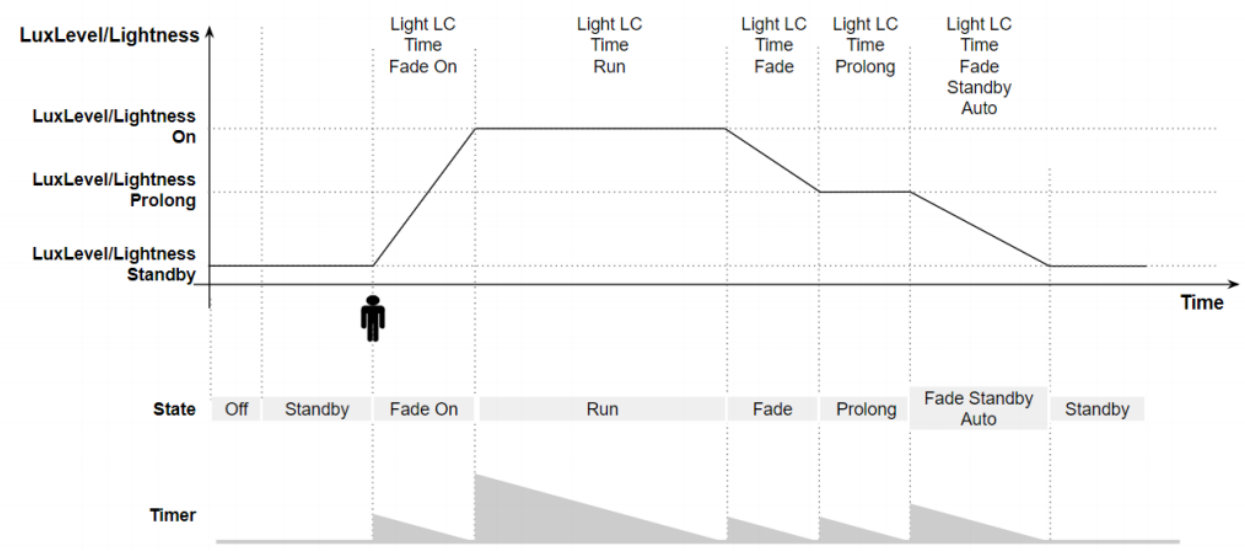


Figure 6.4: Operation of a Light Lightness Controller

Figure 29.6: Operation_of_a_Light_Lightness_Controller

The time parameters and luminance values of each step of the curve are defined by the following macros:

```
#define LC_PROP_VAL_LightnessOn             (LIGHTNESS_MAX)
#define LC_PROP_VAL_LightnessProlong        ((LIGHTNESS_MAX + 1) / 4)
#define LC_PROP_VAL_LightnessStandby        ((LIGHTNESS_MAX + 1) / 20)
......
#define LC_PROP_VAL_TimeFade                (2*1000)    // unit: ms
#define LC_PROP_VAL_TimeFadeOn              (2*1000)    // unit: ms
#define LC_PROP_VAL_TimeFadeStandbyAuto     (3*1000)    // unit: ms
#define LC_PROP_VAL_TimeProlong             (4*1000)    // unit: ms
#define LC_PROP_VAL_TimeRun                 (5*1000)    // unit: ms
```

(4) Wait for step 3 to enter the standby state, and then send the INI command of OCCUPANCY sensor(PRESENCE_DETECTED) status to the node, and the node will also change its brightness according to the curve in step 3.

```
sensor_occupancy_64 = e8 ff 00 00 00 00 00 00 02 00 52 01 42 00 64
```

(5) Wait for step 4 to enter standby state, then send the sensor status command to the lamp node via the sensor node introduced by "ocssnlcp", and the lamp node will also change its brightness according to the curve in step 3.

When the destination address of the sensor status message is a group address, you also need to subscribe to the group address for the generic onoff model or light control model on LC model element, the INI command example is as follows:

```
light_lc_model_sub_set = e8 ff 00 00 00 00 02 01 02 00 80 1b 04 00 01 c0 0f 13
or
light_g_onoff_sub_set = e8 ff 00 00 00 00 02 01 02 00 80 1b 04 00 01 c0 00 10
```

29.7 ocssnlcp

29.7.1 Function Description

Occupancy Sensor NLC Profile (OCS) 1.0 – represents an occupancy sensor

The corresponding spec is "OCSNLCP_v1.0.pdf", please refer to this document for detailed function description.

When a person is detected approaching, the state of the sensor changes, which triggers the sensor to publish sensor status, which contains the current status of whether there is a person or not.

29.7.2 Test Steps

29.7.2.1 SDK Settings

Firmware SDK using 825x_mesh_project

- Set LIGHT_TYPE_SEL to LIGHT_TYPE_NLC_SENSOR.
- NLC_SENSOR_TYPE_SEL Select NLCP_TYPE_OCS
- SENSOR_PROP_ID Select PROP_ID_PRESENCE_DETECTED

```
#elif (NLC_SENSOR_TYPE_SEL == NLCP_TYPE_OCS)
#define SENSOR_PROP_ID   PROP_ID_PRESENCE_DETECTED
```

Then compile it to get the firmware.

29.7.2.2 Function

- Publish set using the ini command

```
CMD-cfg_pub_set_sig = e8 ff 00 00 00 00 02 00 02 00 03 02 00 ff ff 00 00 FF 00 00 00 11
```

For the description of the above commands, you can refer to Publish_adress Configuration Methods, and note that you have to modify the destination address, element address and publish address inside the cfg_pub_set_sig command according to the unicast address of the current test node.

The sensor_measure_proc() function polls the status value detected by the sensor to see if the trigger condition is reached, if it is reached, the value of pub_flag will be 1, and then it will see if there is a publish address configured, if there is one, then it will send a sensor status message, see the processing of sensor_measure_proc() for details.

```
u32 sensor_measure_proc()
{
    ...
    u32 measure_val = 0;
    memcpy(&measure_val, p_sensor_data->p_raw, min2(sizeof(measure_val),
↪   p_sensor_data->len_raw));

    if(sensor_measure_quantity < measure_val){
        if((measure_val - sensor_measure_quantity) > p_cadence->cadence_unit.delta_down){
            pub_flag = 1;
        }
    }
    else{
        if((sensor_measure_quantity - measure_val) > p_cadence->cadence_unit.delta_up){
            pub_flag = 1;
        }
    }

    if(pub_flag){
        model_pub_check_set(ST_G_LEVEL_SET_PUB_NOW, (u8 *)&model_sig_sensor.sensor_srv[0].com,
↪   0);
    }
}
```

measure_val: Previous state value (0/1)

sensor_measure_quantity：Current status value (0/1)

pub_flag: Publish flag. it will be set to 1 when delta between the current state value and the previous one reaches the threshold.

- The change amount threshold can be configured by sending the Sensor Cadence Set, when not configured, p_cadence->cadence_unit.delta_down/p_cadence->cadence_unit.delta_up is 0.
- Since the current dongle board does not have a sensor peripheral, there are currently two ways for the demo sdk to change the sensor value:

Method 1: Go through the BDT to set the value of sensor_measure_quantity. For example, if you set 1 for the first time and 0 for the second time, the node will publish sensor status once. Or the node will publish sensor status once if it set 0 the first time and 1 the second time. In any case, the node publishes sensor status if the value is reversed from the previous time.

Method 2: Change the value of sensor_measure_quantity by pressing the key. In addition to the above "SDK Settings" in this section, open the UI_KEYBOARD_ENABLE, then compile SDK，then burn the compiled firmware to 8258_dongle, then every time you press SW2, the value of sensor_measure_quantity will change once, and the node will publish status once, and the implementation code is as follows:

```
void mesh_proc_keyboard (void)
{
    ......
        if(KEY_SW2 == kb_event.keycode[0]){
            sensor_measure_quantity = !sensor_measure_quantity;
        }
    ......
}
```

The following image shows the result of a packet capture of the publish sensor status:

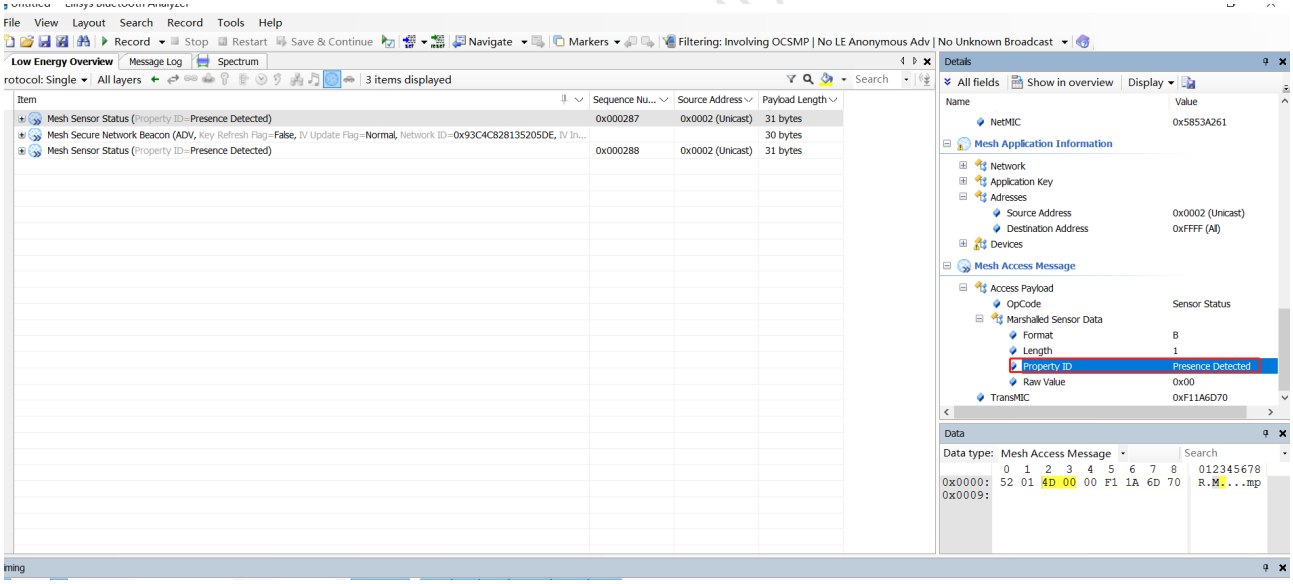


Figure 29.7: pub ocs

Property ID: PROP_ID_PRESENCE_DETECTED(0x004D), it means that the sensor at this moment is the occupancy sensor.

29.8 ALSNLCP

29.8.1 Function Description

Ambient Light Sensor NLC Profile (ALS) 1.0 – represents an ambient light level sensor.(Ambient light sensor)

The corresponding spec is "ALSNLCP_v1.0.pdf", for detailed function description, please refer to this document.

When the detected change in ambient illumination is greater than the set threshold, the sensor will be triggered to publish sensor status, and the sensor status contains the current ambient light level.

29.8.2 Test Steps

29.8.2.1 SDK Settings

The firmware SDK uses 825x_mesh_project, sets LIGHT_TYPE_SEL to LIGHT_TYPE_NLC_SENSOR, enables it, and compiles the firmware.

29.8.2.2 Function Test ALSNLCP

- Publish set using the ini command

```
CMD-cfg_pub_set_sig = e8 ff 00 00 00 00 02 00 02 00 03 02 00 ff ff 00 00 FF 00 00 00 11
```

For a description of the above commands you can refer to Publish_adress Configuration Methods.

- Use the ini command to send Sensor Cadence Set to set the threshold for the amount of change in ambient light level.

```
e8 ff 00 00 00 00 02 00 02 00 55 4E 00 02 01 00 00 01 00 00 0C F0 FF 00 10 00 00
```

The structure corresponding to the command is as follows:

```
typedef struct{
    u16 prop_id;
    sensor_cadence_t cadence;
    sensor_setting_par_t setting;
}sensor_states_t;
```

prop_id: sensor property id, the setting here is PROP_ID_PRESENT_AMBIENT_LIGHT_LEVEL(0x004E)

trig_type: State trigger type. 0: Change value. 1: Percentage.

delta_down: The minimum change value that triggers a node to publish sensor status when the ambient light level decrease. The publish is triggered when the illumination reduction reaches the set threshold, which is 0x000001, but can be set to any other value.

delta_up: The minimum change value that triggers a node to publish sensor status when the ambient light level increase. The publish is triggered when the amount of illumination reaches a set threshold, which is 0x000001, but can be set to any other value.

- Set the value of sensor_measure_quantity, i.e. the current illuminance, through BDT. For example, if the current sensor_measure_quantity value is 0, and it is set to 10, then the change in illumination is greater than the set threshold, then the node will trigger publish status once.

The following is the result of the publish sensor status packet capture.

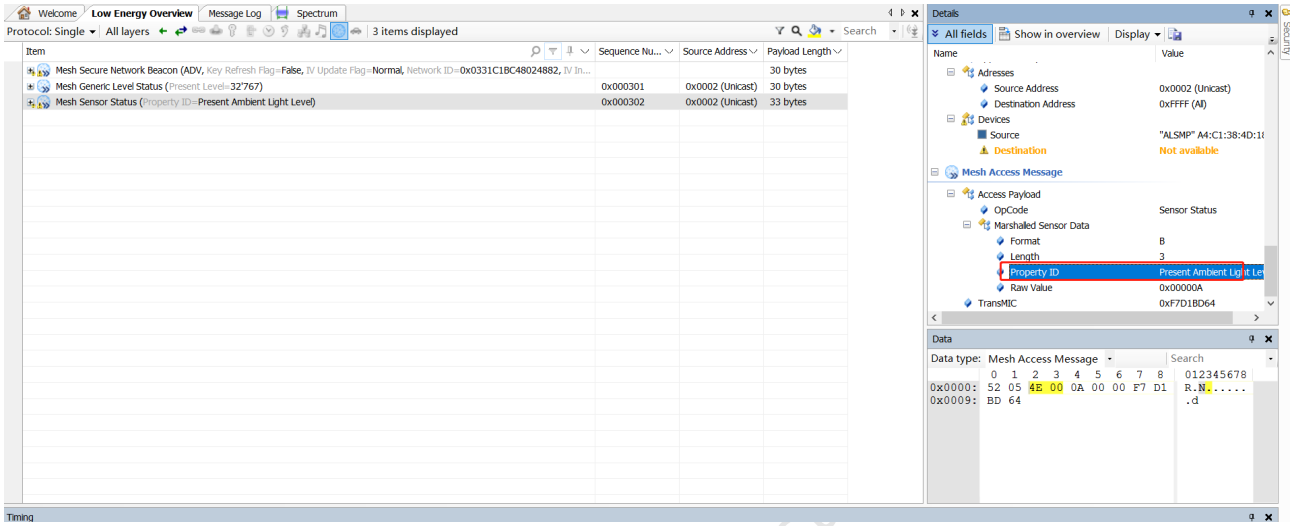


Figure 29.8: pub als

Property ID: PROP_ID_PRESENT_AMBIENT_LIGHT_LEVEL(0x004E), indicates that the sensor status contains the light level.

29.9 ENMNLCP

29.9.1 Function Description

Energy Monitor NLC Profile (ENM) 1.0 – represents a sensor reporting energy consumption

The corresponding spec is "ENMNLCP_v1.0.pdf", for detailed function description, please refer to the energy consumption monitor part in this document.

29.9.2 Test Steps

29.9.2.1 SDK Settings

Firmware SDK using 825x_mesh_project

- Set LIGHT_TYPE_SEL to LIGHT_TYPE_NLC_SENSOR.
- For NLC_SENSOR_TYPE_SEL, select NLCP_TYPE_ENM.

Then compile it to get the firmware.

29.9.2.2 Function Test

- Publish set using the ini command.

```
CMD-cfg_pub_set_sig = e8 ff 00 00 00 00 02 00 02 00 03 02 00 ff ff 00 00 FF 00 00 00 11
```

For a description of the above commands you can refer to Publish_adress Configuration Methods.

- The amount of change threshold can be configured by sending the Sensor Cadence Set, which defaults to 0 when not configured, same goes for setting delta_down. delta_up. How to do, please refer to Function Test ALSNLCP.
- Go through the BDT to set the value of sensor_measure_quantity, which is the current energy value. For example, the value of sensor_measure_quantity is 0, set it to 10, then the node will publish status.

The following is the result of the publish sensor status packet capture.

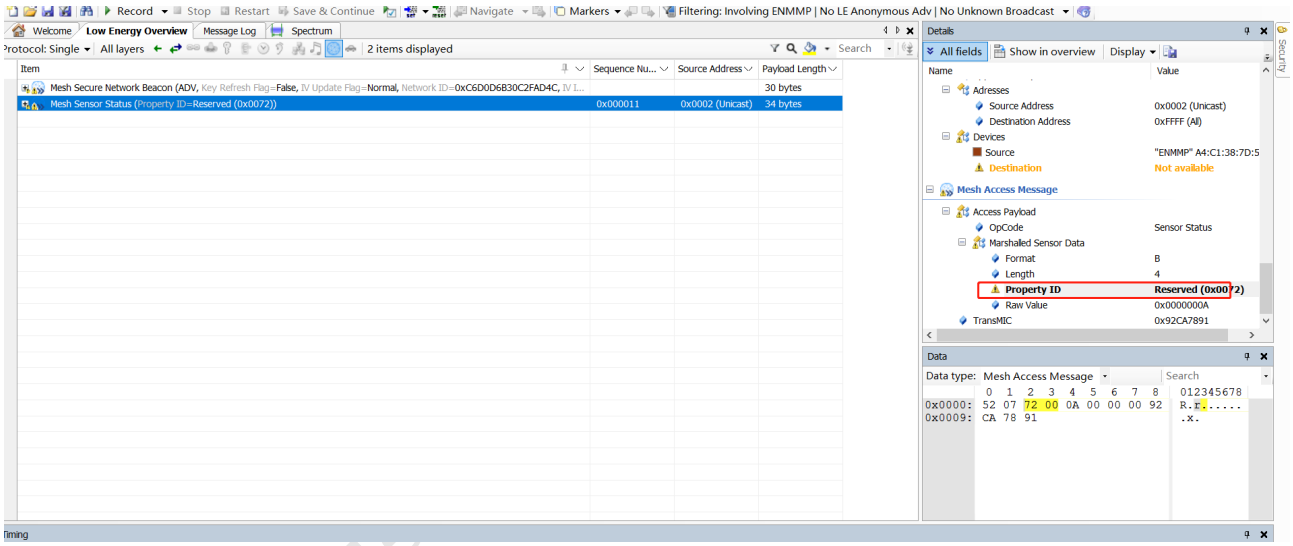


Figure 29.9: pub enm

Property ID: PROP_ID_PRECISE_TOTAL_DEVICE_ENERGY_USE(0x0072) indicates that the sensor status contains energy detection.

30 Ellisys Decrypts Mesh Packets

30.1 Click Record to Grab the Packet

30.2 Fill in Mesh Information for Decryption

- Click on the mesh security option in the menu bar view drop down box and fill in the mesh key information in big endian format (network key, appkey key, and IV index. If you want to see messages using device keys such as the configuration model, such as the publish set command, you also need to fill in the device key of the device).

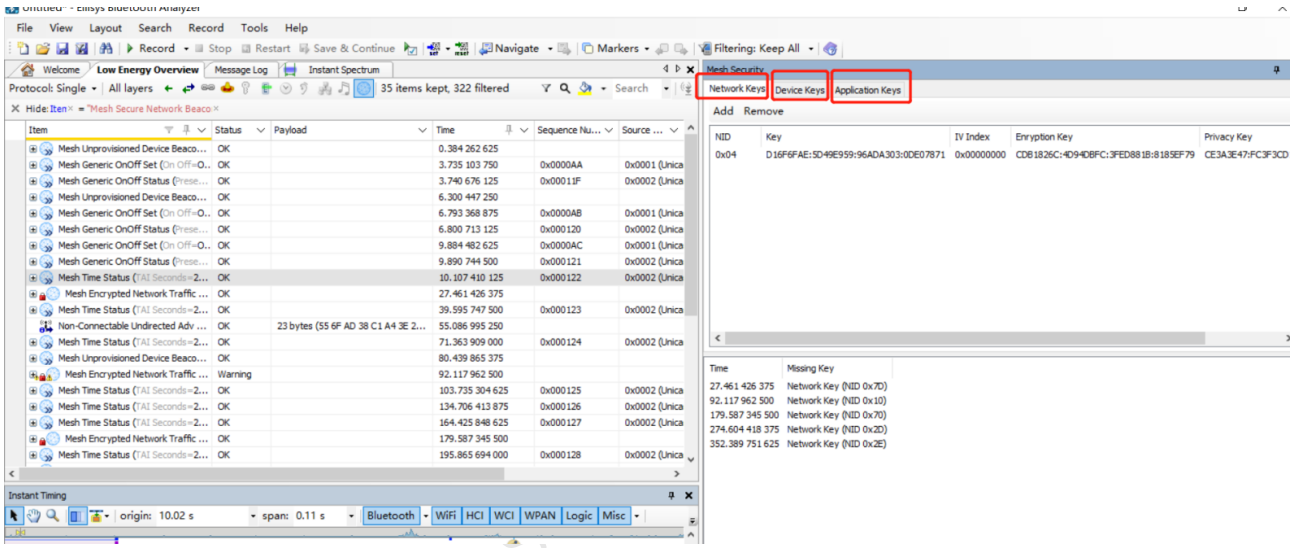


Figure 30.1: Mesh security

- If you don't know the key, you can use BDT tool to get the key from the node, mesh_key global variables include: device key, network key and app key; iv index is in iv_idx_st. The structure of different sdk versions may change, The location of each member field of global variable "mesh_key" and "iv_idx_st" can be found in the code. Take V3.3.3 as an example.

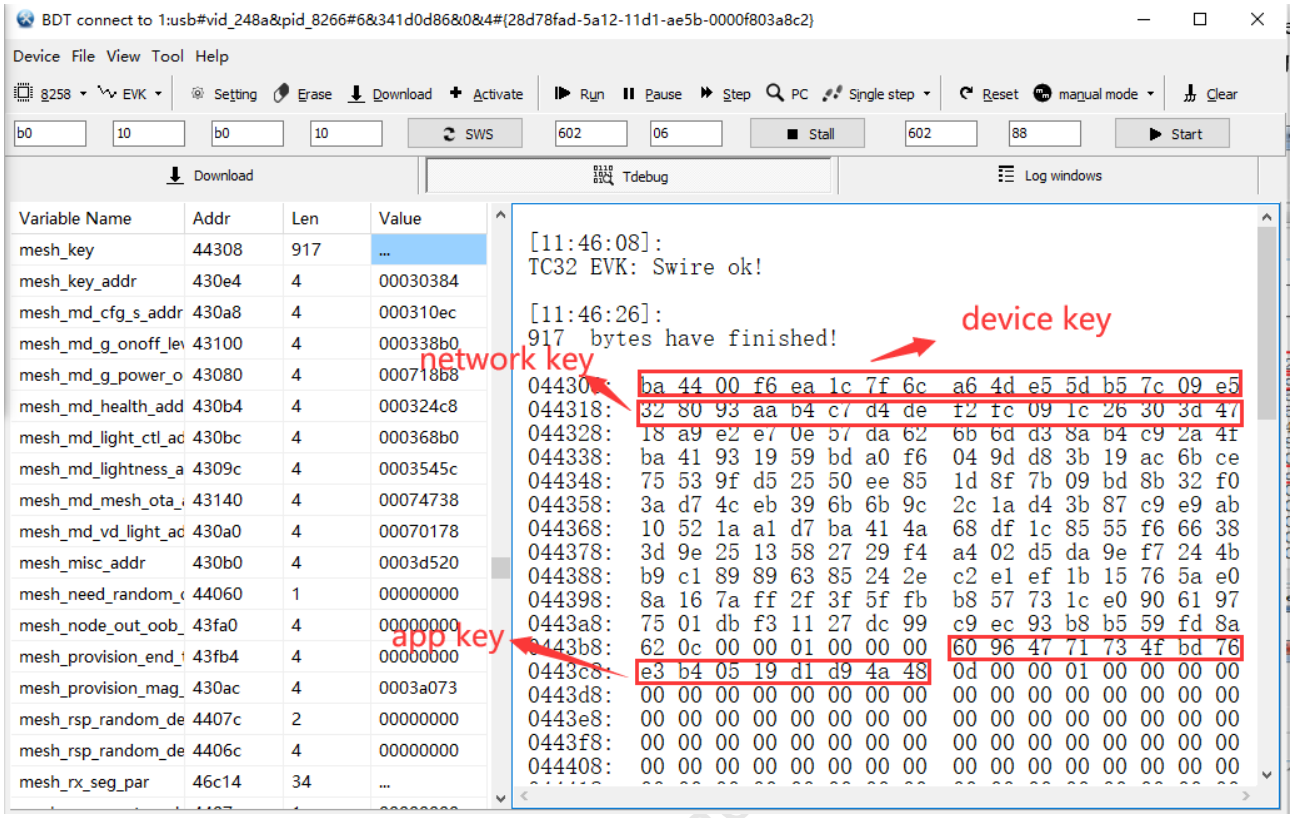


Figure 30.2: Get mesh key

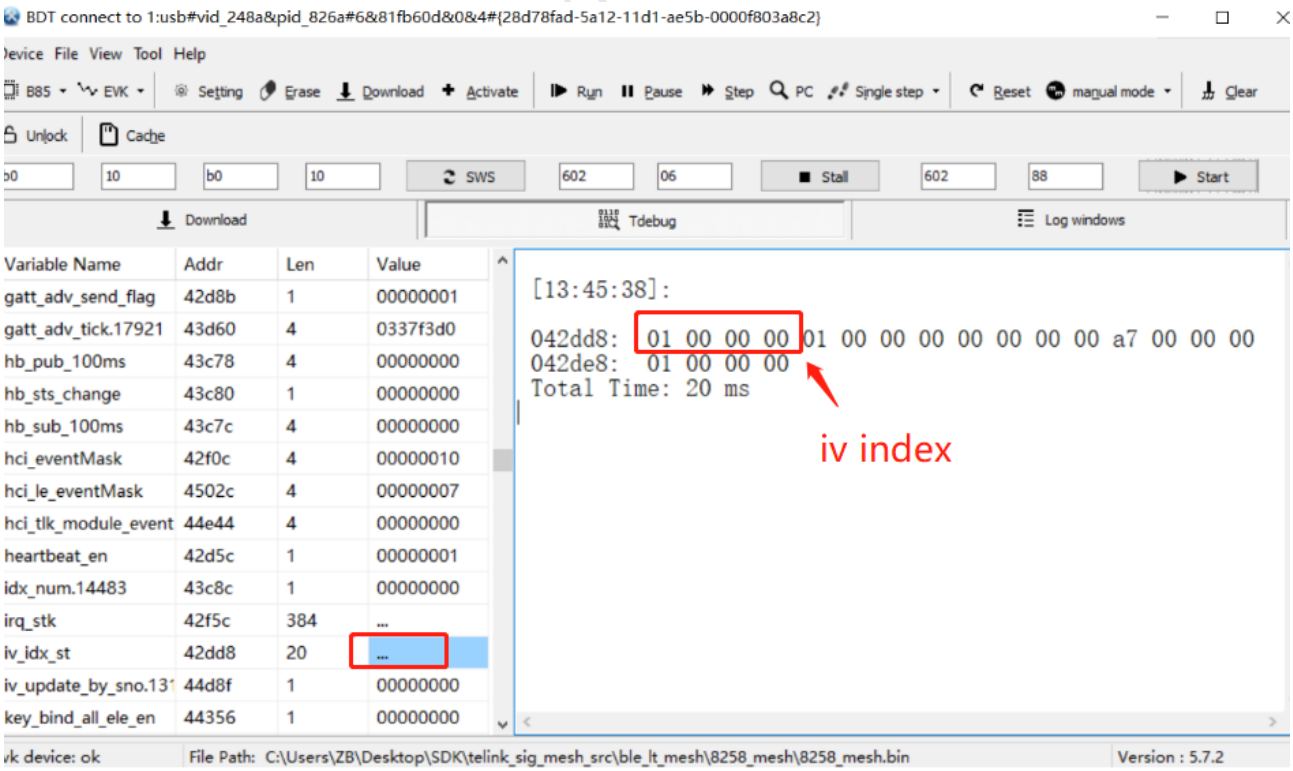


Figure 30.3: Get mesh iv

Note:

> The iv index in BDT is the little endianness of the display, in ellisys need to fill in the big endianness, such as the above figure is to fill in "00000001" or "01". if SIG Mesh SDK version is later than V4.1.0.0(include), the iv index in BDT is the big endianness of the display.

- Perform mesh decryption and view it.

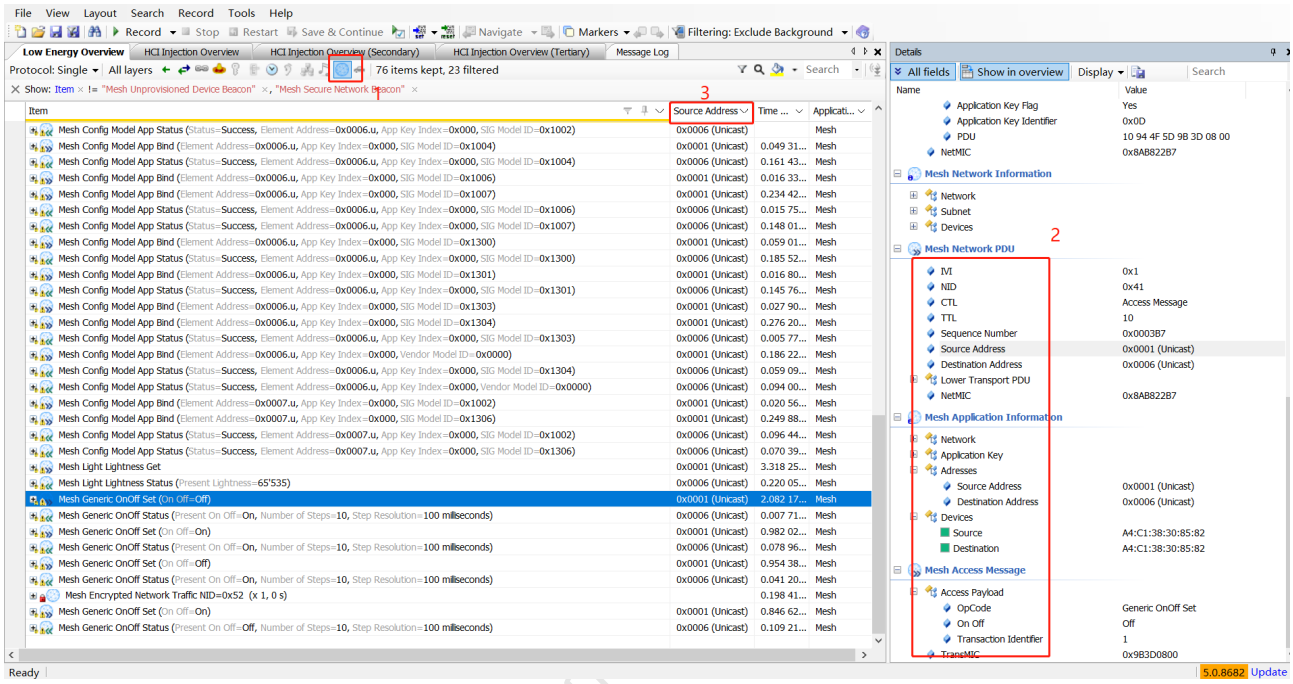


Figure 30.4: Mesh security set

An explanation of steps 1, 2, and 3 in the figure:

1) Click the "mesh" button for mesh decryption;
2) You can use the mouse to drag a certain item from "All Fields" on the right to the main interface for display;
3) The above figure is an example of step 2.

30.3 Other Methods to Get the Key

30.3.1 Provision UART Log of provision flow Via Firmware

```
10:16:23.500] [LIB]:(sdk)mesh tx NoAck,op:0x4e82(LIGHTNESS_STATUS),src:0x53b7,dst:0xffff,sno:0x000081 par_len:2 par:ff ff
10:16:29.551] [LIB]:(sdk)mesh_ble_connect_cb
10:16:30.900] [LIB]:(sdk)rcv provision invite 00 00
10:16:30.902] [LIB]:(sdk)send capa cmd 01 02 00 01 00 00 00 00 00 00 00 00
10:16:31.749] [LIB]:(sdk)rcv start cmd 02 00 00 00 00 00
10:16:31.754] [LIB]:(sdk)rcv pubkey cmd
10:16:31.754] 03 44 93 c1 19 55 eb fc  6c c3 77 fa a9 e2 25 bf  4e 36 8f f4 50 8a d6 9f  2f 9f f8 d9 33 43 76 fb
10:16:31.756] 9c de bd 61 1e 6d a5 50  0d 86 fe b8 00 f4 be da  3d 9d 71 c8 ae 34 78 17  35 f9 66 c9 fb e6 30 2a
10:16:31.765] 15
10:16:31.771] [LIB]:(sdk)send pubkey cmd
10:16:31.771] 03 9d df 25 57 20 75 81  80 85 28 ce 81 da be 03  f8 98 48 9d 70 fd 69 f6  a2 9b 4e 55 68 bc 96 01
10:16:31.771] f1 43 ff aa 03 c5 99 b5  a5 4f 4e 4b 77 94 6b 2b  75 09 11 f8 a6 eb 44 97  a1 1c 0e 2a b7 ff 7a 04
10:16:31.771] a1
10:16:31.921] [LIB]:(sdk)rcv comfirm cmd 05 09 d9 bc 6c 77 38 9b 3f cc 29 2d 76 4d eb ca 3a
10:16:34.019] [LIB]:(sdk)send comfirm cmd 05 4a c5 2e 52 ec 99 f4 ce bf 3e 69 06 5e 44 68 30
10:16:34.082] [LIB]:(sdk)rcv random cmd 06 69 02 40 92 99 db d7 c7 9f d1 b8 86 1e a7 95 af
10:16:34.092] [LIB]:(sdk)send random cmd 06 97 e2 ac 2b ef f9 6a e5 05 2f e6 a5 35 68 fc 9c
10:16:34.157] [LIB]:(sdk)rcv provision data cmd
10:16:34.159] 07 da 91 0e 48 cd 2e 66  d1 e0 5c 40 98 43 fe d5  a2 0d 57 55 30 77 fe ae  3a f4 59 db 0d 0c 0f 09
10:16:34.164] 44 94
10:16:34.171] [LIB]:(provision)device key e7 2b 2b 2e fd 05 d7 c6 77 cc 8b bf fb 4e 33 b0
10:16:34.180] [LIB]:(sdk)provision net info is f5 43 f2 2c 83 a3 14 a1 ce aa a2 e0 fc 9f ff 10 00 00 00 00 00 00 00 93 00
10:16:34.229] [LIB]:(sdk)provision suc!
10:16:38.247] [LIB]:(sdk)mesh_ble_disconnect_cb
10:16:38.810] [LIB]:(sdk)mesh_ble_connect_cb
10:16:40.107] [LIB]:(iv_update)app tx beacon with GATT,IV index step0: 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00
10:16:40.112] [LIB]:(iv_update)tx GATT secure NW beacon:17 2b 01 00 df 8c 67 d7 cf d0 3b 31 00 00 00 00 8f 85 00 67 38 ff fb 96
10:16:40.118] [LIB]:(sdk)add filter adr part 00 01
10:16:40.130] [LIB]:(sdk)tx upper_ctl,sno:0x1 op:0x3,dst:0x0 par:00 00 01
10:16:40.252] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c94b op:0x0000(APPKEY_ADD),par_len:19,par:00 00 00 6a 77 aa ce b3 88 98 8e de 9e 9d 1d 5d 52 99 68
10:16:40.267] [LIB]:(sdk)mesh tx NoAck,op:0x0380(APPKEY_STATUS),src:0x0093,dst:0x0001,sno:0x000003 par_len:4 par:00 00 00 00
10:16:40.324] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c94e op:0x0880(COMPOSITION_DATA_GET),par_len:1,par:00
10:16:40.333] [LIB]:(sdk)mesh tx NoAck,op:0x0002(COMPOSITION_DATA_STATUS),src:0x0093,dst:0x0001,sno:0x000004 par_len:61 par:00 2e 05 99 00 33 33 69 00 07 00 00 00 11 01 00 00 02 00 03 00 00 fe 00 ff 02 fe 01 fe 01 ff 00
10:16:41.679] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c950 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 02 00
10:16:41.690] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000a par_len:7 par:00 93 00 00 00 02 00
10:16:41.750] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c951 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 00 fe
10:16:41.766] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000b par_len:7 par:00 93 00 00 00 00 fe
10:16:41.865] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c952 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 00 10
10:16:41.883] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000c par_len:7 par:00 93 00 00 00 00 10
10:16:41.938] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c953 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 02 10
10:16:41.951] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000d par_len:7 par:00 93 00 00 00 02 10
10:16:42.013] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c954 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 00 13
10:16:42.024] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000e par_len:7 par:00 93 00 00 00 00 13
10:16:42.125] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c955 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 03 13
10:16:42.137] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x00000f par_len:7 par:00 93 00 00 00 03 13
10:16:42.200] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c956 op:0x3d80(MODE_APP_BIND),par_len:6,par:93 00 00 00 04 13
10:16:42.213] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x000010 par_len:7 par:00 93 00 00 00 04 13
10:16:42.274] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c957 op:0x3d80(MODE_APP_BIND),par_len:6,par:94 00 00 00 02 10
10:16:42.289] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x000011 par_len:7 par:00 94 00 00 00 02 10
10:16:42.352] [LIB]:(sdk)rcv access layer,retransaction:0,ttl:10,src:0x0001,dst:0x0093 sno:0x08c958 op:0x3d80(MODE_APP_BIND),par_len:6,par:94 00 00 00 06 13
10:16:42.368] [LIB]:(sdk)mesh tx NoAck,op:0x3e80(MODE_APP_STATUS),src:0x0093,dst:0x0001,sno:0x000012 par_len:7 par:00 94 00 00 00 06 13
```

device key

network key

iv index(big endianness)

app key

Figure 30.5: Provision UART log

30.3.2 Via Android App

Home page->setting->Mesh Info

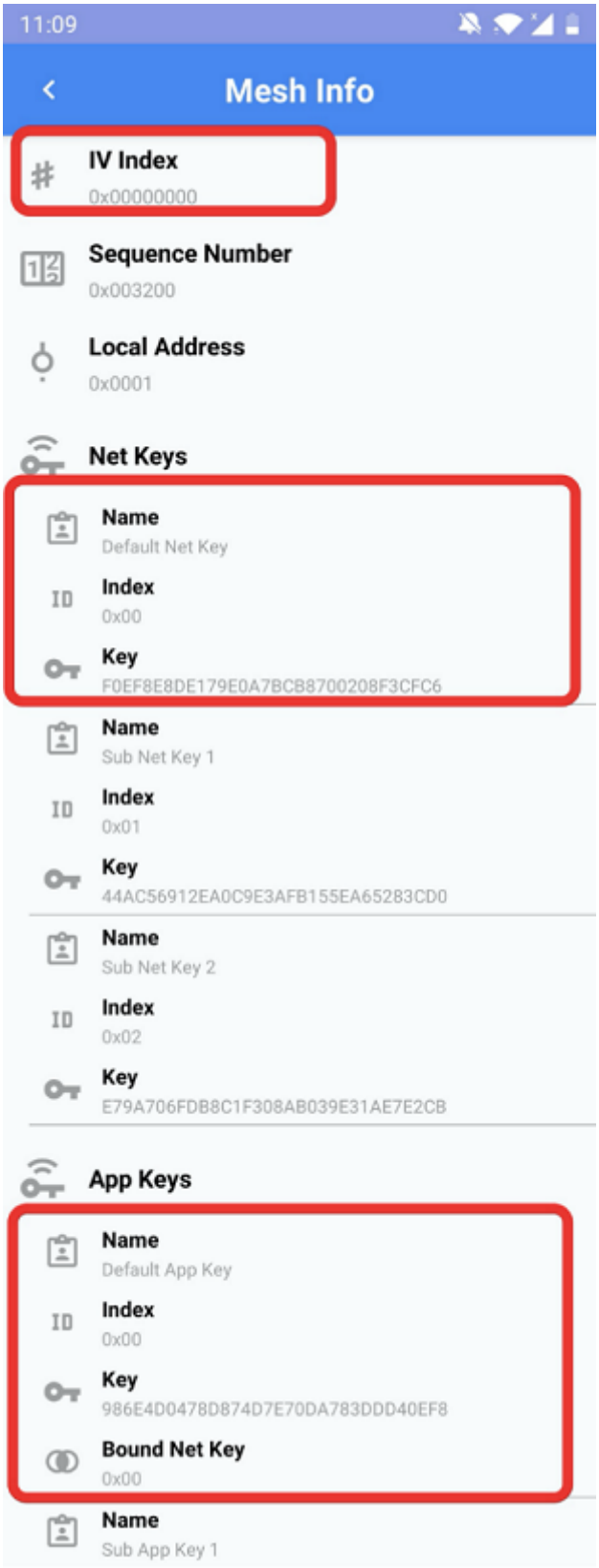


Figure 30.6: Android key

30.3.3 Via iOS App

Home page->setting->Mesh Info

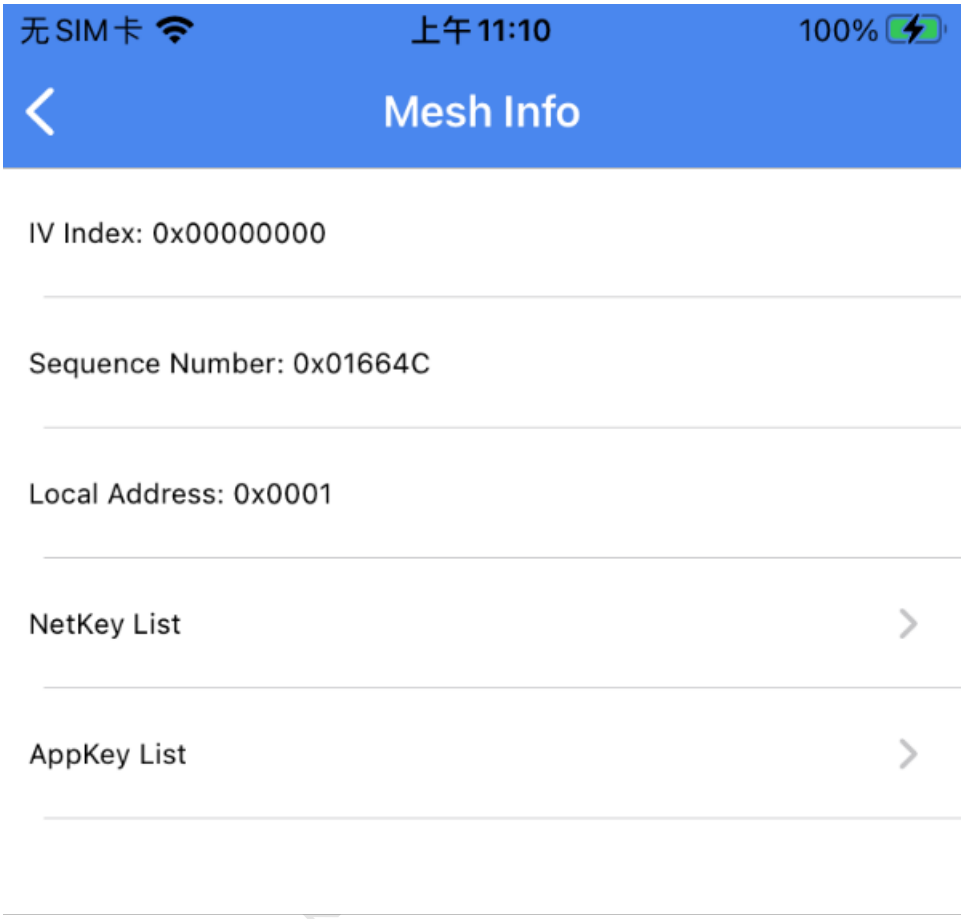


Figure 30.7: iOS key

30.3.4 Via JSON File

Open "mesh_database.json" inside the folder which contains "sig_mesh_tool.exe". Search "netKeys" and "appKeys" for net key and app key.

Figure 30.8: json key

Search "ivIndex" for iv index.

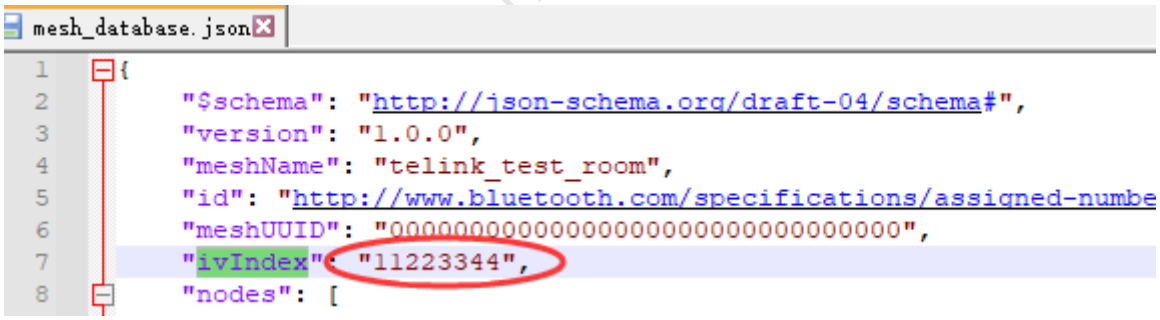


Figure 30.9: json iv

31 Operating Instructions for Telink-developed Bluetooth Mesh Decryption and Analysis Tool

31.1 Application Background

When debugging and developing Bluetooth Mesh products, in addition to opening the Log on the device side, sometimes it is also necessary to use a packet capture tool to analyze whether the format of the mesh messages in the air and the interaction flow is correct. Currently, the price of packet capturing instruments in the market is relatively expensive. In the absence of professional packet capturing tools, Telink-developed Bluetooth Mesh packet decryption analysis tool can be used for preliminary analysis.

This tool requires only one TLSR8258 Dongle (hereinafter referred to as Monitor) and one serial module, and then compile the 8258_mesh_monitor project(Download link of the SDK: http://wiki.telink-semi.cn/tools_and_sdk/BLE_Mesh/SIG_Mesh/sig_mesh_sdk.zip). Burn the compiled 8258_mesh_monitor.bin, and then monitor and decrypt the advertised mesh messages in the current mesh network, GATT proxy pdu and decrypted mesh messages encrypted with Device Key are not supported.

31.2 Operation Procedure

31.2.1 Configure Monitor serial port

The packet capture tool uses the 8258_mesh_monitor compilation option, and MESH_MONITOR_EN has been enabled in the project settings. It shares the application layer code with 8258_mesh project, and configures the IO of the serial port in the header file app_config_8258.h through the macro UART_TX_PIN.

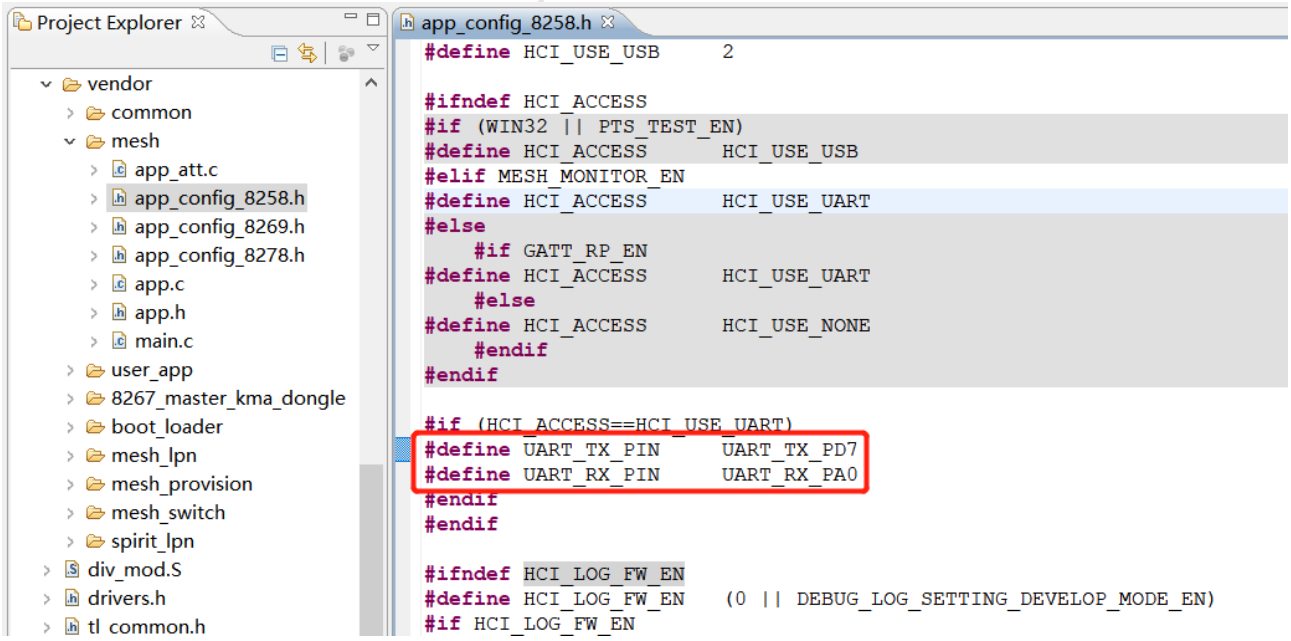


Figure 31.1: mesh_monitor_uart_io_setting

Clean the compilation to get the 8258_mesh_monitor.bin file, which is burned into the Monitor.

31.2.2 Connect the serial hardware

Connect Monitor's uart tx and uart rx to the rx and tx of the serial module respectively, open the serial port debugging assistant (general-purpose serial port tool can be used), select the corresponding COM port, set the baud rate to 115200, and hex display.

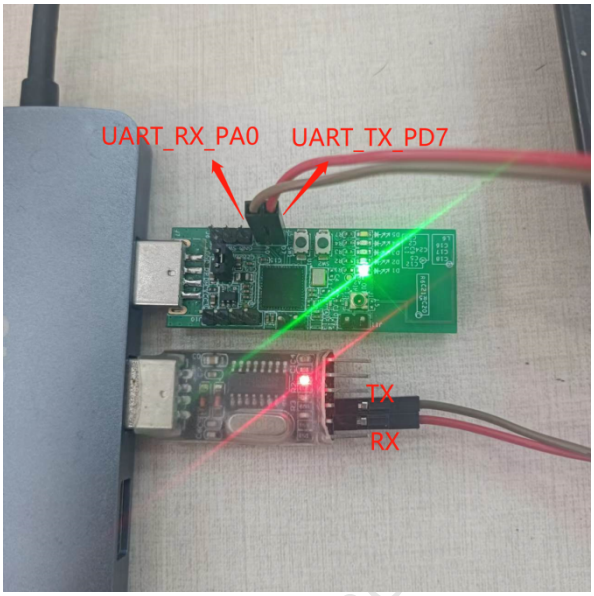


Figure 31.2: mesh_monitor_hardware_connection

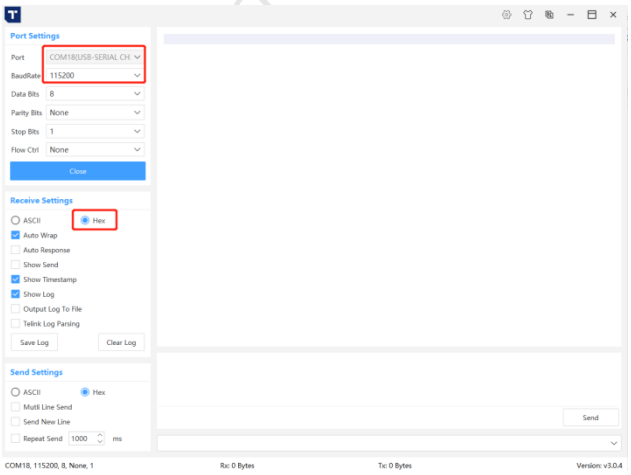


Figure 31.3: mesh_monitor_baudrate_setting

31.2.3 Add Monitor to a Mesh Network

Before joining the mesh network, Monitor is equivalent to an unprovisioned node and does not enable the monitoring function. After it is added to the mesh network with the app/gateway, it will automatically turn on the monitoring function.

31.2.4 Log Parsing

After the Monitor joins the mesh network, it decrypts and outputs the advertise type mesh messages within the mesh network that it monitors through the serial port, in the format:

0xF5 + rf_head + rf_length + advA + mesh_payload_length(type+PDU) + 0x2A(Mesh type) + Network PDU + RSSI + Frequency Offset + Channel

- rf_len: length of all data after it (excluding rssi, Frequency Offset, channel).
- advA: mac of the sending node.
- mesh_payload_length: data length of the mesh type and PDUs.
- RSSI: energy value in dBm. Note: data type is s8.
- FREQUENCY: Frequency deviation value. Unit is kHz. Note: data type is s16.
- CHANNEL: Indicates on which advertise channel the packet was sent.

The Network PDU corresponds to the member variables in the red box in the structure mesh_cmd_bear_t in the figure below, which is consistent with the PDU format defined in the Mesh specification, and the specific mesh message is known by parsing the reported raw data according to the format.

app_mesh.h | app.c

```
typedef struct{
    bear_head_t tx_head;    // parameters for tx bear that can be read in app_advertise_prepare_
    u8 trans_par_val;   // report_type
    u8 len;
    u8 type;    // adv type
    union{
        mesh_cmd_nw_t nw;
        struct{
            u8 nw_pre[9];    // 9 = OFFSETOF(mesh_cmd_nw_t, data)
            union{
                // message
                mesh_cmd_lt_unseg_t lt_unseg;
                mesh_cmd_lt_seg_t lt_seg;
                struct{
                    u8 lt_pre_unseg[1];    // 1 = OFFSETOF(mesh_cmd_lt_unseg_t, data)
                    mesh_cmd_ut_unseg_t ut_unseg;
                };
                struct{
                    u8 lt_pre_seg[4];   // 4 = OFFSETOF(mesh_cmd_lt_seg_t, data)
                    mesh_cmd_ut_seg_t ut_seg;
                };
                // control
                mesh_cmd_lt_ctl_unseg_t lt_ctl_unseg;
                mesh_cmd_lt_ctl_seg_t lt_ctl_seg;
                mesh_cmd_lt_ctl_seg_ack_t lt_ctl_ack;
            };
        };
        mesh_beacon_t beacon;
        online_st_adv_t online_st_adv;
    };
}mesh_cmd_bear_t;
```

Figure 31.4: mesh_monitor_report_format

As shown in the following figure, after adding a lighting node and Monitor, the gateway sends onoff set and customized vendor set messages to the mesh network, respectively, and the lighting node receives them and replies with onoff status and vendor status.

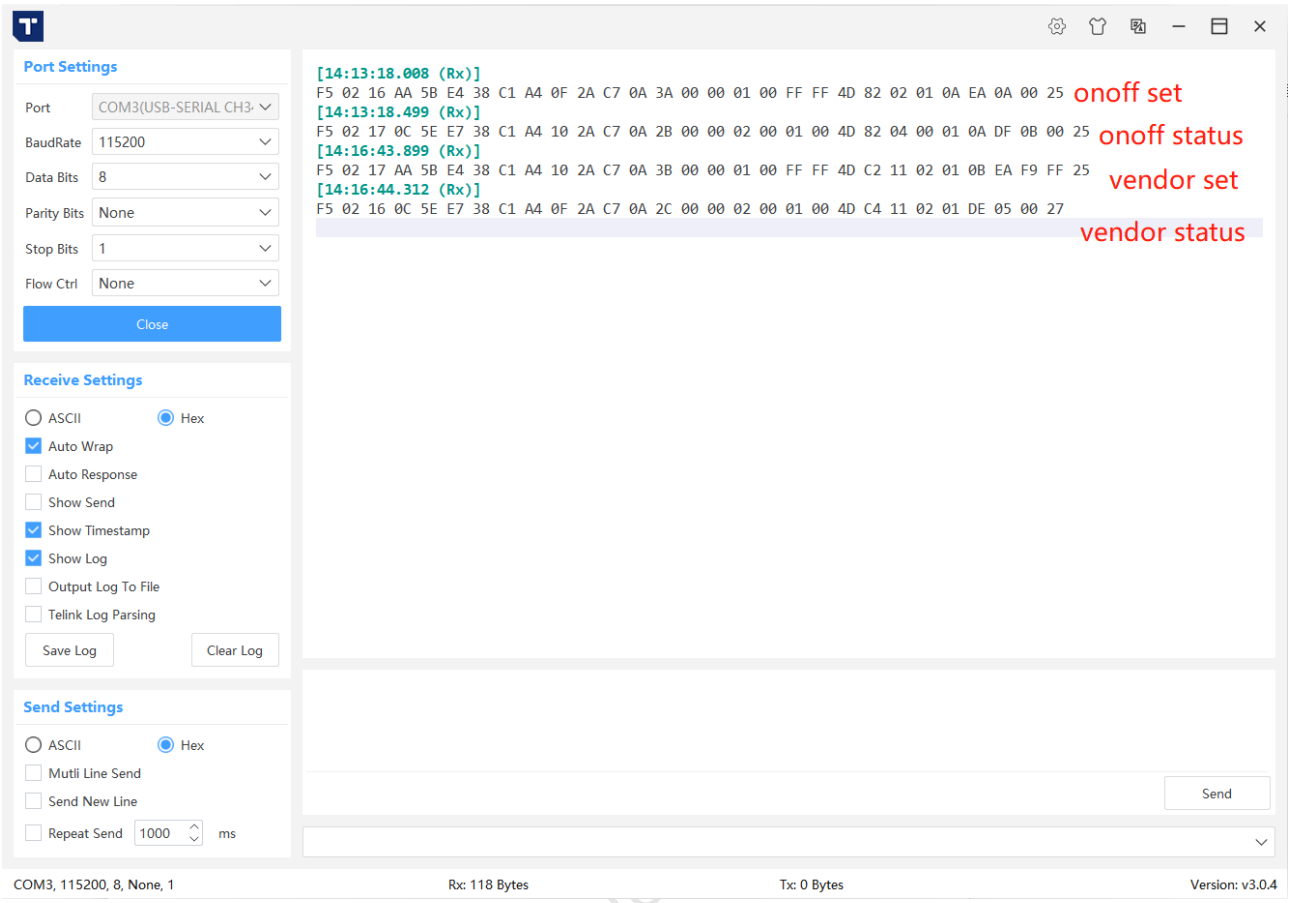


Figure 31.5: mesh_monitor_test_demo

31.2.5 Extended Functions

(1) If monitoring the mesh packets of the default network (the network before performing the mesh operation), you can send a command to the Monitor through the serial port to enable the monitoring function.

a8 ff + 00 + monitor_en。

- monitor_en is 1 for on, 0 for off.
- Setting success returns a8 ff + 00 + 00.
- Setting failure returns a8 ff + 00 + 01 (01 means error code is 1).

(2) Monitor enables sno filter (i.e. relay protect list) by default, if you need to disable it, you can set it by sending command to Monitor through serial port.

a8 ff + 01 + sno_filter.

- A sno_filter of 1 turns on sno filtering and 0 turns off sno filtering.
- Setting success returns a8 ff + 01 + 00.
- Setting failure returns a8 ff + 01 + 01 (01 means error code is 1).

(3) Monitor can specify the packet capture channel, which can be set by the following commands

a8 ff + 02 + chn1 + chn2 + chn3

- chn of 25, 26, 27 are advertise channels. Here chn as parameters can be 1, 2, 3 respectively. For example: (a) a8 ff 02 25 (b) a8 ff 02 25 26 (c) a8 ff 02 25 26 27. Note: 25, 26, and 27 are in hexadecimal, which corresponds to 37, 38, and 39 in decimal.
- Setting success returns a8 ff + 02 +00.
- Setting failure returns a8 ff + 02 + 01 (01 means error code is 1)

(4) Monitor provides whether to capture unprovision beacon advertise packets that are not in the network, which can be commanded through the serial port.

a8 ff + 03 + unprovision_beacon_enable

- unprovision_beacon_enable is 1 to turn on packet capture, 0 to turn it off.
- Setting success returns a8 ff + 03 + 00.
- Setting failure returns a8 ff + 03 + 01 (01 means error code is 1).

These are the steps of Bluetooth Mesh Packet Decryption and Analysis Tool, which can quickly locate and analyze the problem by capturing the over-the-air packets.

32 Spirit LPN

32.1 Function Description

The Spirit LPN is a privately defined low-power node that does not need to establish a friendship, but is implemented by the node periodically waking up for a period of time to receive packets. The default of the demo SDK is to wake up every 360ms, and each wake up lasts for 20ms to perform a broadcast scan and receive commands. Since the node is not receiving packets all the time, the sender (gateway) needs to send commands continuously to ensure the success rate.The duration of continuously sending commands should be greater than the wake-up cycle and scanning cycle of low-power nodes. For example, the demo SDK should have a minimum of 360+20=380ms. and it is recommended to exceed 1000ms.

32.2 Configuration

32.2.1 Set Gateway to Continuous Packet Sending Mode

32.2.1.1 Enable Key Detection

For testing purposes, the spirit LPN is switched on and off by pressing the SW2 button of the gateway, and UI_KEYBOARD_ENABLE is turned on in the 8258_mesh_provision project to enable key detection.

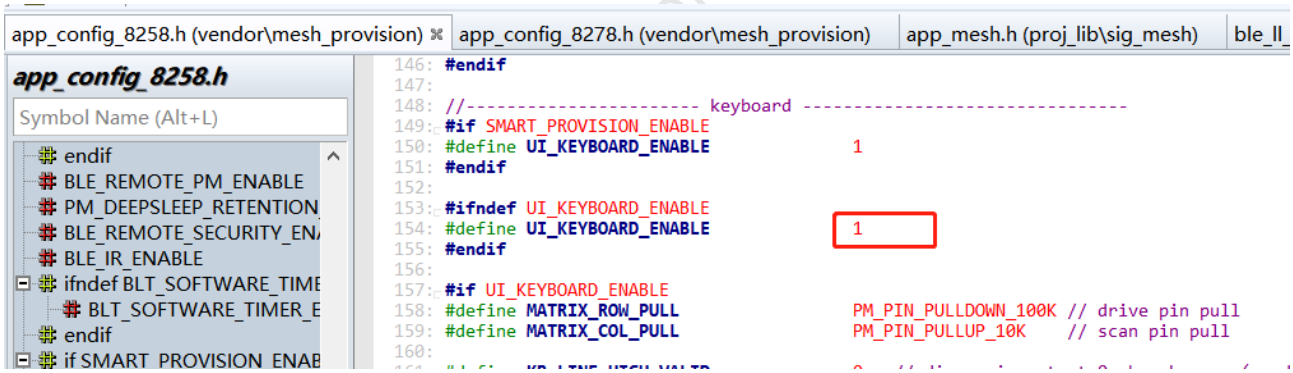


Figure 32.1: Open key

Change "#if 0" to "#if 1" in the mesh_proc_keyboard() function.

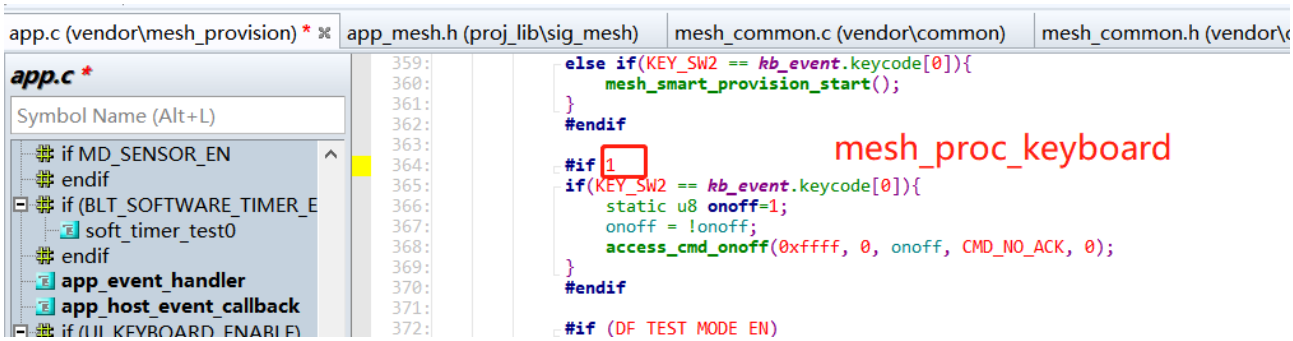


Figure 32.2: Open demo

32.2.1.2 Configure the Numbers of Gateway Sending Packets Continuously

generic_model.c (vendor\common) | mesh_access_layer.c (vendor\common) | mesh_common.c (vendor\common) | mesh_comm

```
3445:             p_tx_head = &tx_head;
3446:         }
3447:         p_tx_head->par_type = BEAR_TX_PAR_TYPE_REMAINING_TIMES;
3448:         p_tx_head->val[0] = 255;// extend count, interval 10ms
3449:     }
3450: #elif (1 && GATEWAY_ENABLE) // for spirit lpn test
3451:     if(0 == p_tx_head)
3452:     {
3453:         bear_head_t tx_head;
3454:         p_tx_head = &tx_head;
3455:     }
3456:     p_tx_head->par_type = BEAR_TX_PAR_TYPE_REMAINING_TIMES;
3457:     p_tx_head->val[0] = 100;// extend count, interval 10ms
3458:
3459:     mesh_op_resource_t op_res;
3460:     if(0 == mesh_search_model_id_by_op(&op_res, op, 1)){
3461:         if(is_cfg_model(op_res.id, 1) && ((COMPOSITION_DATA_GET == op) || (APPKEY_ADD == op) || (MODE_APP_BIND
3462:             p_tx_head->par_type = BEAR_TX_PAR_TYPE_NONE;
3463:         }
3464:     }
3465: #endif
3466: #if DU_ENABLE
3467:     if(0 == p_tx_head)
3468:     {
3469:         bear_head_t tx_head;
3470:         p_tx_head = &tx_head;
```

set_material_tx_cmd

Figure 32.3: Setting the number of packets sent

Set the number of extra packets in set_material_tx_cmd() function, the interval of extra packets is 10~12ms.Among 10~12ms, 2ms is the running time deviation of the mainloop cycle. Change "#if 0" to "#if 1", the packets will be sent continuously for 100 times with 10~12ms interval in demo. Of course, you can also set different values according to the demand, i.e. change the value of p_tx_head->val[0].

32.2.2 Setting the Wake-up Period and Scan Window for LPN

By default, the wake-up period of spirit LPN node is 360ms, and the scanning window is 20ms, if you want to change the wake-up period and scanning window time, you can follow the below modification.

32.2.2.1 Setting the Wake-up Period

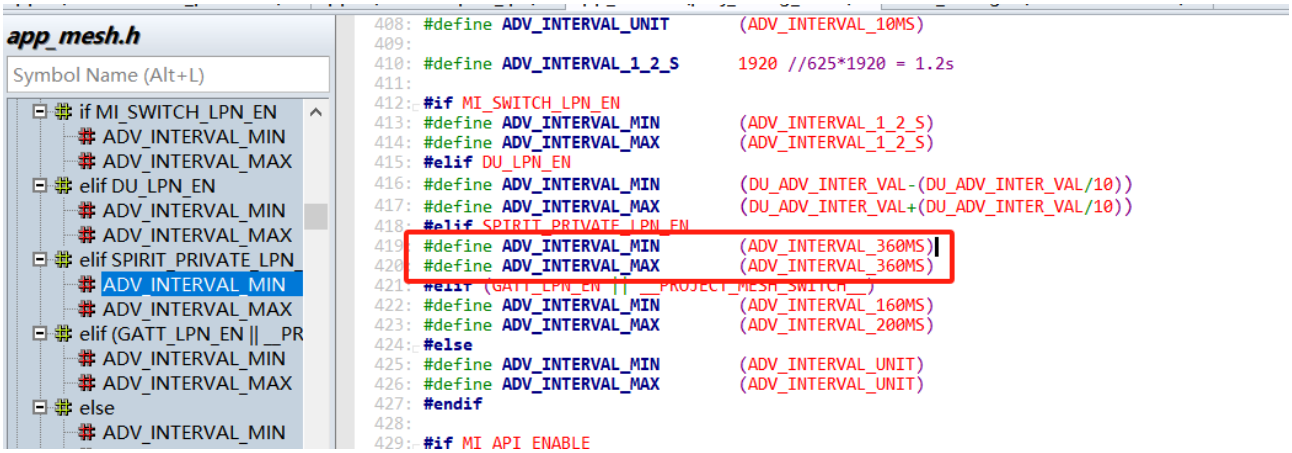


Figure 32.4: Setting the wake-up period

That is, change the value of ADV_INTERVAL_MIN, ADV_INTERVAL_MAX.

32.2.2.2 Setting the Scanning Window after Wake-up

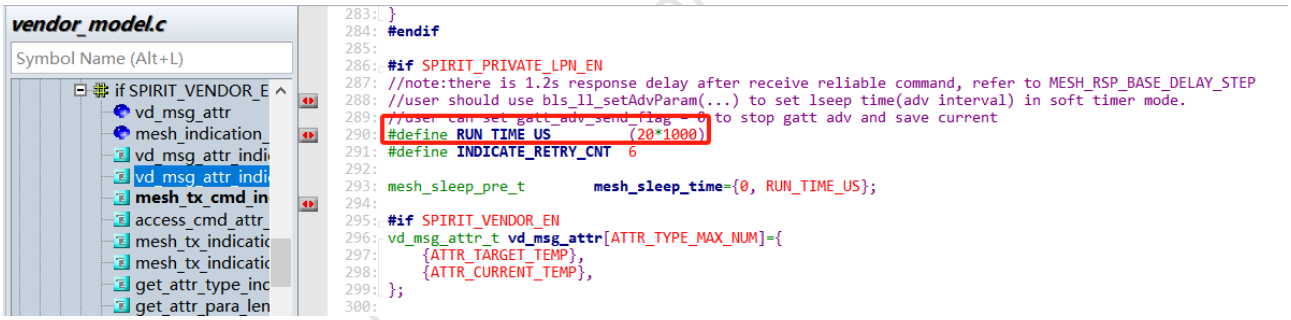


Figure 32.5: Setting the scan time

That is, change the value of RUN_TIME_US.

32.3 Function Demonstration

- Enable UI_KEYBOARD_ENABLE for 8258_mesh_provision project, then Compile 8258_mesh_spirit_lpn and 8258_mesh_gw project, burn the compiled fw into 8258 dongle via BDT respectively. The gateway is plugged into the usb port and the spirit lpn is networked via sig_mesh_tool.exe.
- After the networking is completed, pressing gateway SW2 can control the onoff state of spirit_lpn node. Every time it is pressed, the onoff status of spirit_lpn node will be toggled.

32.4 Platform Access Setting

The spirit LPN is in generic mode by default, and can be used by the demo SDK's Gateway and App for networking.

If you are accessing Tmall Genie mode, set MESH_USER_DEFINE_MODE to MESH_SPIRIT_ENABLE, please refer to Connect with a Platform section.

33 Android and iOS APP User Guide

33.1 App download

The Android App can be obtained from the firmware SDK package, for example:

\telink_sig_mesh_sdk\app\android\TelinkBleMesh\TelinkBleMeshDemo-V4.1.0.0-20231113.apk.

The iOS app can be obtained by searching for telinksigmesh in the App Store.

Developers can also recompile the App, and the corresponding code can be obtained from the firmware SDK package: \telink_sig_mesh_sdk\app

33.2 Device Network

Networking is divided into manual networking mode and automatic networking mode.

33.2.1 Manual Provision Networking

33.2.1.1 Add Device in Manual Mode

The Android/iOS version of the APP enables manual provision mode by default. After launching the app, click the upper right corner of the "+" button of the main interface to enter the add interface, the APP will automatically search for peripheral devices as shown below. You can click the button on the right side of the corresponding device to network, or you can click the button on the top right corner of the corresponding device to delete the corresponding device, or click the ADD ALL button to network all the devices in the list.

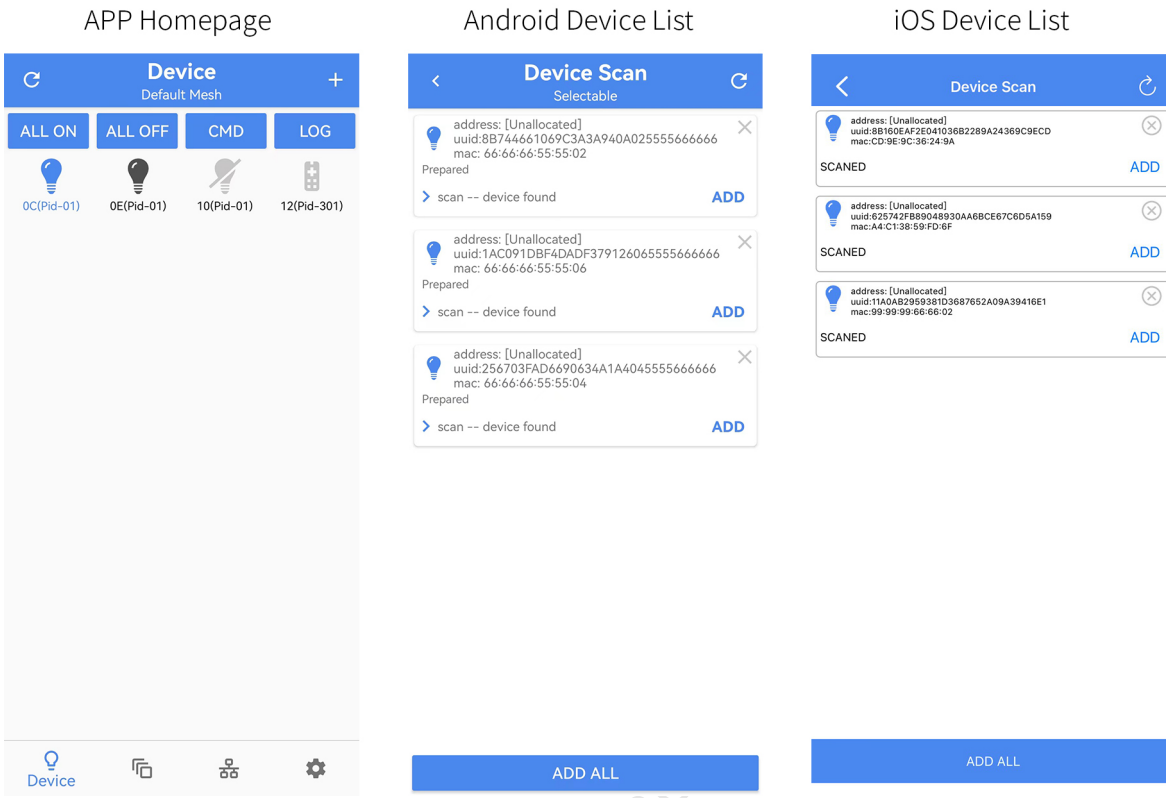


Figure 33.1: Manual mode adding devices

33.2.1.2 Status During Manually Adding Devices

Android APP

The Scan-device found is the status that the device is scanned and found, and the left arrow expands the status of each state during the networking process (as shown below).

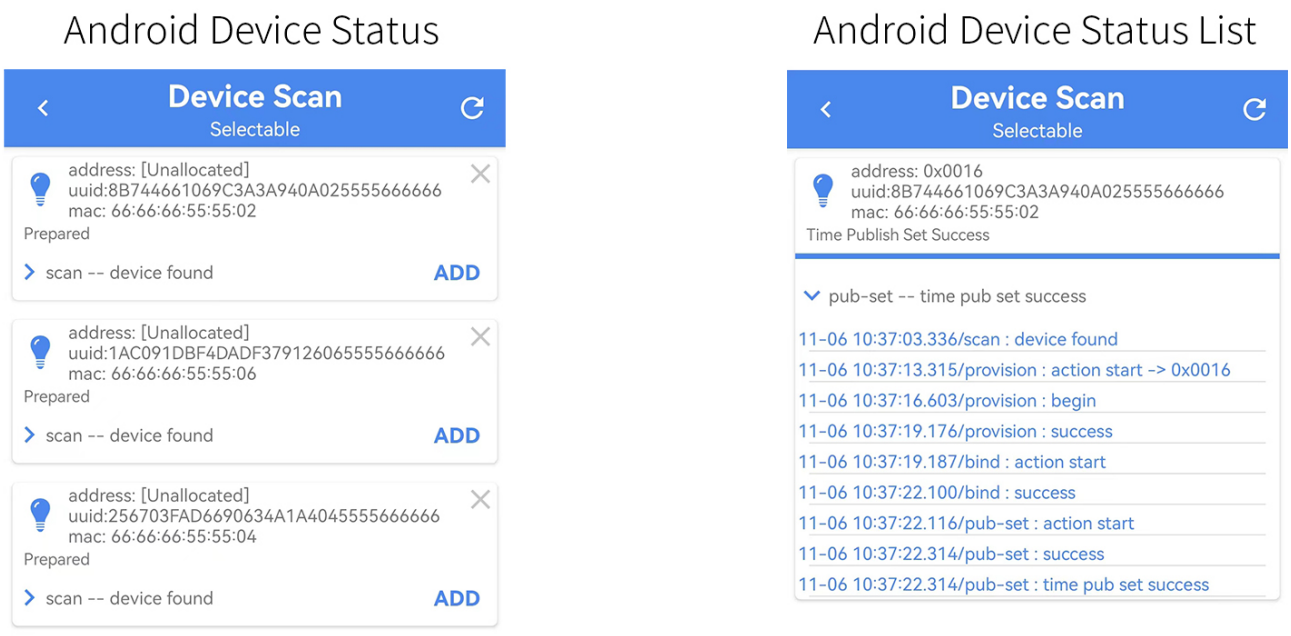


Figure 33.2: Status display for android manual mode add device process

iOS APP

The iOS version of the APP only displays the current status, more detailed information can be viewed in the APP's log record.

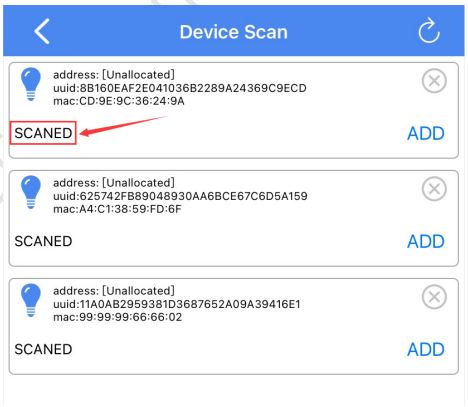


Figure 33.3: iOS device status

33.2.2 Auto Provision Networking

From APP home page - setting - settings in the Provision Mode, select Normal (Auto) to switch to Auto provision mode. At home page click the upper right corner of the + sign to enter the Device Scan interface, at this time the title will show Device Scan (Auto provision) and it automatically adds all the surrounding un-networked devices as the figure below. If scanning timeout, the Android app will pop-up the return button in the upper-left corner, the iOS version of the APP will pop-up Go Back button at the bottom of the app.

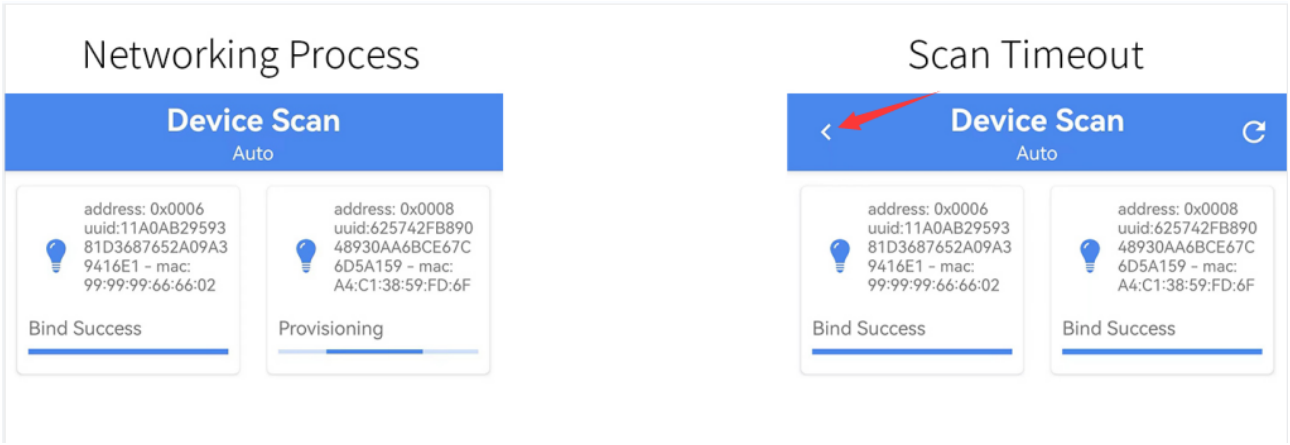


Figure 33.4: Android auto provision

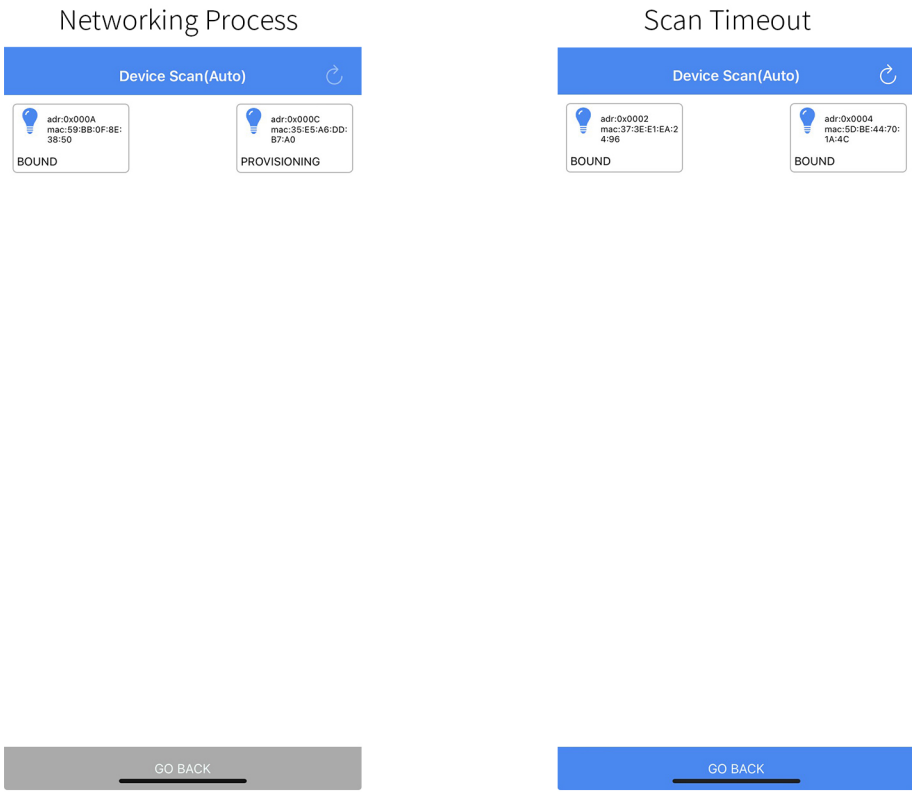


Figure 33.5: iOS auto provision

33.2.3 Rescan Peripheral Devices

The icon at the upper right corner of Device Scan interface is the reload button. The manual provision networking mode can reload the device list as the figure below. The auto provision mode can re-scan the peripheral devices and automatically network as the figure below.

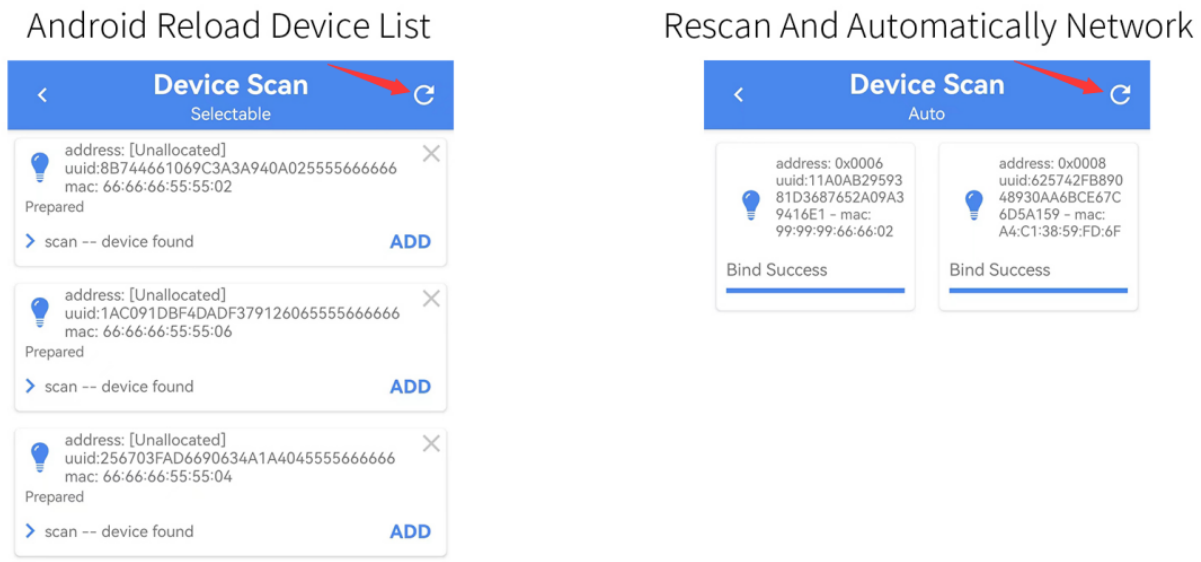


Figure 33.6: Android device reloads device list & rescan and auto-networking

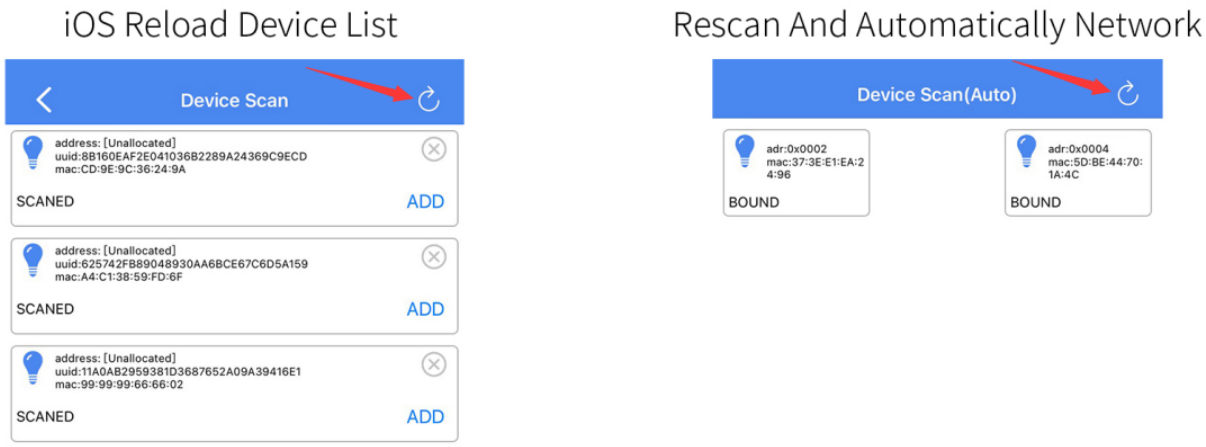


Figure 33.7: iOS device reloads device list & rescan & auto-networking

33.3 Device Interface

Device interface (APP home page) lists 4 states, directly connected node name is shown in blue, off state is shown in dark grey, offline state is shown in light grey with slash, and different Pid devices show different icons.

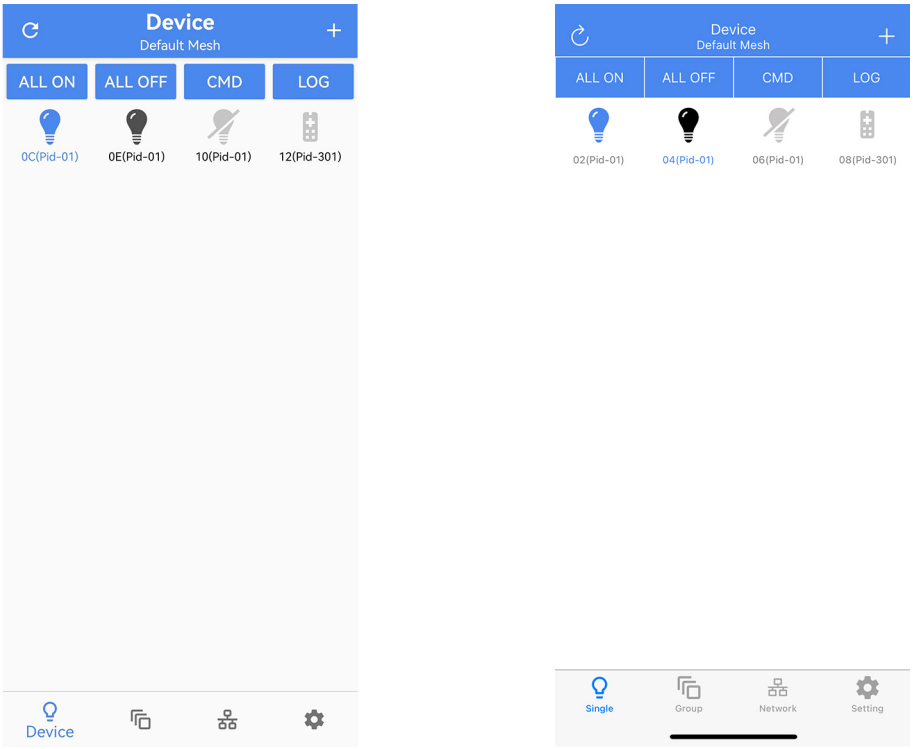


Figure 33.8: Android & iOS app device interface

33.3.1 Refresh Device

The icon  on the top left corner of the Android/iOS APP homepage can refresh the current networked device status.

33.3.2 All on/off

The Android/iOS APPs control all networked devices to turn on/off the lights by sending "all on/off" commands. Blue is on, dark grey is off.

33.3.3 Single Device on/off

Click on the corresponding device icon to turn on/off the light. Blue is on, dark grey is off.

33.3.4 CMD Command

The CMD command has built-in Vendor on/off, Vendor on/off no-ACK, Vendor on/off get, Generic on/off (For iOS, it is not built-in yet), Opcode Aggregator (Lightness Default/Range Get, TTL/Friend/Relay Get), Device/App Key Get command for iOS. Users can also customize the commands through APP Custom for Android (APP Vendor Data for iOS), which can customize the following commands.

- Access Type: (decide what key to use to send)

- dst adr：(destination address)
- opcode：operation code
- params：the parameters that follow the opcode
- rsp opcode：(response code for opcode)
- rsp max：(if rsp opcode is not 0, the value is the quantity of nodes are expected to receive replies, and if not enough replies are received, a retry will be performed), retry count (the maximum number of times a retry will be performed if the number of replies specified by rsp max is not received)
- ttl：time to life.
- tid position：this value must be 0 for non-vendor commands and indicates the position of the tid for vendor commands (a value of 0 indicates that there is no tid field, 1 indicates that the tid is at the para[0] position, and 2 indicates that it is at the para[1] position).

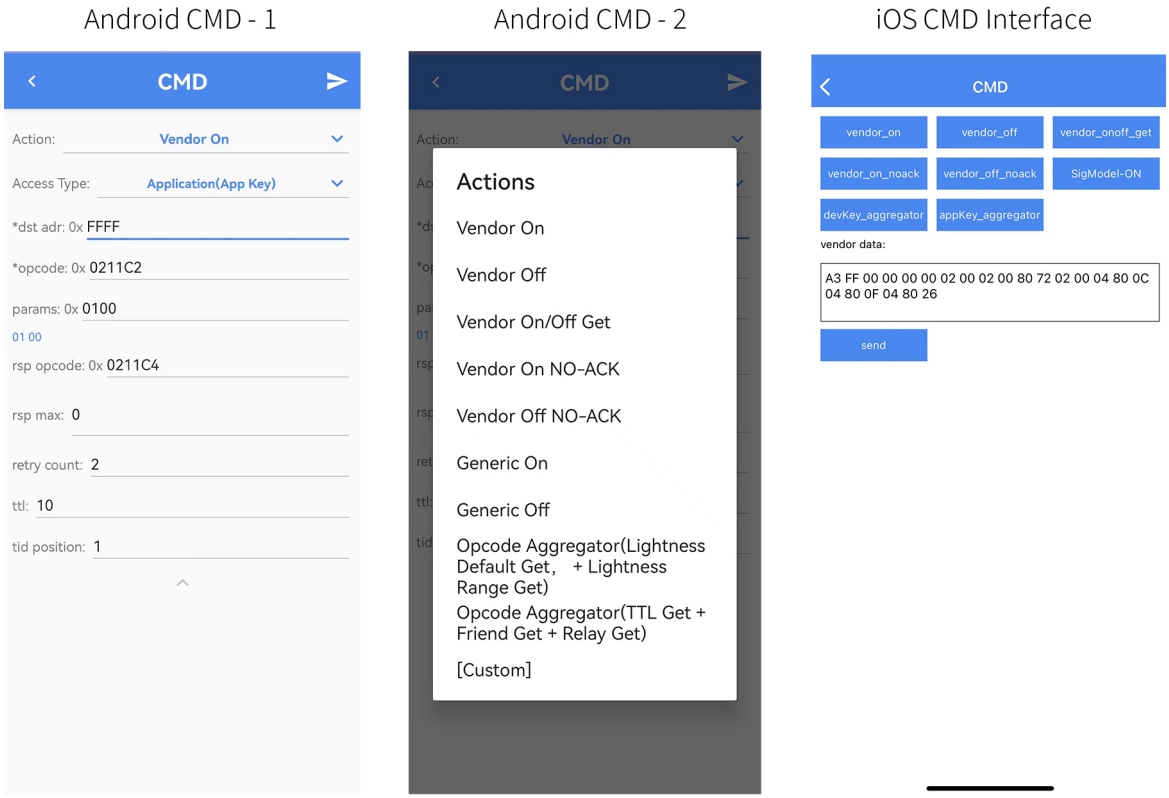


Figure 33.9: Android & iOS CMD interface

33.3.5 Log

The log function is enabled by default to record the log information of the current operation. For Android version, you can turn on/off Enable LOG in APP homepage – setting – settings. Click Save in File button to save the logs, save path: default storage/TelinkBleMesh. (Note: iOS automatically save, if you need to export the log to PC, you can use iTunes – File Sharing – TelinkBleMesh to export the log to PC.) Click the Refresh button to refresh the log (for iOS version, exit and re-enter the log interface), click Clear button to clear the log information.

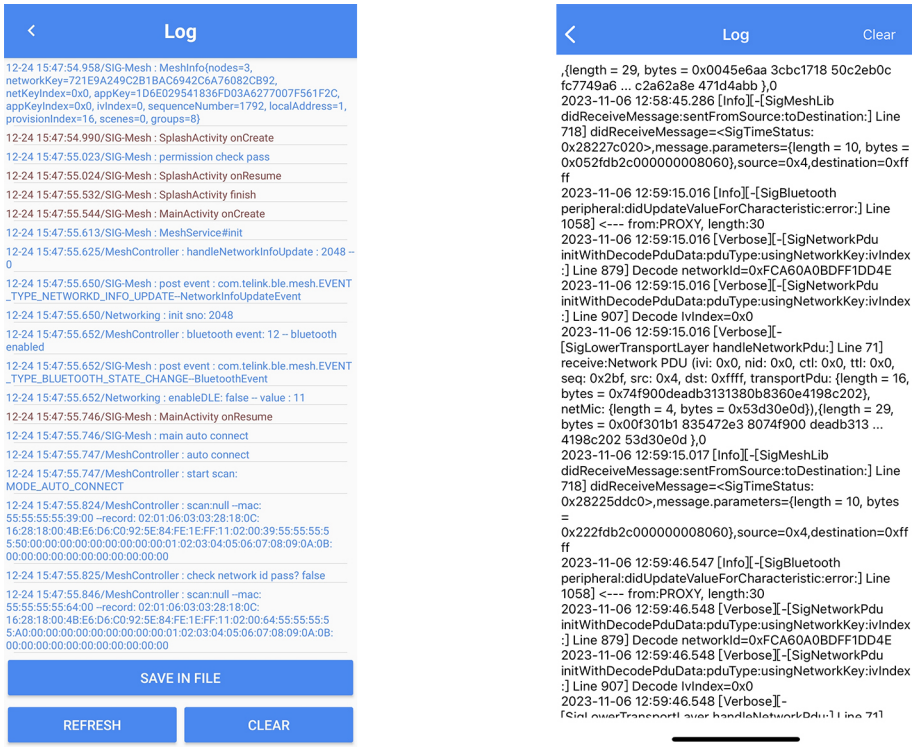


Figure 33.10: Android & iOS log interface

33.3.6 Device Setting (Light device)

Long press the icon of the networked Light device on the Android/iOS APP homepage to enter Device Setting interface.

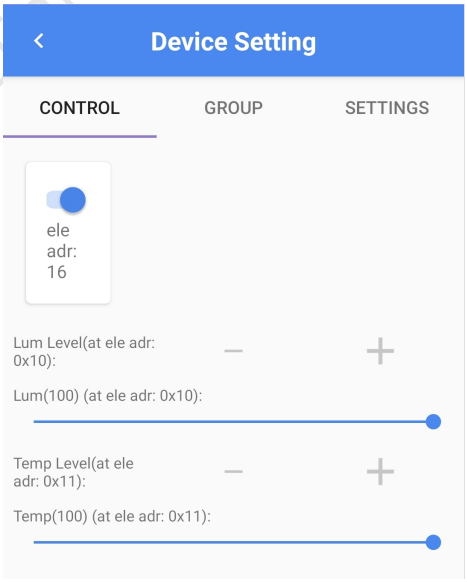


Figure 33.11: The light device setting interface

33.3.6.1 Light Device Control

The "Ele Adr X" is used to switch the device on and off; "Lum Level" is used to adjust the brightness of the device; "Temp Level" is used to adjust the colour temperature of the device.

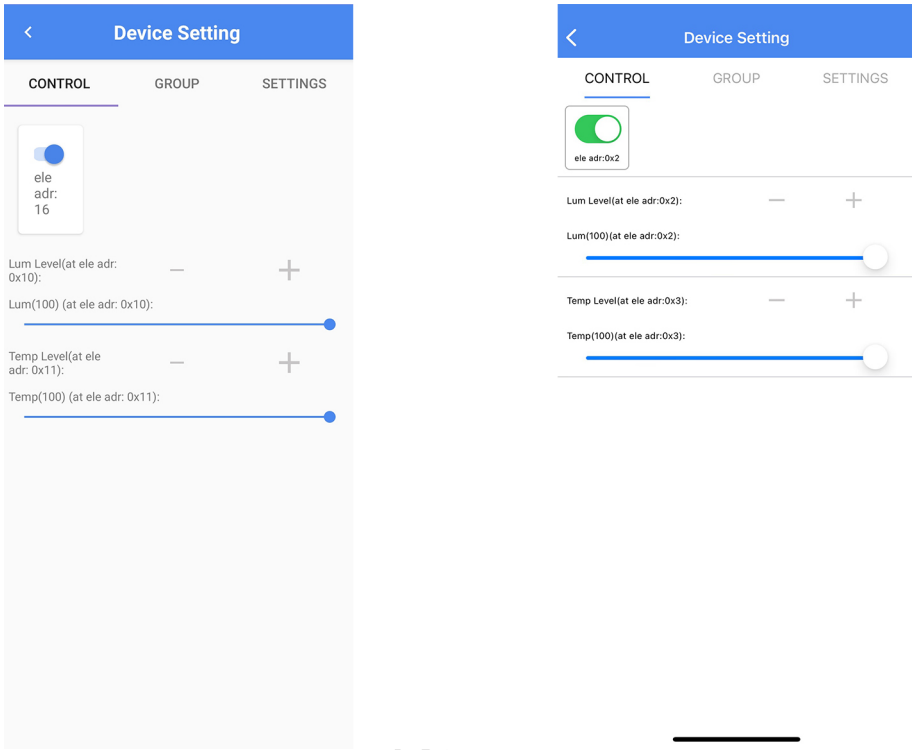


Figure 33.12: Android & iOS control interface

Light Device Network Lighting Control

When the Light device supports Light LC Server, the Network Lighting Control sub-page entry will appear under the Control page of the device. The Network Lighting Control page is shown in the following figure.

(1) The Light device needs to turn on the Enable LC mode and Enable LC Occupancy mode switches before it will process the sensor data reported by the sensor device and determine whether to execute the Light Control action.

(2) The Set LC light on/off button is used to send the `LightLCLightOnOffSet` command to the Light device.

(3) The following Properties list contains 3 Lightness parameters and 7 Time parameters, all of which are already configured by default on the device side. Users can get and set these parameters, for detailed description of each parameter, please refer to the Handbook document on the firmware side or the sig mesh protocol document.

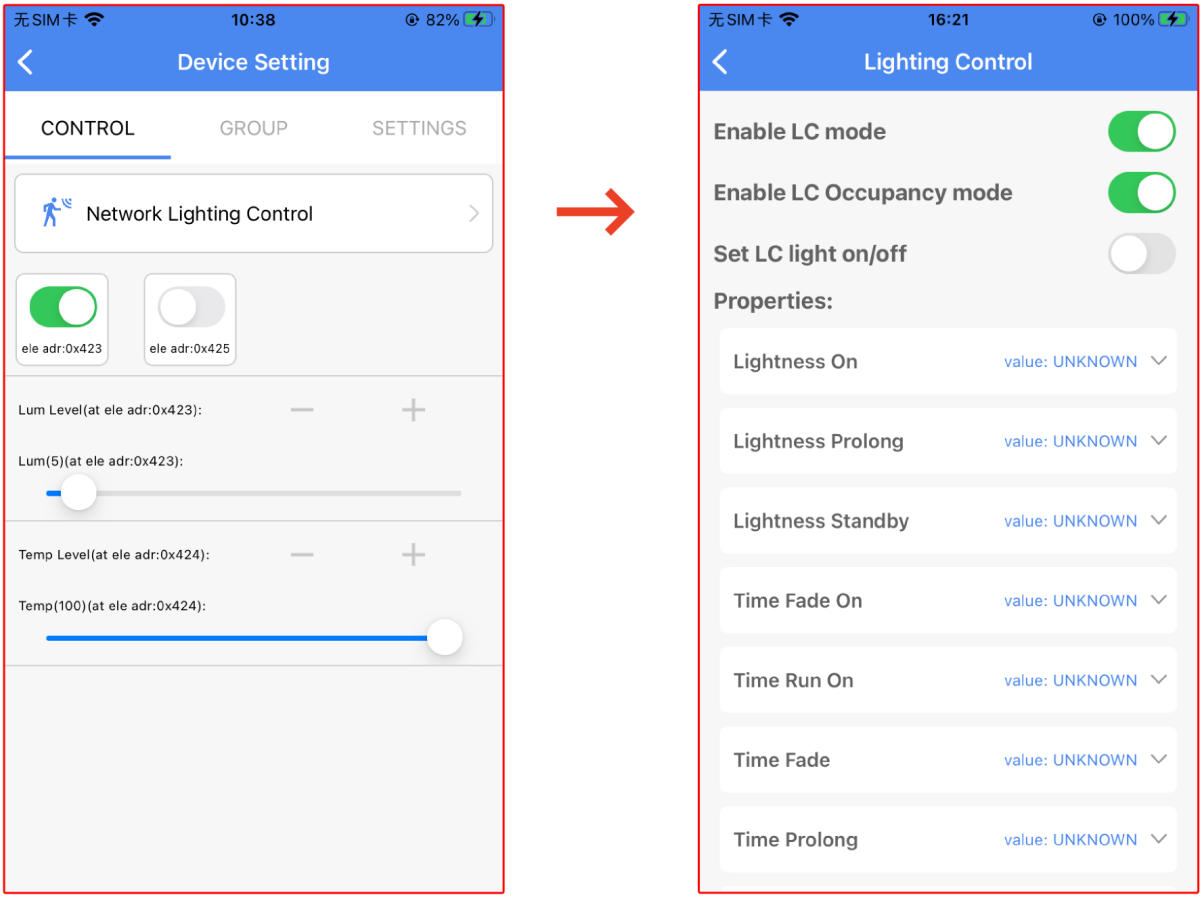


Figure 33.13: Light Device Lighting Control interface

Sensor Device Sensor Control

When the Sensor device supports Sensor Server, the Sensor Control sub-page entry will appear under the Control page of the device. The Sensor Control page is shown in the following figure.

(1) The publish adr is the address where sensor status data can be received. The publish adr of 0 means that the sensor does not report status data. The publish adr of 0xFFFF means that all devices can receive sensor status data. If a single device is required to receive status data from this sensor, it is needed to set publish adr to the element address where the Light LC Server for that device is located.

(2) The period is set to 0 by default, which means that Sensor Data will not be reported periodically, but data reporting will be done when there is a change in the value of Sensor Data.

(3) Sensor Data is the sensor data reported by the sensor via the `SensorStatus` command, Sensor Descriptor is the configuration parameter cured by the sensor reported by the sensor via the `SensorDescriptorStatus` command, and Sensor Cadence is the sensor's modifiable configuration parameters reported by the sensor via the `SensorCadenceStatus` command.

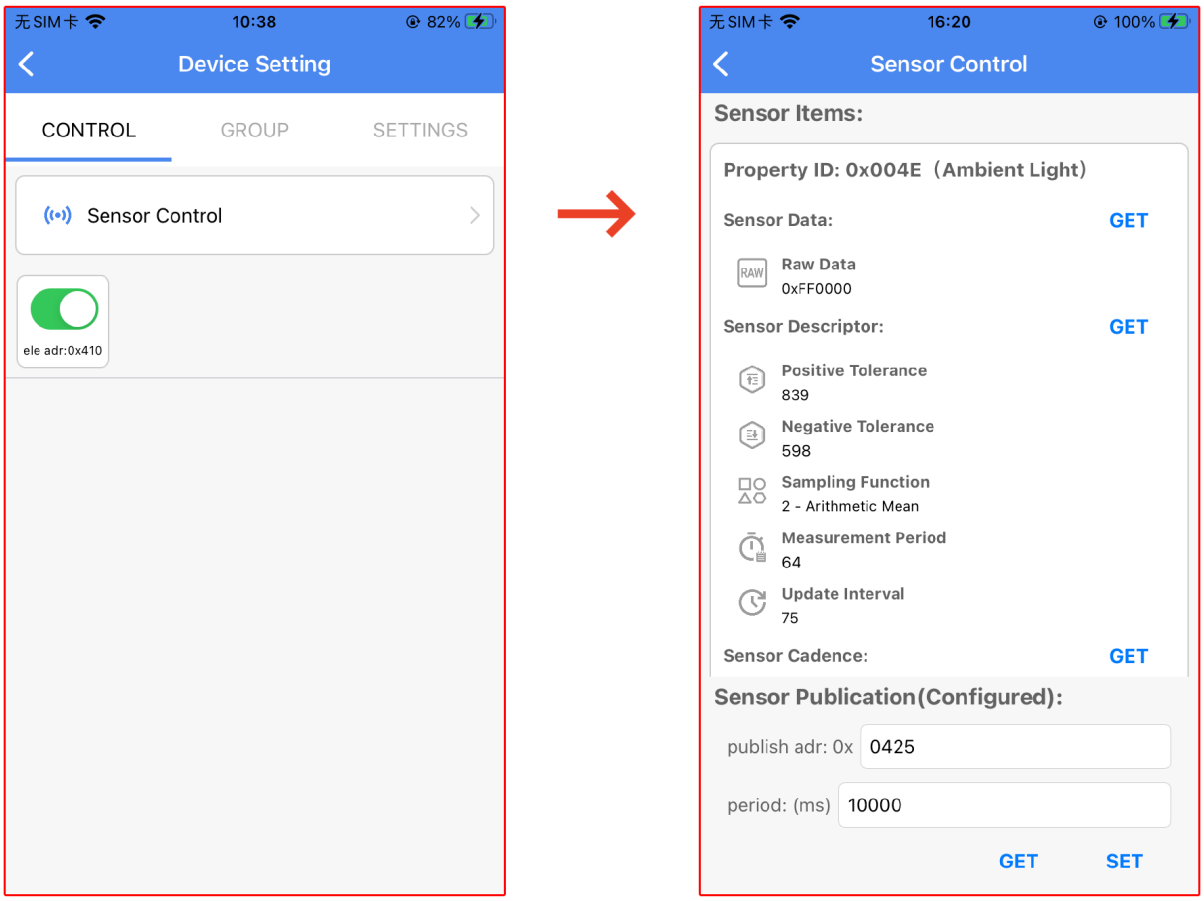


Figure 33.14: Sensor Device Sensor Control interface

33.3.6.2 Single Device Group

The "Group" is used to group the device (a single device supports a maximum of 8 groups).

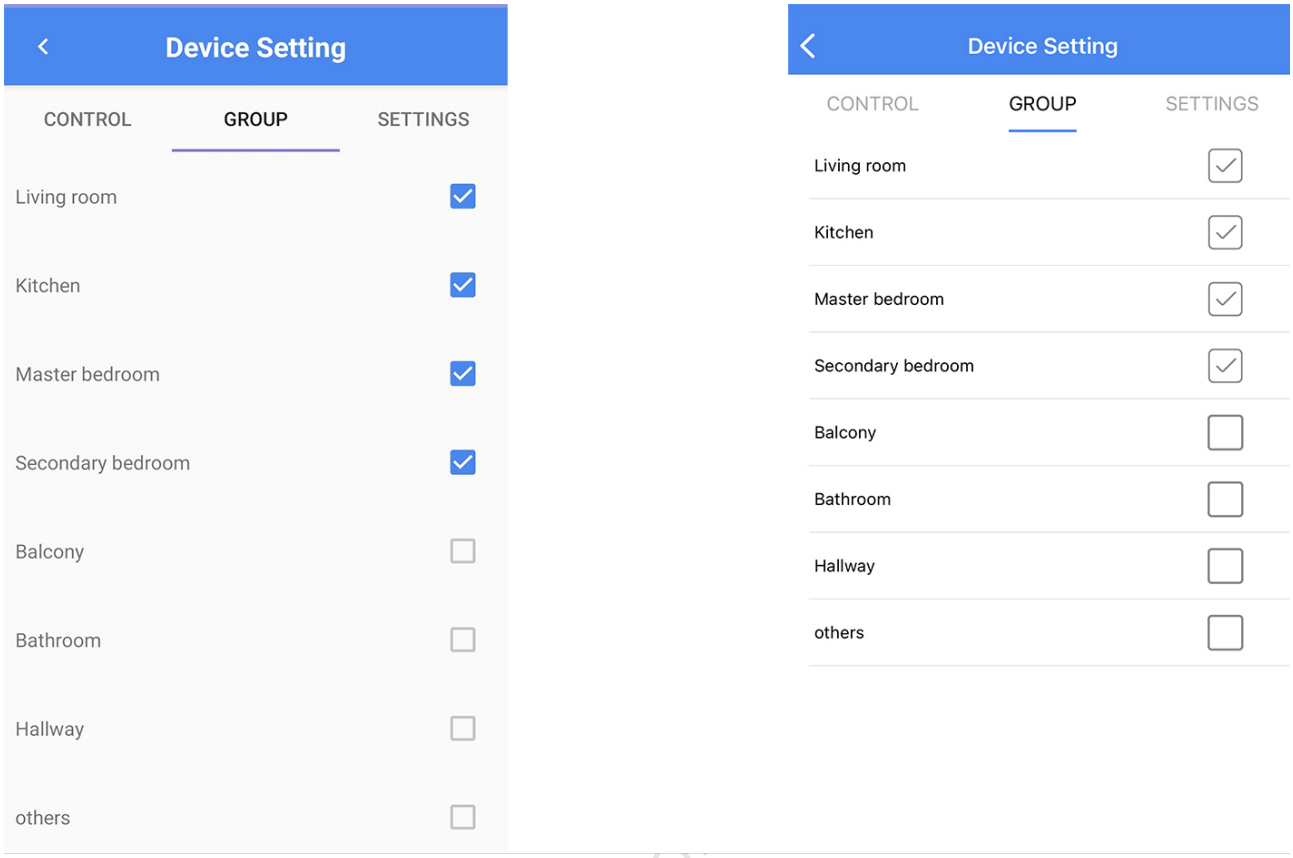


Figure 33.15: Android & iOS add group interface

33.3.6.3 Light Device Settings

The Settings menu enables user to view the UUID, and execute operations including Device Config, Composition Data, Network Keys, Subnet Bridge Setting, Schedule Setting, Subscription Models, Device Ota, Publication, and Kick Out.

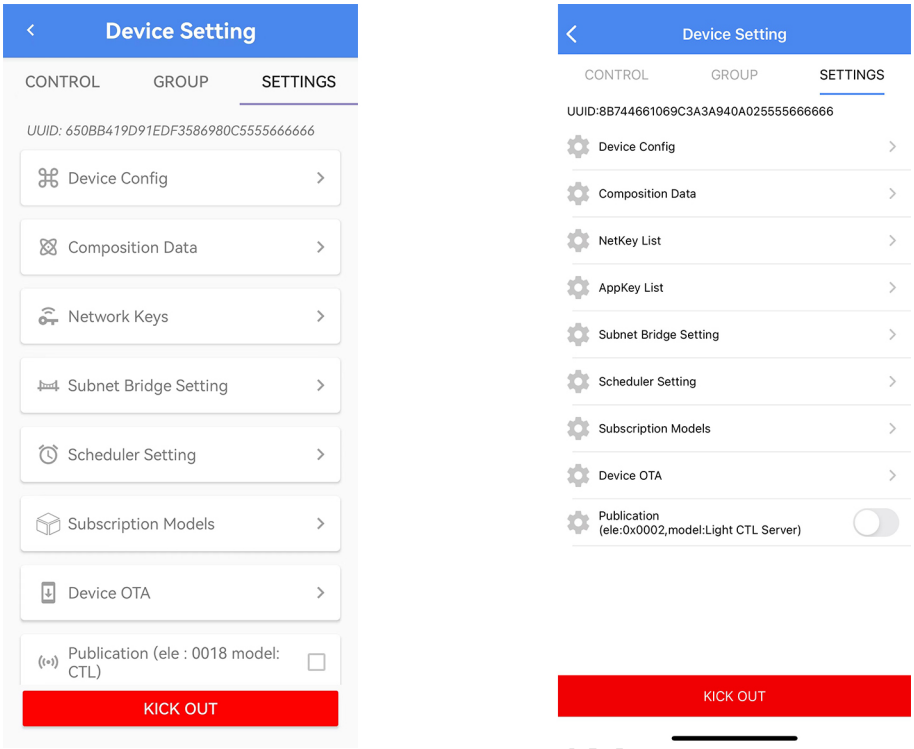


Figure 33.16: Android & iOS settings interface

Device Config

The Device Configuration is to configure the device's TTL, Relay and Relay Retransmission, Secure Network Beacon, Gatt Agent, Nedeidentity, Friend, Key Refresh Phase, Network Transmission.

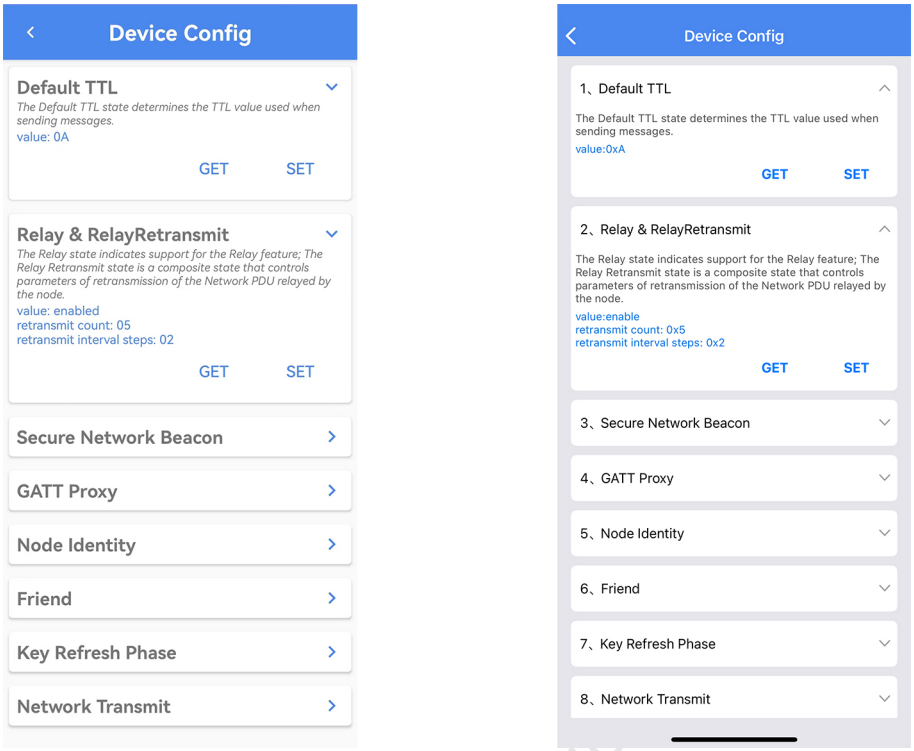


Figure 33.17: Android & iOS Device Config

Composition Data

The "Composition Data" is used to view the data of the device (including: cid/pid/vid/crpl/features/relay support/proxy support/freind support/low power support/position type of each sig model and vendor model). Clicking the icon on the top right corner can refresh the data.

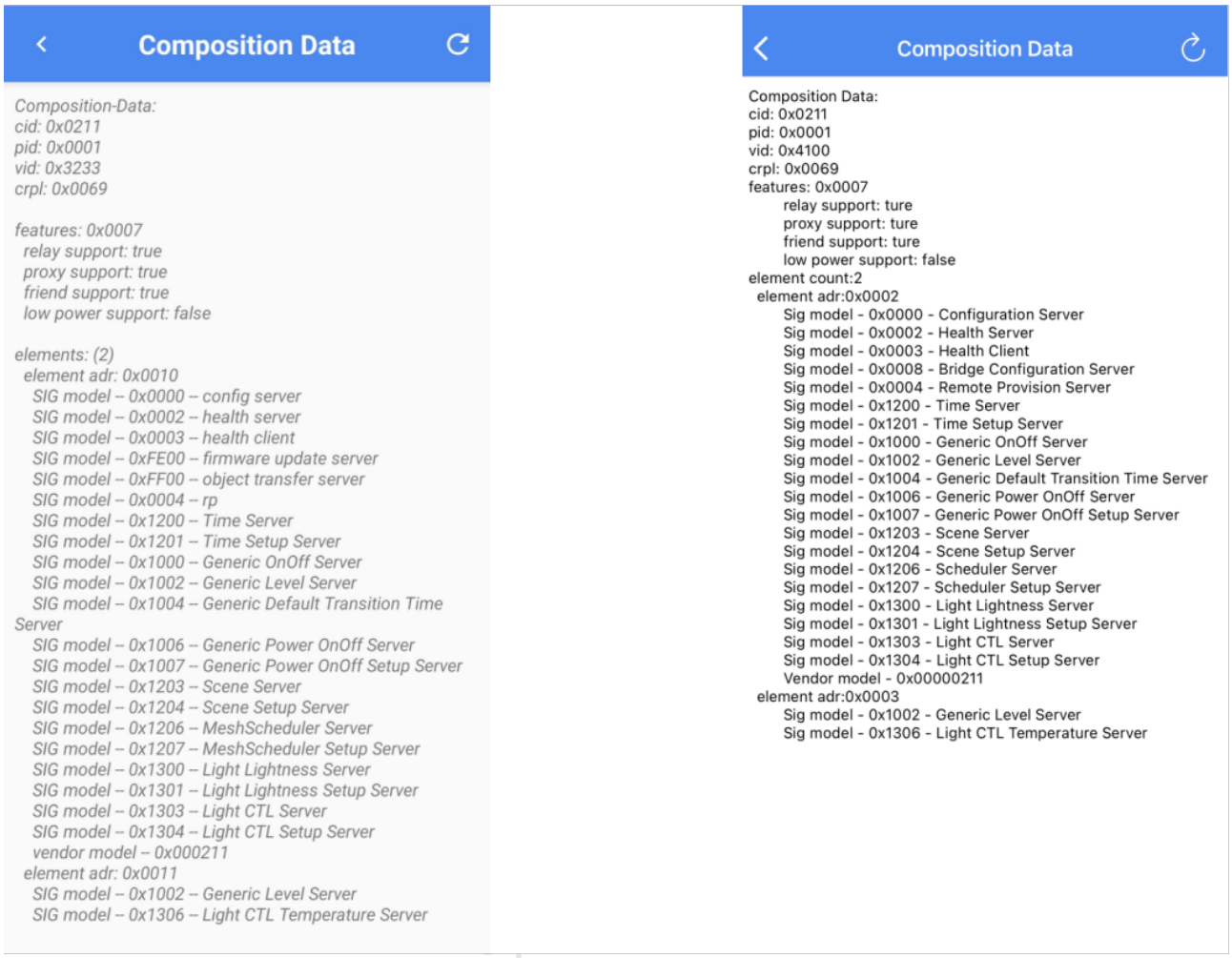


Figure 33.18: Android & iOS Composition Data

Networ Keys (iOS：NetKey List / AppKey List)

In a Network users can create different Network Keys (iOS：NetKey List / AppKey List). In Network Keys (iOS：NetKey List / AppKey List) you can view the key bound to the current device, you can also configure the specified node with a new Network Key in order to connect different keys to different devices, and also share the specified node out to become a shared device by means of the key. The detailed operation is as follows：

Preset conditions： prepare two mobile phones A and B; add more than 2 devices to mobile phone A for default network key.

Steps for Android：(Android phone as phone A)

(1) Mobile phone A creates a new Net Key for the specified device. The detailed operation：

Long press a device that needs to creat a new Net Key on the APP homepage – Settings – Network Keys – Click "+" on the upper right corner to select a Net key (Currently, there are two built-in Net / APP key, which can be viewed in APP home page – click the Network in the lower right – Mesh info).

(2) Mobile phone A shares device to mobile phone B by sharing Net Key：

Mobile phone A APP home page – Setting – Share – Export – select the newly created Net Key – export by file/QR code;

Mobile phone B APP home page – setting – Manage Network – Import button at the bottom right corner – Import by file/QR code.

At this time, the device with the newly configured Net key just now becomes a shared device, and mobile phone B can only get the status of the device corresponding to the Net key, and the other device shows offline status due to different Net key.

Steps for iOS: (iOS phone as phone A)

(1) Mobile phone A creates a new Net Key for the specified device. The detailed operation:

Click Network in the lower right corner of APP homepage – Mesh info – Netkey List – create a new Net Key;

Return to Mesh info interface – App Key List – create a new App Key (Note: the key is required to be the same as the currently existing App Key; index, BoundNetkey bind to the newly created Net Key);

Long press a device that needs to creat a new Net Key/App Key on the APP homepage – Settings – NetKeys List – Click "+" on the upper right corner to select a Net key – Done – return to Device Setting – select AppKey List – Click "+" on the upper right corner to select a App key – Done.

(2) Mobile phone A shares device to mobile phone B by sharing Net Key:

Mobile phone A APP home page – Setting – Manage Network – Click the Network just configured Network Key – Share Export – Select the new Network Key just created – Export by file/QR code–Export;

Mobile phone B APP home page – setting – Manage Network – Import button at the bottom right corner – Import by file/QR code.

At this time, the device with the newly configured Net key just now becomes a shared device, and mobile phone B can obtain and control the status of the device corresponding to this Net key, and the other device displays offline status due to the difference in Net key.

Subnet Bridge Setting

The Subnet Bridge feature allows bridging tables to be configured to multiple subnet (Network Keys) nodes within a Network, allowing messages to be forwarded to specific subnets. For example, if node 1 is configured with shared devices for Netkey1 and Netkey2, and node 2 is configured with private devices for Netkey1 only, and Netkey2 wants to control the private devices of Netkey1, then it needs to configure a bridge table from network Netkey2 to Netkey1 by configuring shared node 1.

Mobile phone A operation (initially with network sharing devices): APP home page long press a shared device (configured Netkey1 and Netkey2, Netkey2 has been shared to mobile phone B) – setting – Subnet Bridge Setting – Turn on Enable Subnet Bridge switch – Click ADD Subnet Bridge button. – Add Bridging Table interface, for Net key 1 fill in the shared Net key; for Net Key 2 fill in the Net key that needs to be converted (Note: that is, the Net key of the original network sharing device) – For Address 1 Fill in the Local Address of the Net Key that has been shared (Viewing steps: mobile phone B that imports Net Key through sharing: APP home page – Network – Mesh info). – For Address 2 fill in the short address of the device to be controlled – Add Bridge Table to save.

Mobile phone B operation (get netkey through sharing): long press the node specified by mobile phone A to enter Device Setting interface to switch the control node.

Scheduler

Click "+" on the top right corner of the Scheduler list interface to add scheduler. After setting the conditions in the scheduler setting interface, click to get and set the time (click setTime for iOS). Then click to save scheduler (Note: Scheduler is turned off by default, the device needs to enable MD_TIME_EN macro).

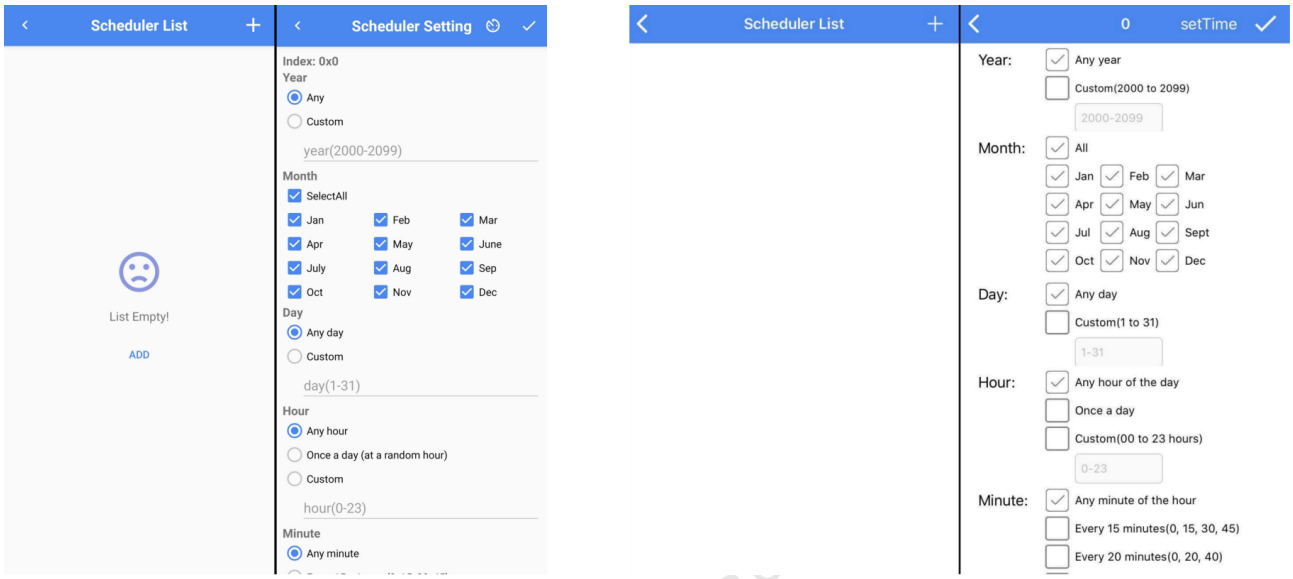


Figure 33.19: Android & iOS Scheduler

The Schedulers added in the Scheduler list interface can also be edited by clicking . After setting the conditions in the scheduler setting interface, click to get and set the time (click setTime for iOS). Then click to save scheduler.

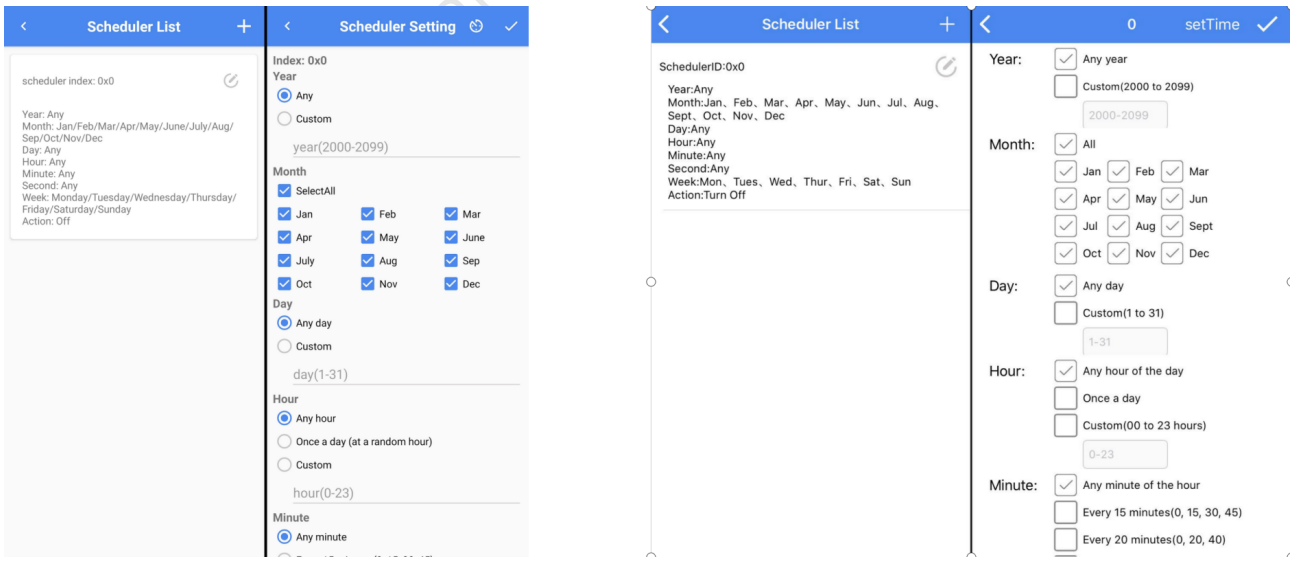


Figure 33.20: Android & iOS edit Scheduler

Subscription Models

Subscription models can be viewed for the currently supported Models for the device:

ID: 0x1000 (model name: Generic onoff server)

ID: 0x1300 (model name: Light Lightness server)

ID: 0x1303 (model name: Light CTL server)

ID: 0x1306 (model name: Light CTL Temperature server)

ID: 0x1307 (model name: Light HSL server)

Device OTA

Android APP:

Device OTA can perform GATT OTA upgrade on the device. The OTA interface can display the current device information, the different pid upgrade options of devices (unticked by default, users can tick the item as needed), click select file to select firmware, it shows target firmware version after selecting the firmware. Click "START" to start the upgrade, it will prompt start OTA and display the upgrade progress. When the upgrade is completed, it displays OTA_SUCCESS and the progress is 100%, and the device flashes slowly. To check whether the device is upgraded to the target version, users can refresh and view the vid data by long pressing the device on the APP homepage – settings – Composition Data (refer to section 2.6.3.2 Composition Data).

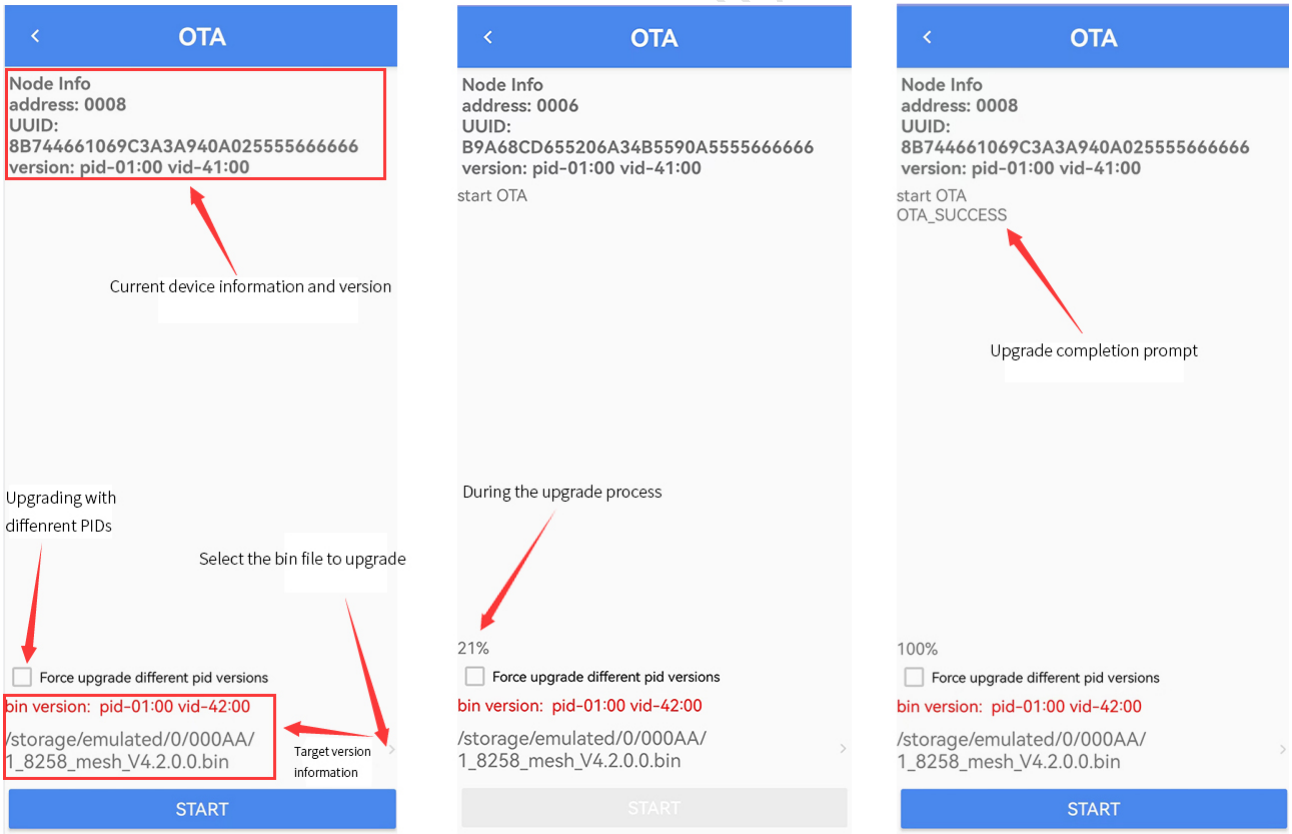


Figure 33.21: Android device OTA interface

iOS APP:

The OTA interface can display the current device information and the target version Pid and Vid. Tick the corresponding version and click "START" to start the upgrade. When the upgrade is completed, the device flashes slowly. To check whether the device is upgraded to the target version, users can refresh and view the vid data by long pressing the device on the APP homepage – settings – Composition Data (refer to section 2.6.3.2 Composition Data).

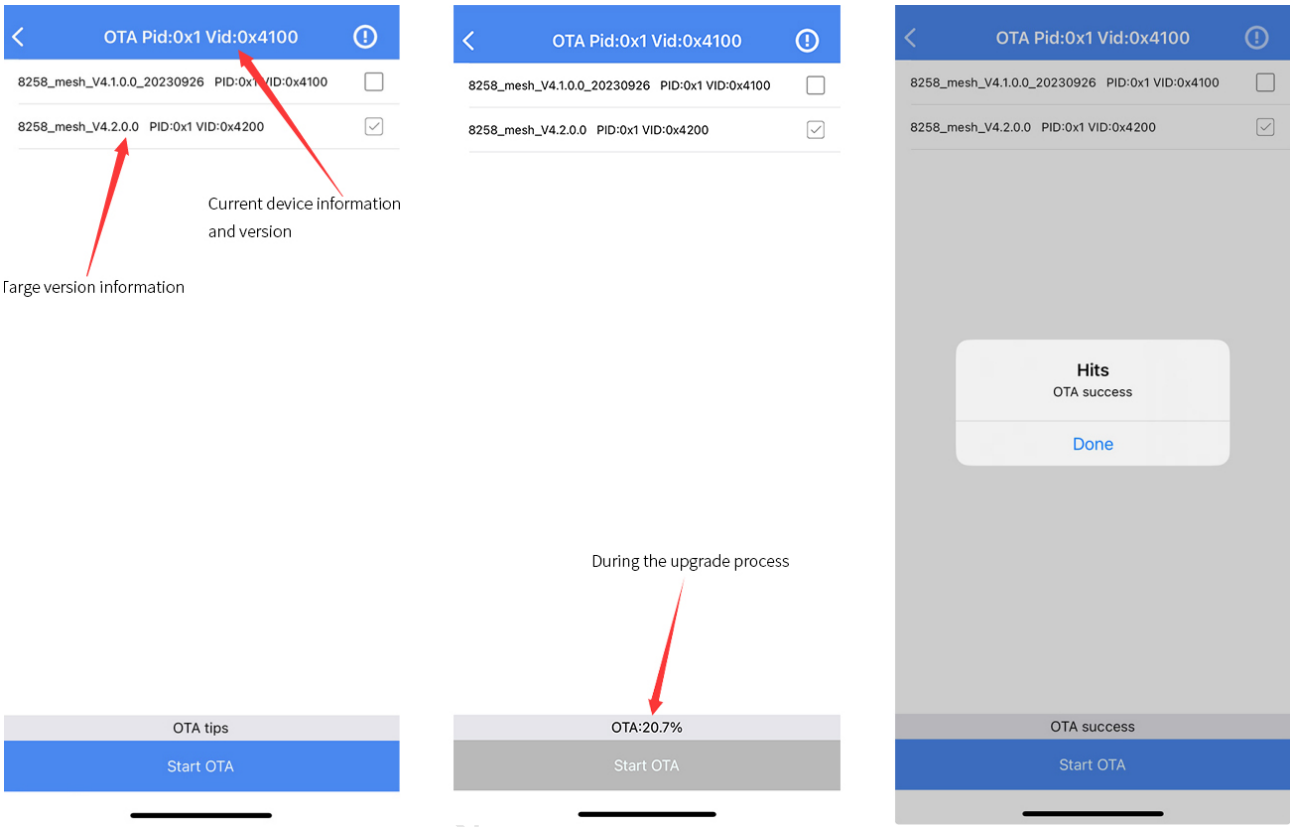


Figure 33.22: iOS device OTA interface

Publication (ele：xxxx model：CTL)

The Android/iOS APP will send status every 20 seconds after opening the corresponding device publication (it can be viewed in the log interface, CT light reports CtlStatusMessage, HSL light reports HslStatusMessage).

KICK OUT

The "KICK OUT" is used to kick out the current device. After kicking out, the device flashes slowly, and the device will be in the state of pending network.

33.3.7 Device Setting (Switch Device)

Long press the SW10 + SW13 keys of the unnetworked Switch remote control to trigger the broadcast (Note: the flicker frequency of the unnetworked state light is 200ms/s, and the networking state is 500ms/s) for networking. After the networking is successful, the broadcast also needs to be triggered. Long press the Switch icon on the APP homepage to enter the Device Setting interface to connect the remote control. At this time, the bottom of the interface will show "Device Connected" which represents the connected. If the connection fails, the position will pop up a button and user can click to reconnect.

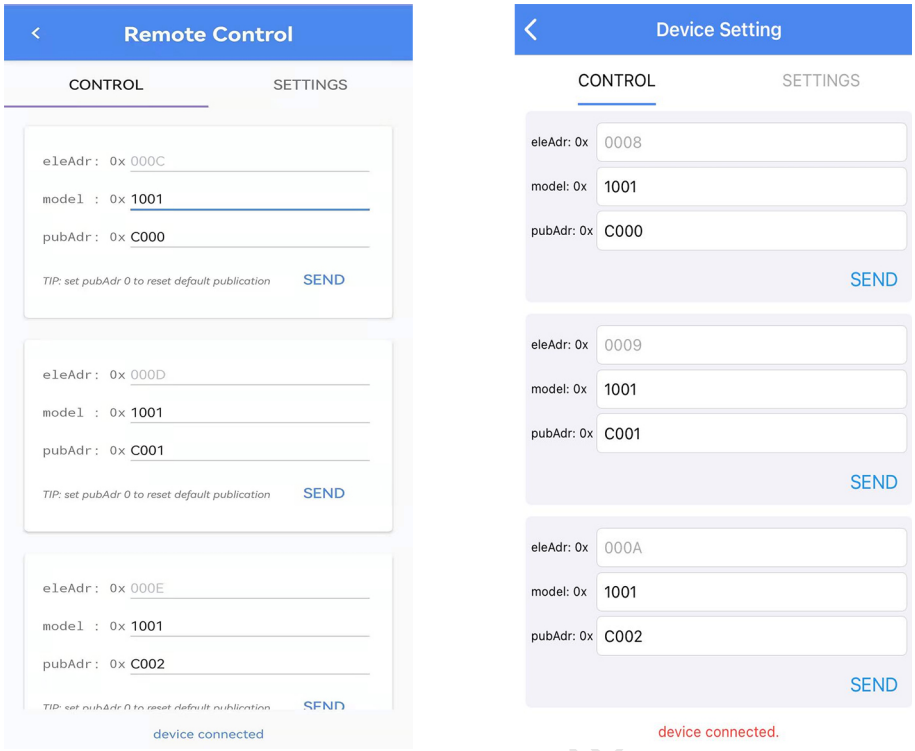


Figure 33.23: Android & iOS switch device setting interface

33.3.7.1 Switch Device Control

The 0x0008 in Eleadr corresponds to Switch remote control SW7/SW10 keys, 0x0009 corresponds to SW8/SW11, 0x000A corresponds to SW9/SW12, 0x000B corresponds to SW3/SW6, and Model can execute Switch-supported models (for details, please refer to Switch Device Composition Data). 0xC000 in Pubadr corresponds to Living room in Group, 0xC001 corresponds to Kichen, 0xC002 corresponds to Masterbedroom, and 0xC003 corresponds to Secondary bedroom. The Group to be controlled can be set at the specified key.

33.3.7.2 Switch Device Setting

Please refer to 33.2.6.3 Light Device Settings.

33.4 Group Interface

There are 8 groups in the Group interface (refer to 2.6.2 Single Device Group，Device interface – long press a device – click group for grouping), and assign devices to groups before operation.

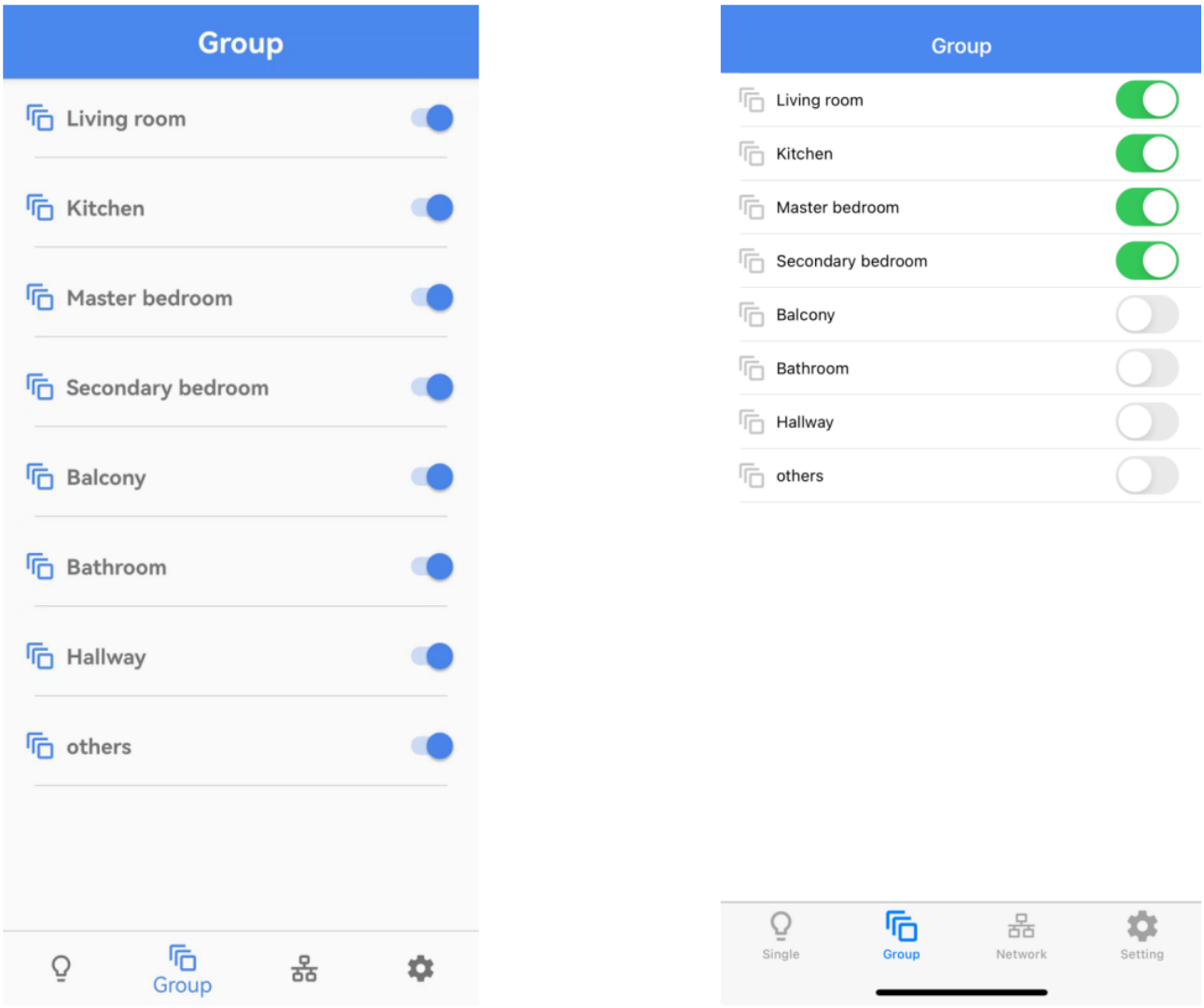


Figure 33.24: Android & iOS Group interface

33.4.1 On/Off Group

The Group interface enables user to On/Off the devices belonging to the corresponding group.

33.4.2 Group Setting

Long press the corresponding Group to enter the Group Setting interface.

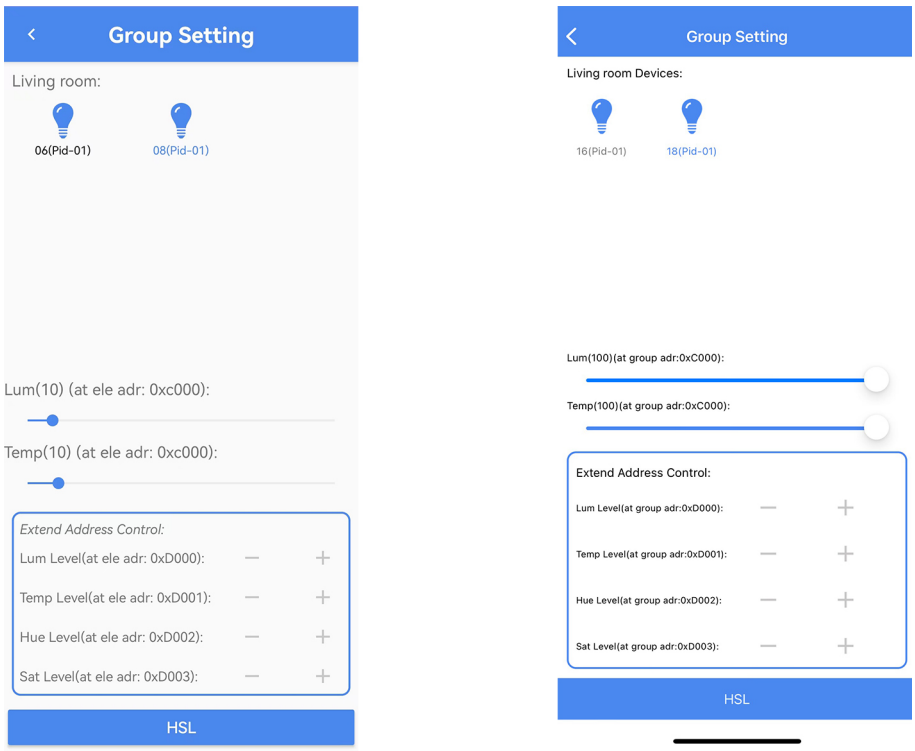


Figure 33.25: Android & iOS Group setting

33.4.2.1 On/Off Group Devices Individually

Click the device icon in the Group setting interface to on/off the device (the blue icon is On status, the gray is Off status), the blue device name is the direct connection device.

33.4.2.2 Lum & Temp

Lum adjusts the brightness of the devices belonging to the group, and Temp adjusts the color temperature.

33.4.2.3 Extend Address Control

Extend Address Control supports to enable the Level control function under the group, you can control the Lum Level, Temp Level, Hue Level and Sat Level of the group separately. Before grouping nodes, you need to open Extend SubscriptionLevel Service Model ID in APP Home – setting – settings.

33.4.2.4 HSL

The color palette enables user to adjust the color of the GRB, or user can adjust the color by adjusting R, G, B or H, S, L individually, RGB corresponds to HSL color, and V below the palette can adjust the brightness. Note: The device is required to enable the LIGHT_TYPE_HSL macro.

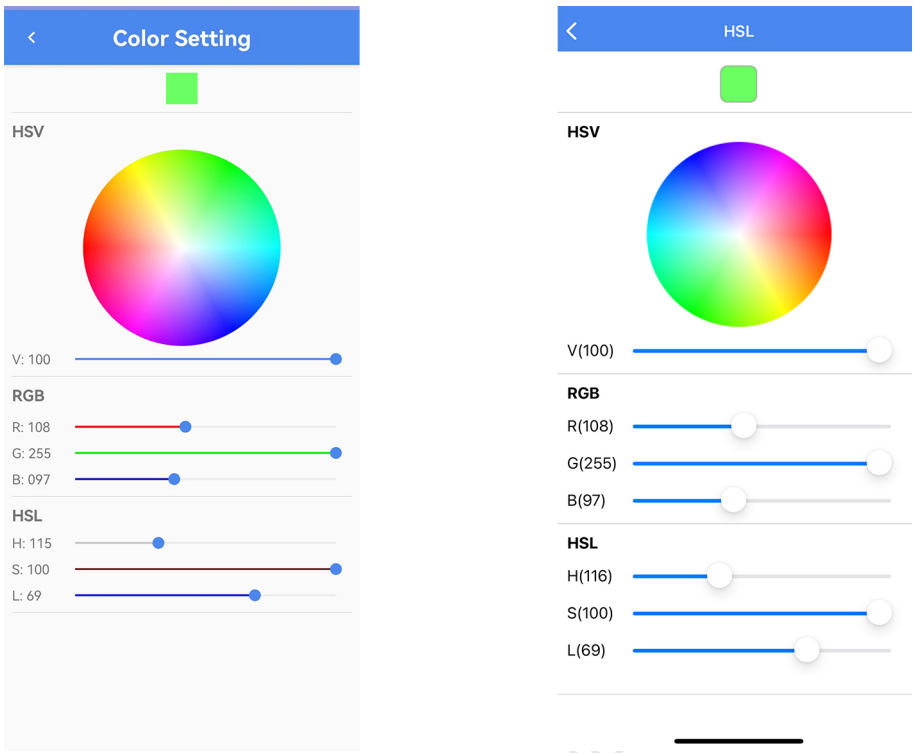


Figure 33.26: Android & iOS HSL interface

33.5 Network Interface

In the version V4.1.0.0, it adds a Network interface, which is used to view and set the current Network's Mesh Info, Scenes, Direct Forwarding, Mesh OTA, and Private Beacon individually.

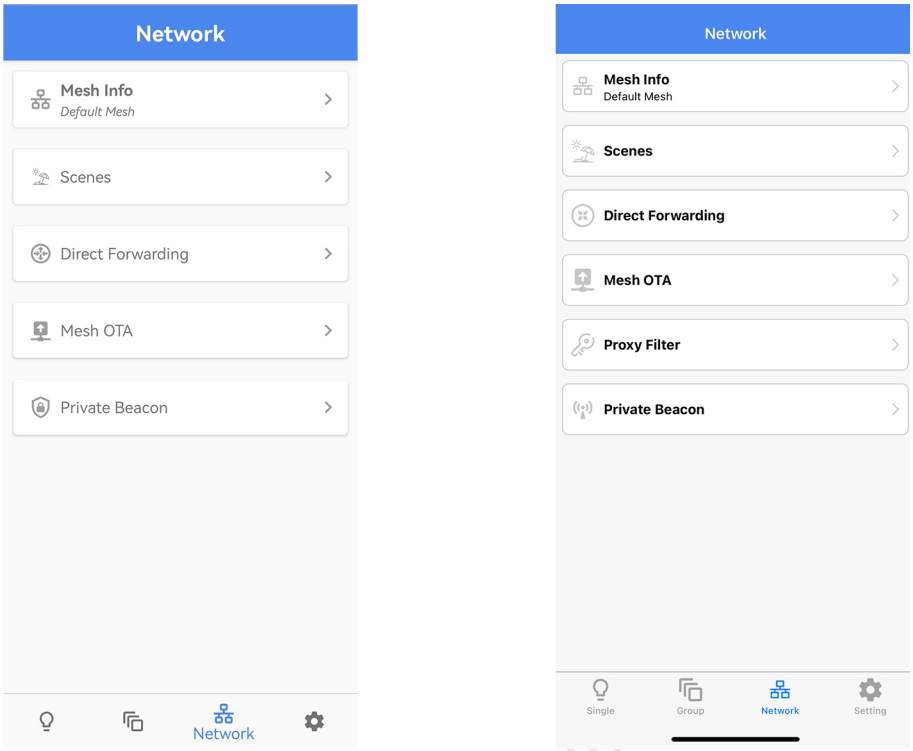


Figure 33.27: Android & iOS network interface

33.5.1 Mesh info

In Mesh Info interface, we can see the current Network's Mesh Name (click the Edit button on the upper right corner to edit the name), Mesh UUID, Iv Index, Sequence Number, Local Address, Net Keys/App Keys name and its index and key. Long press the key to copy the corresponding key (currently Android app has three built-in keys, in iOS app we need to manually add the Netkey List, Appkey List).

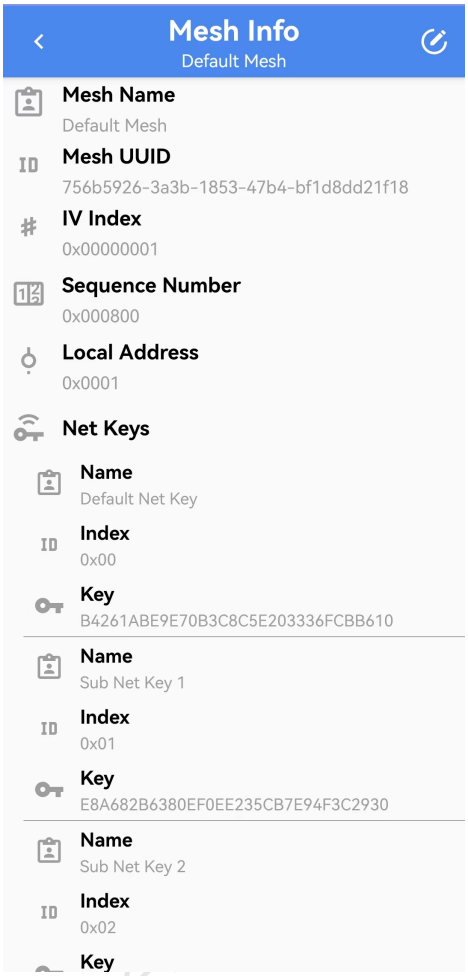


Figure 33.28: Android Mesh info interface

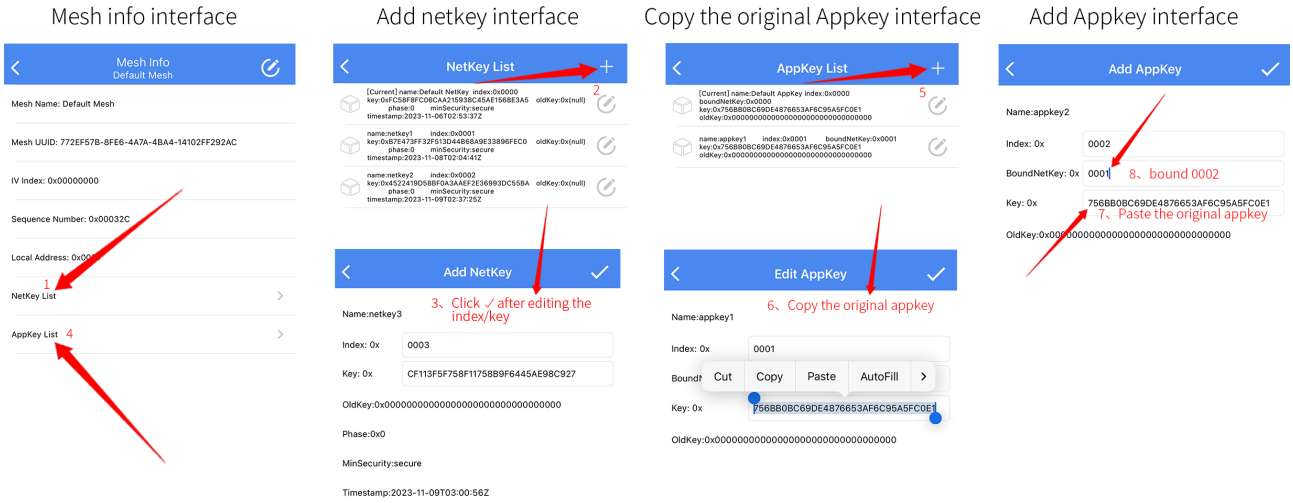


Figure 33.29: iOS Mesh info interface & Steps of adding Netkey / APPkey

33.5.2 Scenes

By Scene, we can save the current state of the specified device as a scene, in order to quickly start the set scene.

33.5.2.1 Create Scene

Click the + sign in the upper right corner of the iOS APP Scene interface to save the current state of the specified device as Scenes (Android version currently only creates Scenes, you need to tap the Edit button to configure Scenes).

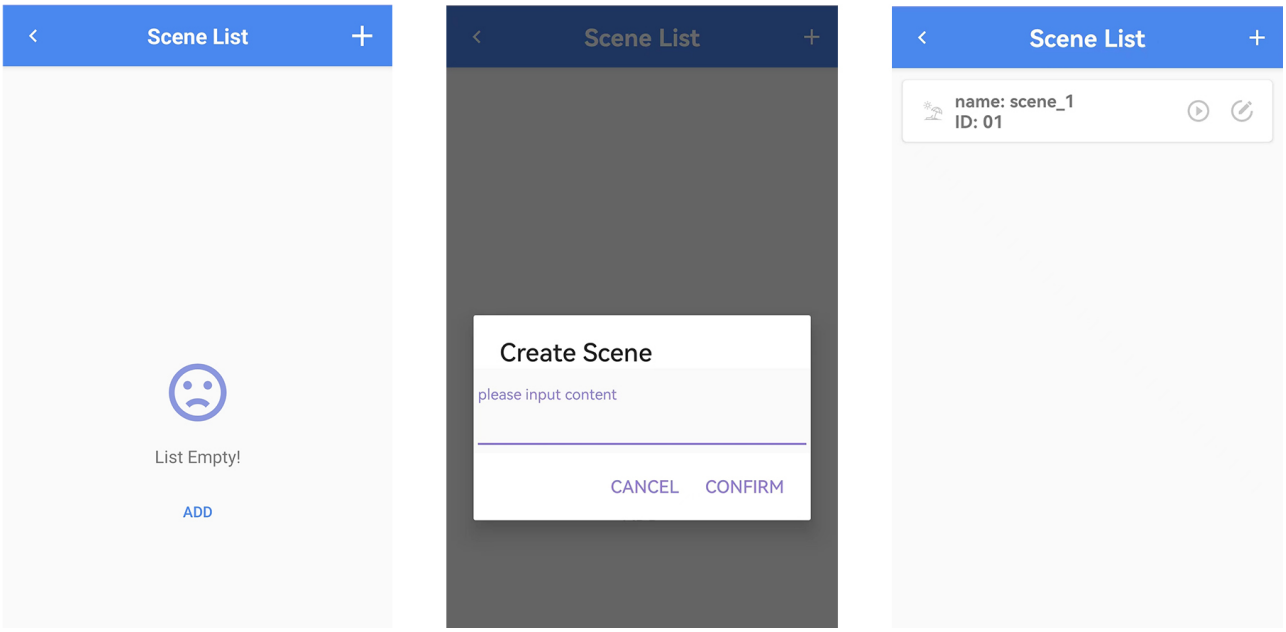


Figure 33.30: Android create Scene interface

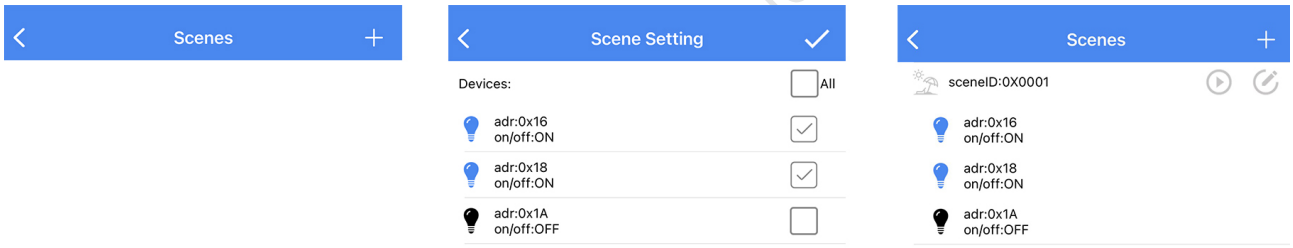


Figure 33.31: iOS create Scene interface

33.5.2.2 Edit Scene

After creating a Scene, click Scene List interface to edit Scenes.

Android edit scene: in the scene setting interface the left icon of "address" shows the current node switch status, the right of "address" shows the corresponding address, it only supports one element node in the right box (another element shows not support), you can tick the box to add the node to the scene. If a node that has been in the scene before needs to update its status, you can click to update the node to the current status.

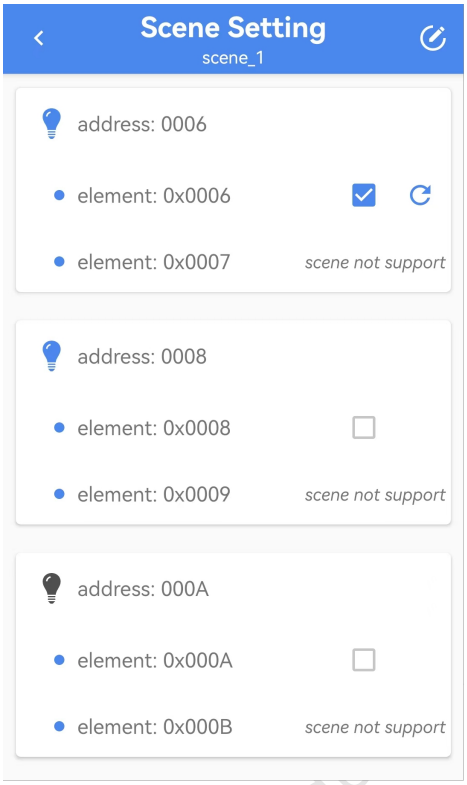


Figure 33.32: Android scene edit interface

iOS edit scene: in the scene setting interface, the left icon and the bottom of "adr" shows the current node switch status, the right of "adr" shows the corresponding address, check the corresponding box and click on the upper right corner of the Save button to update the selected node scene to the current state.

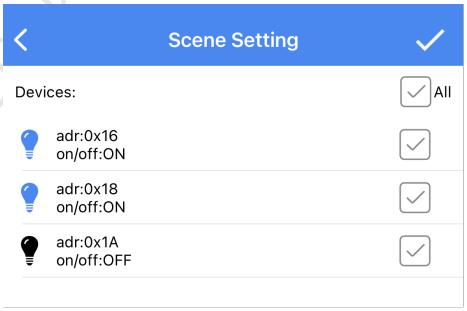


Figure 33.33: iOS scene edit interface

33.5.3 Direct Forwarding

Direct Forwarding reduces the number of packets forwarded over the air by engaging commands in forwarding at specified path nodes (routing tables).

Controllable Flooding: means that when a mesh message is propagated outward from the source, it is similar to the ripples spreading in all directions when a stone is thrown into the water. The range of transmission is controlled through ttl and relay feature controls the nodes involved in forwarding, this transmission is called controllable flooding.

The controllable flooding does not control the direction of message delivery and wastes bandwidth on parts of the network that are not related to the message. For example, if there are 2 switches in the middle of a large conference room that control the podium and the lights in the back row, when controlling the lights at the podium, messages are also retransmitted between the back row light nodes.

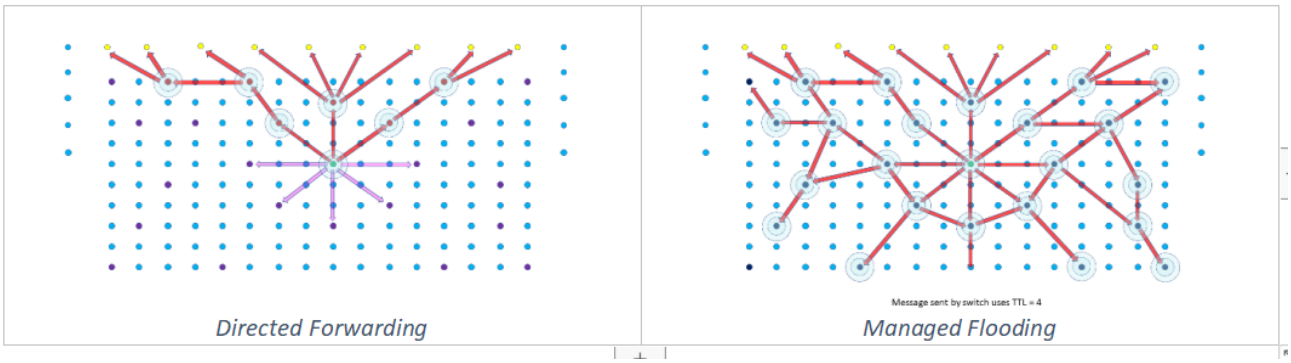


Figure 33.34: Directed Forwarding & Managed flooding

Routing table: is a command from the starting point to the end of the route intermediate nodes involved in the forwarding of a path identity, the end point can be unicast, multicast and virtual address, each routing node saves the path information through it, that is, routing table. A subset of routing nodes is selected to form a path in the network, a route may have 1 or more paths.

When a message is sent in the specified routing method, it will check if the path exists, and if it does, it will be sent as routed, otherwise it will be sent as flooding, and route establishment will be triggered. Messages sent by flooding are encrypted with network key and messages sent by routing are encrypted with directed key (derived from network key). In DF_TEST_MODE_EN mode, the node flashes when forwarding messages encrypted with the directed key.

The path establishment message is directed key encrypted, so all nodes flash their lights to indicate that the node forwarded the path establishment message. After the path is established, only the nodes on the path will flash, indicating that only the nodes on the path forwarded the directive.

33.5.3.1 Fixed Routing

Fixed Routing is configured and managed by the provisioner to forward nodes on a specified path. You need to turn on Direct Forwarding(Main), Direct Relay, Direct Proxy, and Direct Friend to the nodes on the path in the Direct Forwarding–Direct Toggles interface in advance. Note: If you don't check Direct forwarding(main), you will be prompted "(relay) check direct forwarding first" at the bottom.

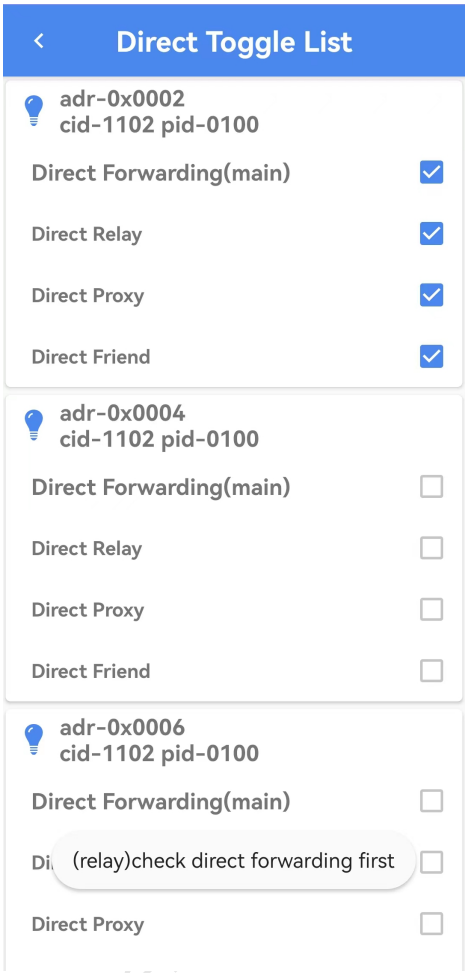


Figure 33.35: Fixed Routing Directangle Toggle list interface

The Direct forwarding interface allows you to add a fixed-route path to the mesh network by clicking the Add Table button at the bottom. A path contains a start point and an end point, as well as the nodes through which the path passes. When a message is routed from the start point, all nodes on the path will participate in forwarding the message, and nodes not on the path will ignore the message. You can see that the LEDs of nodes 0x06, 0x0e, and 0x16 on the path are flashing, while the LEDs of other nodes not on the path are not flashing.

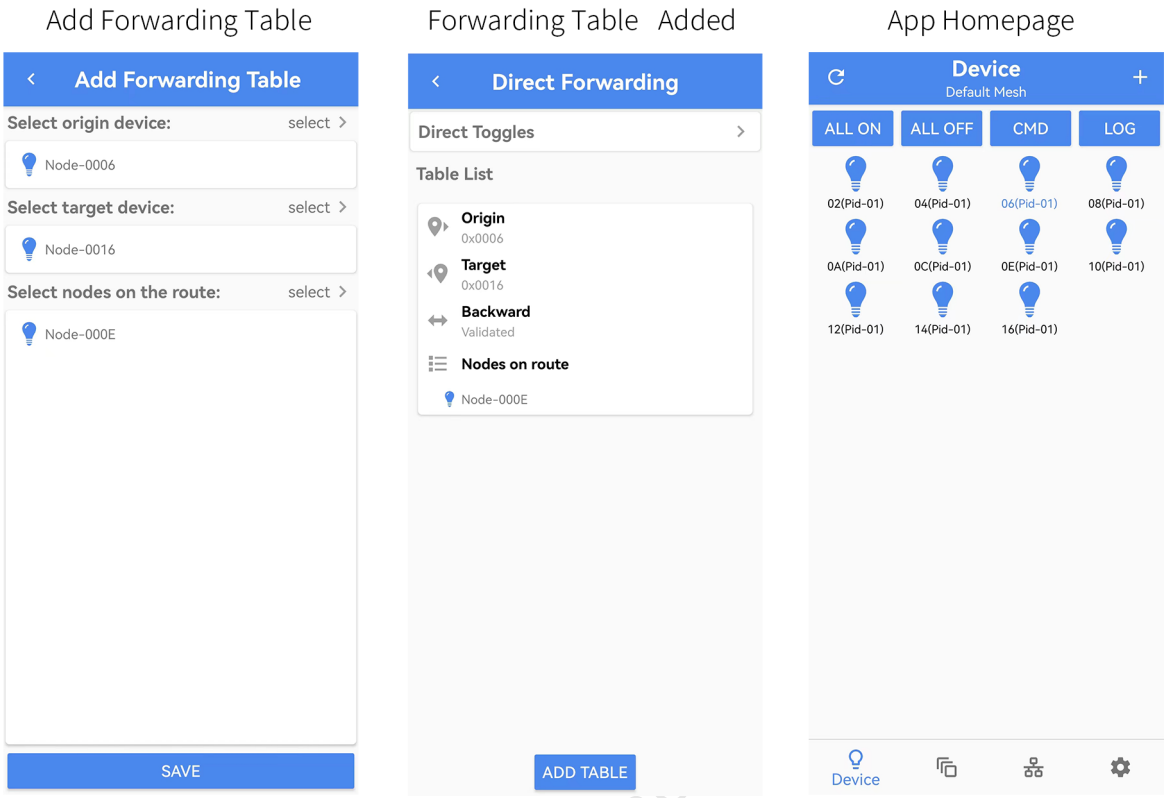


Figure 33.36: Adding a fixed route

33.5.3.2 Non-fixed Routing

The non-fixed routing does not need to be operated by Direct Forwarding in the APP, it is created and maintained by the sender (path origin), which sends the message in a controlled flood and triggers the route establishment. The route establishment message is Directed Key encrypted, so all nodes flash to indicate that the node forwarded the route establishment message. After the path is established, only the nodes on the path flash lights, indicating that only the nodes on the path forwarded the command.

Non-fixed Route Establishment Rules: The figure below illustrates the creation of two paths from PO (Path Origon) to PT (Path Target) by selecting the shortest path that meets the set energy threshold.

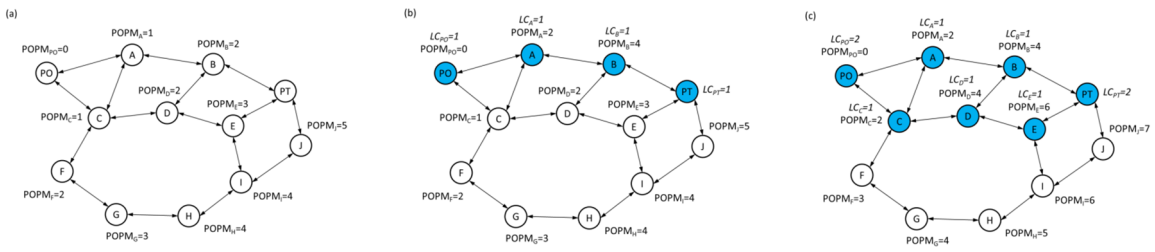


Figure 33.37: Non-fixed route establishment rules

33.5.4 Mesh OTA

Mesh OTA allows simultaneous OTA upgrades of multiple devices specified by the mesh network, and there are three ways for Mesh OTA to be loaded: (1) No Extend (All nodes short packet loaded); (2) Extend GATT Only (Long packet loading for directly connected nodes and short packet loading for non-directly connected nodes); (3) Extend GATT & ADV (All nodes long packet loading), setting path: APP home page click on the lower right corner Setting–Settings–Extend Bearer Mode.

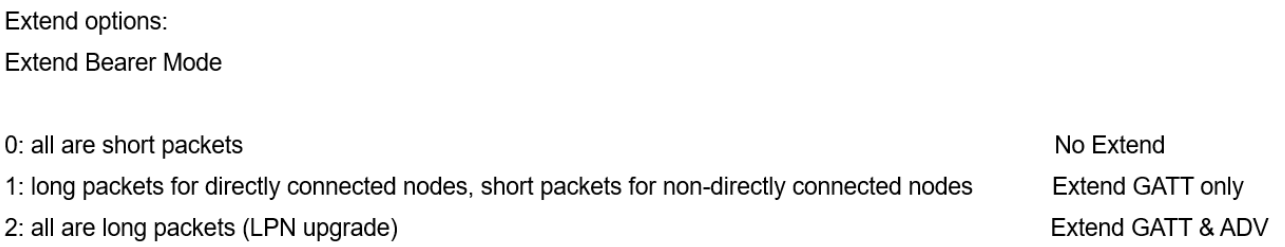


Figure 33.38: Introduction to loading methods

Note:

(1) Mesh OTA is turned off by default, you need to turn on the macro of node MD_MESH_OTA_EN, otherwise it will not support Mesh OTA, and you can't check this device as the object of mesh OTA. Open method: in the mesh_config.h file,

- Enable MD_MESH_OTA_EN
- It needs to turn on DISTRIBUTOR_UPDATE_SERVER_EN if the directly connected node after testing is as distributor mode.
- If it is needed to use Telink's Extended Broadcast Packet mode to speed up the OTA time, you need to set EXTENDED_ADV_ENABLE to 1.

(2) After the Mesh OTA upgrade is complete, exit the mesh OTA interface and re-enter to read off the upgraded version.

(3) For iOS, put the upgraded bin file through itunes – file sharing – TelinkSigMesh to show up in the mesh OTA upgrade interface.

33.5.4.1 Distributor: Phone mode upgrade (App as distributor mode)

The Distributor chooses the Phone method to upgrade，it directly transfers OTA data to the target device through the phone.

Specific operation steps:

Android version:

At the APP home page, click on the bottom right Network – Mesh OTA according to the following step by step operation.

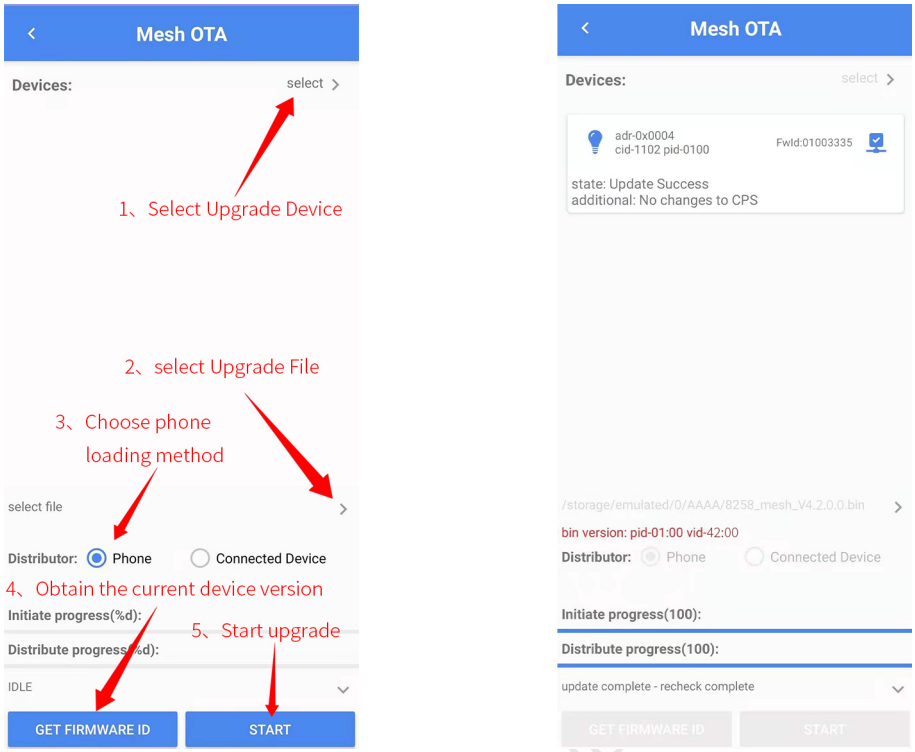


Figure 33.39: Android phone method upgrade steps and upgrade completion interface

iOS version:

At the APP home page click on the bottom right Network – Mesh OTA according to the following step by step operation.

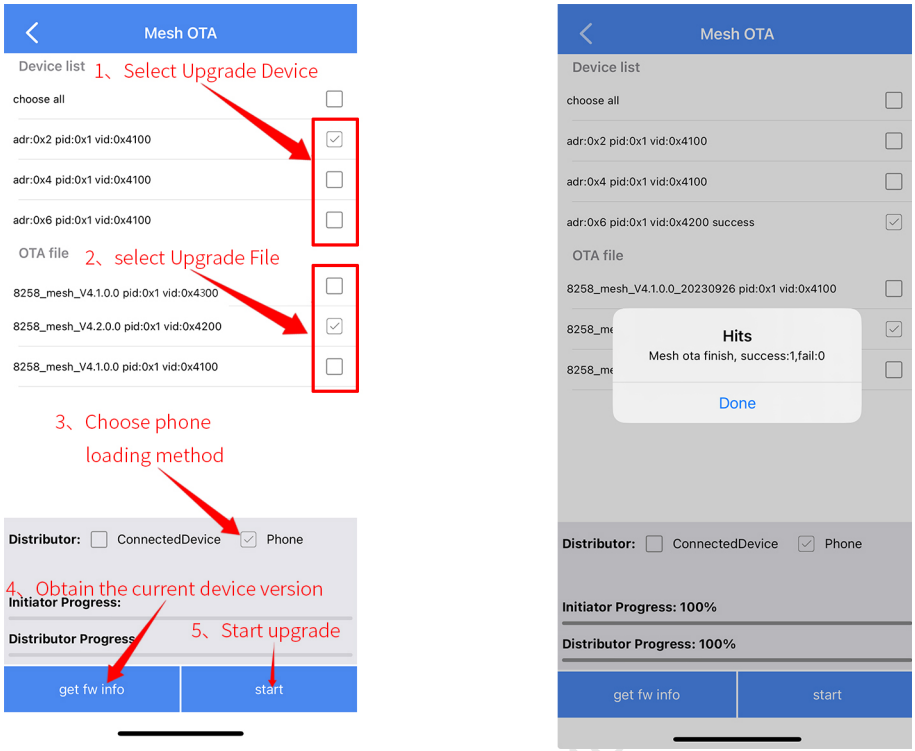


Figure 33.40: iOS phone method upgrade steps & upgrade completion interface

33.5.4.2 Distributor：Verify and Apply Mode Upgrade (Directly Connected Nodes as Distributor Mode)

For Distributor, select connected device, FOr Apply Policy, select Verify and Apply method of upgrading, upload the firmware to the directly connected node through the mobile phone, and then distributes it to the target node through the directly connected node, and then automatically applies the new version after loading is completed.

Note:

> Make sure that the DISTRIBUTOR_UPDATE_SERVER_EN macro is enabled for the directly connected node.

Specific operational steps:

Android version:

At the APP home page, click on the bottom right Network – Mesh OTA according to the following steps to operate.

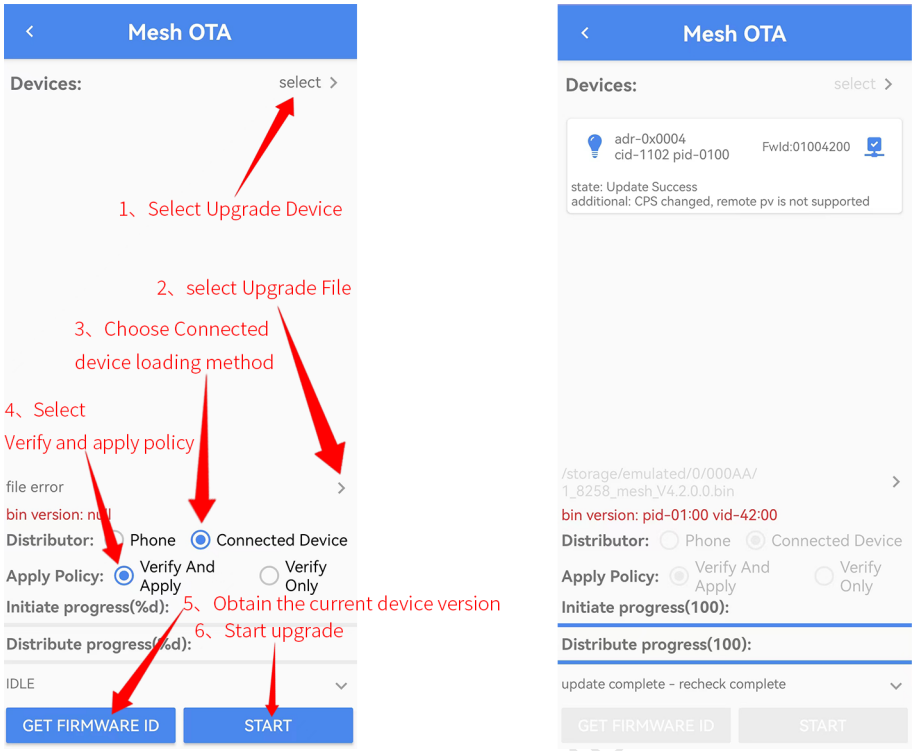


Figure 33.41: Android verify and apply upgrade steps & upgrade completion interface

iOS version:

At the App home page, click on the lower right Setting – Mesh OTA according to the following steps.

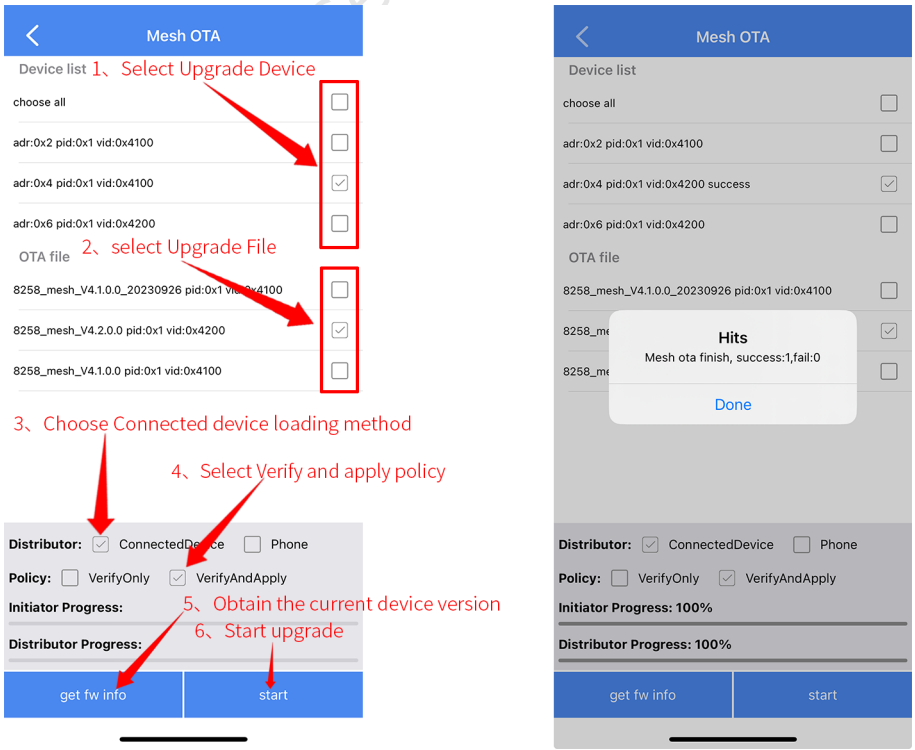


Figure 33.42: iOS verify and apply upgrade steps & upgrade completion interface

33.5.4.3 Distributor：Verify Only Mode Upgrade (Directly Connected Nodes as Distributor Mode)

For Distributor, choose the Verify only method to upgrade, upload firmware to the directly connected node through the mobile phone, and then distribute it to the target node through the directly connected node, and after the loading is completed, you need to reconnect the node with the APP before applying the new version.

Note：

> Make sure that the DISTRIBUTOR_UPDATE_SERVER_EN macro is enabled for the directly connected node.

Specific operational steps：

Android version：

At the APP home page, click on the bottom right Network – Mesh OTA according to the following steps.

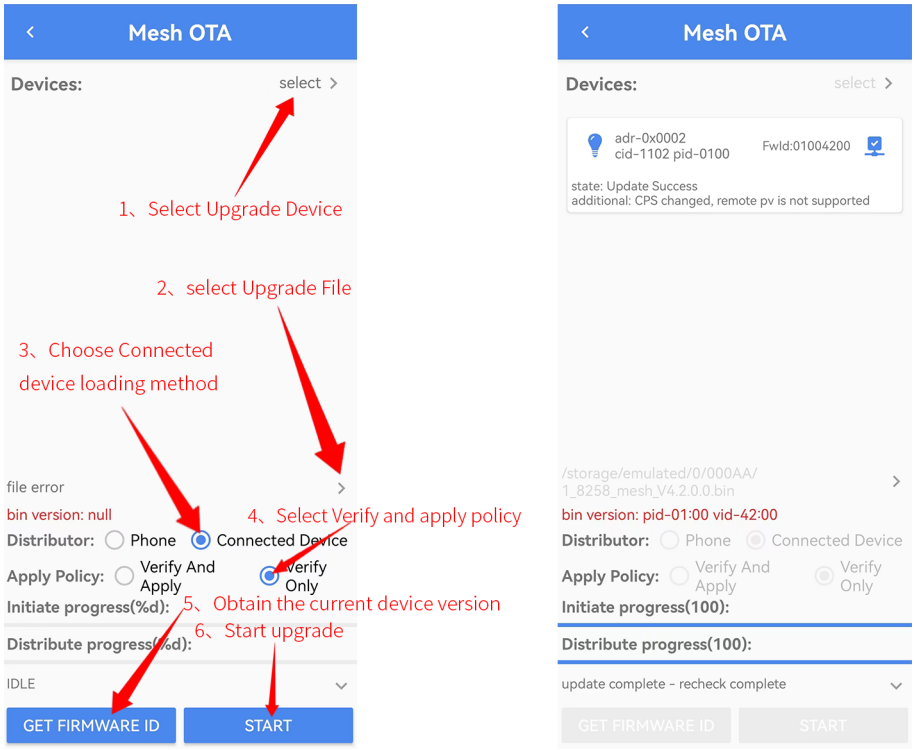


Figure 33.43: Android verify only method upgrade steps & upgrade completion interface

iOS version：

Click Setting – Mesh OTA at the bottom right of APP homepage and follow the steps in the figure below.

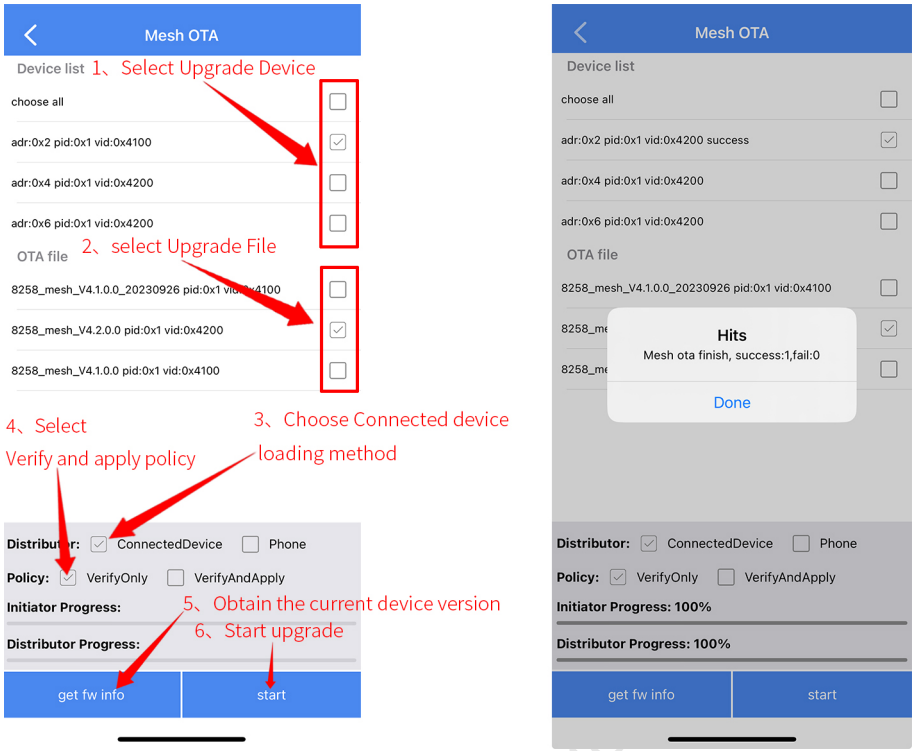


Figure 33.44: Android verify only method upgrade steps & upgrade completion interface

33.5.5 Private beacon

Using private beacon can set to send the specified broadcast after the node is networked, enable MD_PRIVACY_BEA and PRIVATE_PROXY_FUN_EN in the mesh_config.h file.

33.5.5.1 Config GATT Proxy

Open Config GATT Proxy alone and always send only Network ID broadcasts with light blue APP broadcast type 0x00.

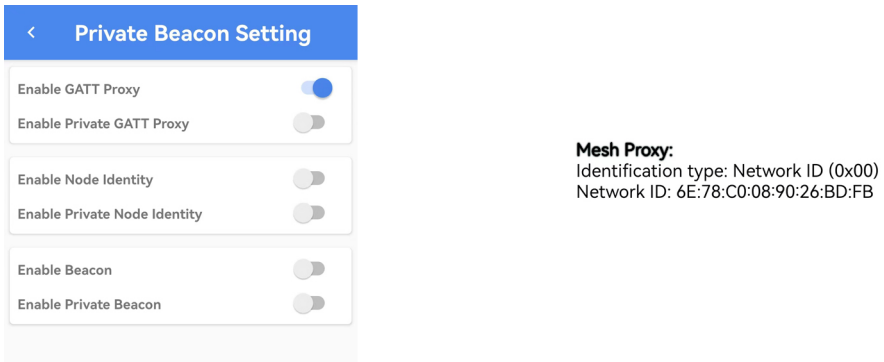


Figure 33.45: Open config GATT proxy & broadcast type separately

33.5.5.2 Private GATT Proxy

Open private GATT Proxy alone, Network ID is private, send encrypted processed Network ID with new Mac address, light blue APP broadcast type 0x02 .

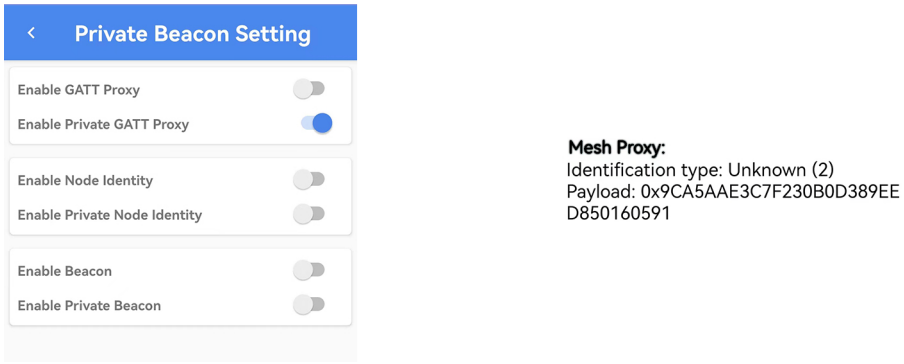


Figure 33.46: Open config GATT proxy & broadcast type separately

33.5.5.3 Config Node Identity

Open Config Node Identity individually, send Node Identity information within 60 seconds (light blue APP broadcast type 0x01), and automatically switch to send the default Network ID broadcast after 60 seconds (light blue APP broadcast type 0x00).

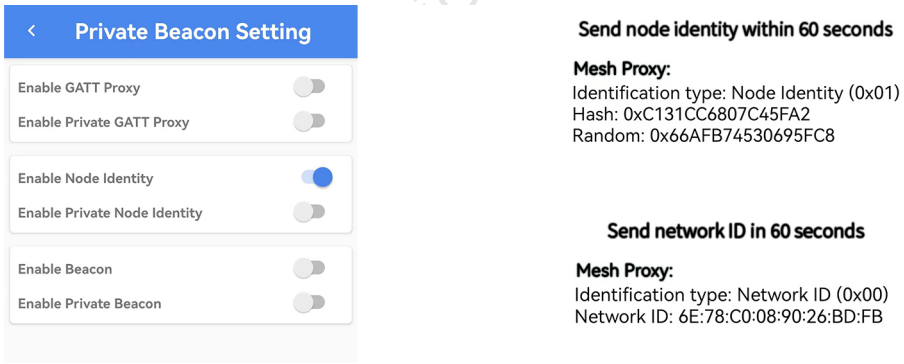


Figure 33.47: Open config node identity & broadcast types individually

33.5.5.4 Private Node Identity

Open Private Node Identity individually, Node Identity is in private state, send encrypted Node Identity broadcast (light blue APP broadcast type 0x03) with new Mac address within 60 seconds, and automatically switch to send default Network ID broadcast (light blue APP broadcast type 0x00) after 60 seconds.

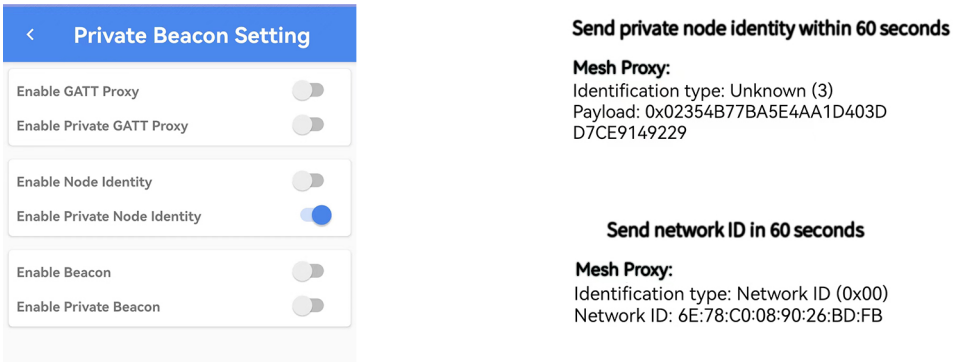


Figure 33.48: Open private node identity & broadcast types individually

33.5.5.5 Config GATT Proxy + Config Node Identity

Open Config GATT Proxy and Config Node Identity at the same time, send Node Identity information within 60 seconds (light blue APP broadcast type is 0x01), and automatically switch to send the default Network ID broadcast after 60 seconds (light blue APP broadcast type is 0x00).

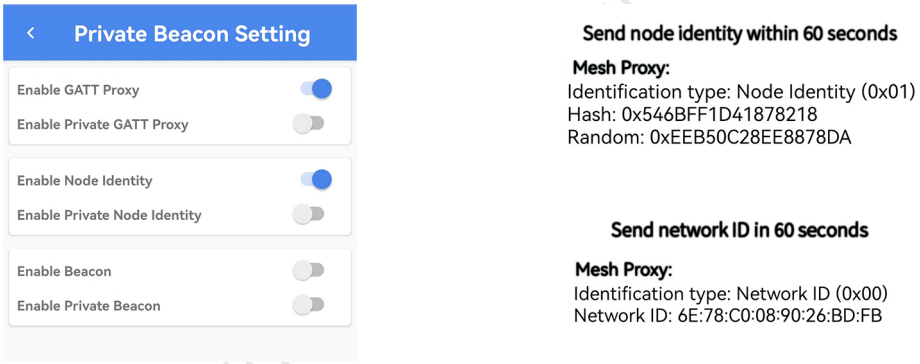


Figure 33.49: Config GATT proxy + Config node identity & broadcast type

33.5.5.6 Config GATT Proxy + Private Node Identity

At the same time, open Config GATT Proxy and Private Node Identity, the Node Identity is private, the encrypted Node Identity broadcast will be sent with the new Mac address within 60 seconds (light blue APP broadcast type 0x03), and after 60 seconds, it will automatically switch to send the default Network ID broadcast (light blue APP broadcast type 0x00).

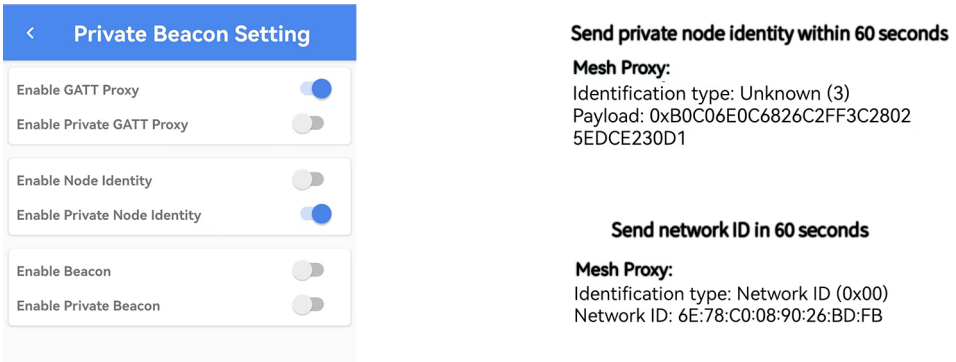


Figure 33.50: Config GATT proxy + Private node identity & broadcast type

33.5.5.7 Private GATT Proxy + Config Node Identity

At the same time open Private GATT Proxy + Config Node Identity, within 60 seconds send Node Identity information (light blue APP broadcast type is 0x01), after 60 seconds Network ID is private, with the new Mac address after encrypted processing and then send the broadcast to the outside world, light blue APP broadcast type is 0x02.

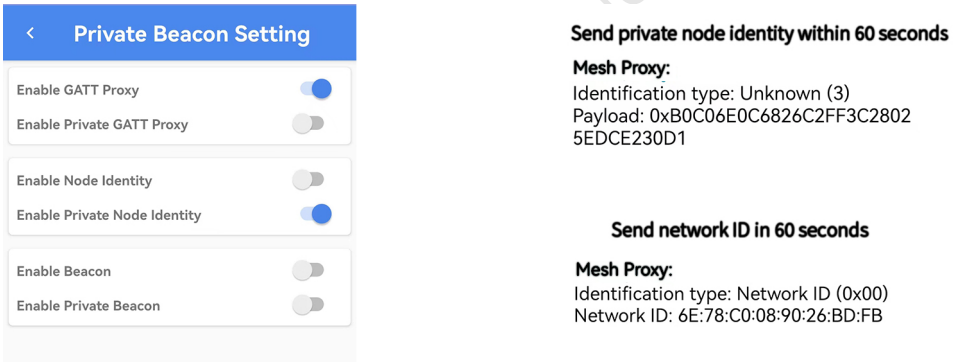


Figure 33.51: Private GATT proxy + Config node identity & broadcast type

33.5.5.8 Private GATT Proxy + Private Node Identity

Open Private GATT Proxy + Private Node Identity at the same time, Node Identity and Network ID are both in private state, send encrypted Node Identity broadcast (light blue APP broadcast type 0x03) with new Mac address within 60 seconds, send encrypted Network ID broadcast (light blue APP broadcast type 0x02) with new Mac address after 60 seconds.

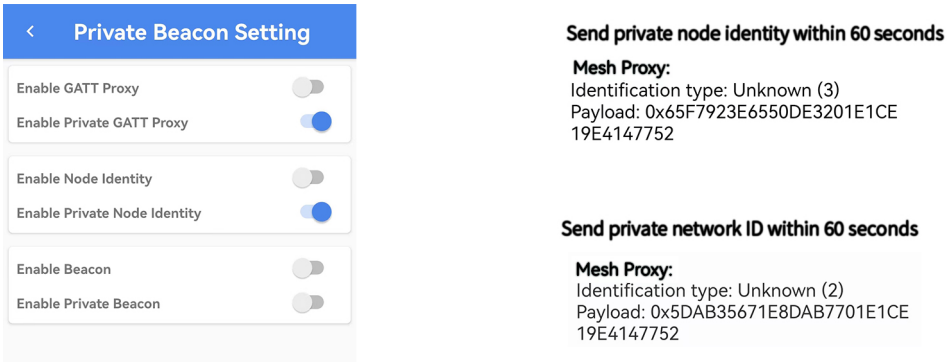


Figure 33.52: Private GATT proxy + Private node identity & broadcast state

33.5.5.9 Config Beacon

After opening Config Beacon, the node will send Non-Comnecable undrected Adv Packet of Secure Network Beacon type every 10 seconds, which can be viewed through light blue APP (whether the phone can receive the beacon packet when the node sends the beacon packet every 10 seconds depends on the Bluetooth refresh rate of the mobile phone), or you can view it through the packet grabbed by Netkey and APPkey through ellisys.

light blue APP beacon packet

mesh beacon:
Beacon type: Secure Network (0x01)
Key Refresh flag: false
IV Update flag: IV Update active
Network ID: 6E:78:C0:08:90:26:BD:FB
IV Index: 1
Authentication value:
0x74DB8A889DE4411C

ellisys beacon packet

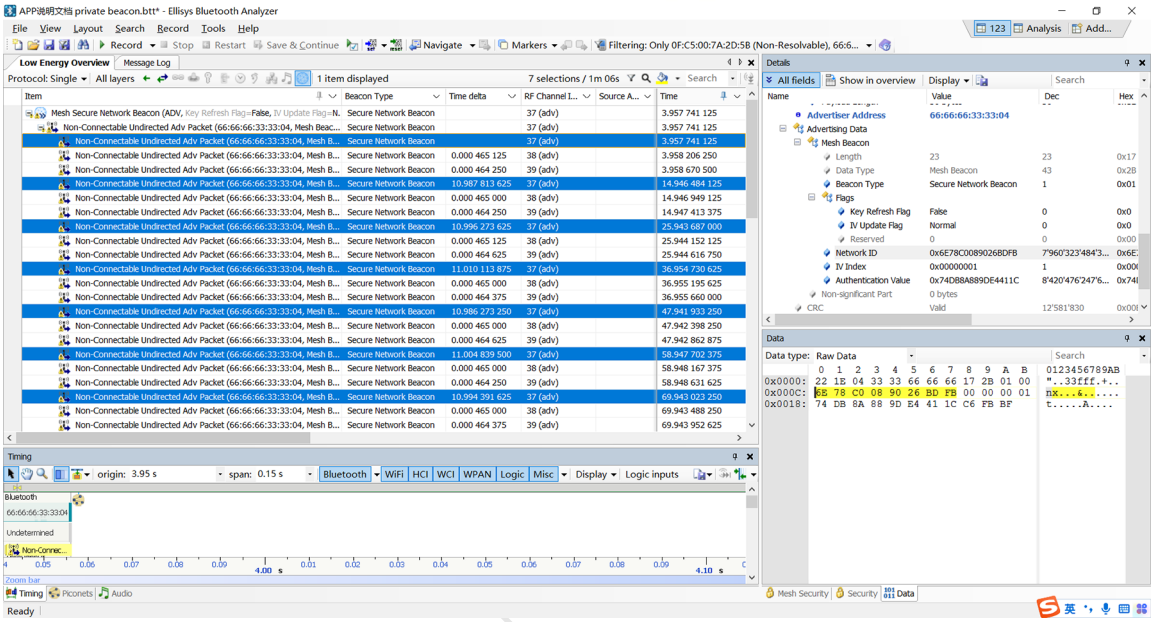


Figure 33.53: Light blue APP & ellisys receive beacon packets after opening beacon

33.5.5.10 Private Beacon

To open Private Beacon, you need to open Private GATT Proxy at the same time, the node will change the Mac address, and send Reserved (0x02) Beacon type Non-Comnecable undrected every 10 seconds, which can be viewed through light blue APP (whether the phone can receive the beacon packet when the node sends it every 10 seconds depends on the Bluetooth refresh rate of the phone), and can also be viewed through Netkey and APPkey by grabbing packets through ellisys.

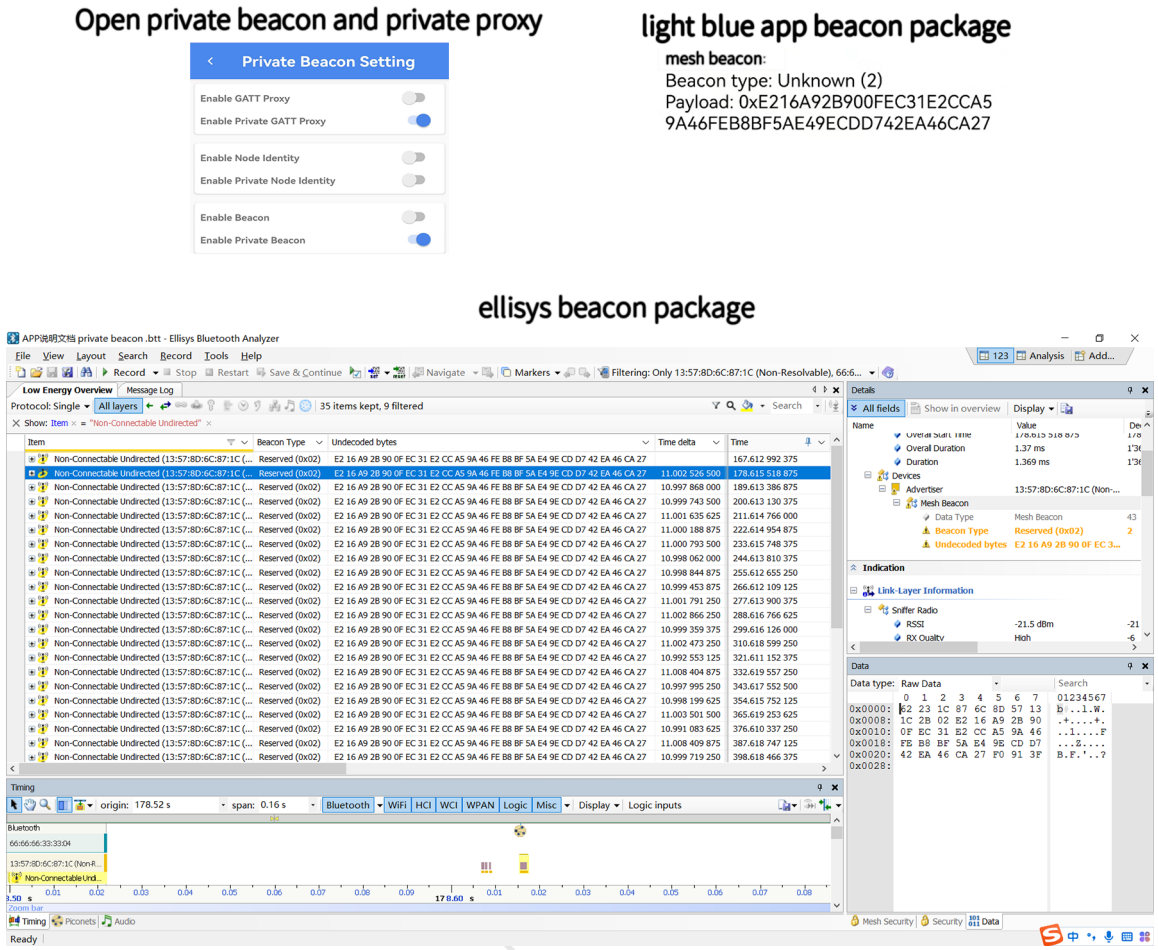


Figure 33.54: Open private beacon after light blue APP & ellisys receive beacon packets

33.5.5.11 Beacon + Private Beacon

Open Beacon and Private Beacon at the same time (Private GATT Proxy needs to be opened at the same time), the node will send Non-Comnecable undrected Adv Packet of Secure Network Beacon type every 10 seconds with the original Mac address, at the same time, the node will change the Mac address and send a Non-Comnecable undrected Adv Packet of type Reserved (0x02) Beacon every 10 seconds, it can be viewed by light blue APP (whether the phone can receive the beacon packet when the node sends it every 10 seconds depends on the Bluetooth refresh rate of the phone), and it can also be viewed by Netkey and APPkey through ellisys packet grabbing.

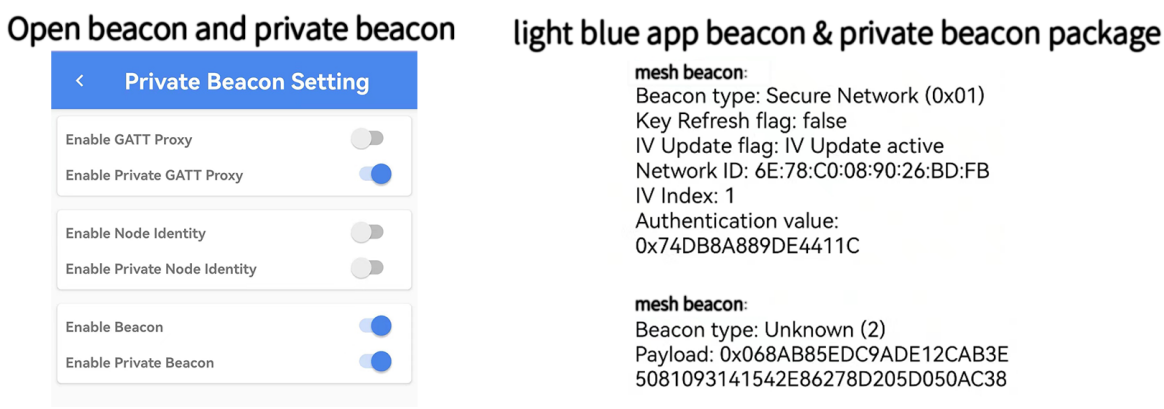


Figure 33.55: Beacon packets received by light blue APP after opening beacon + private beacon

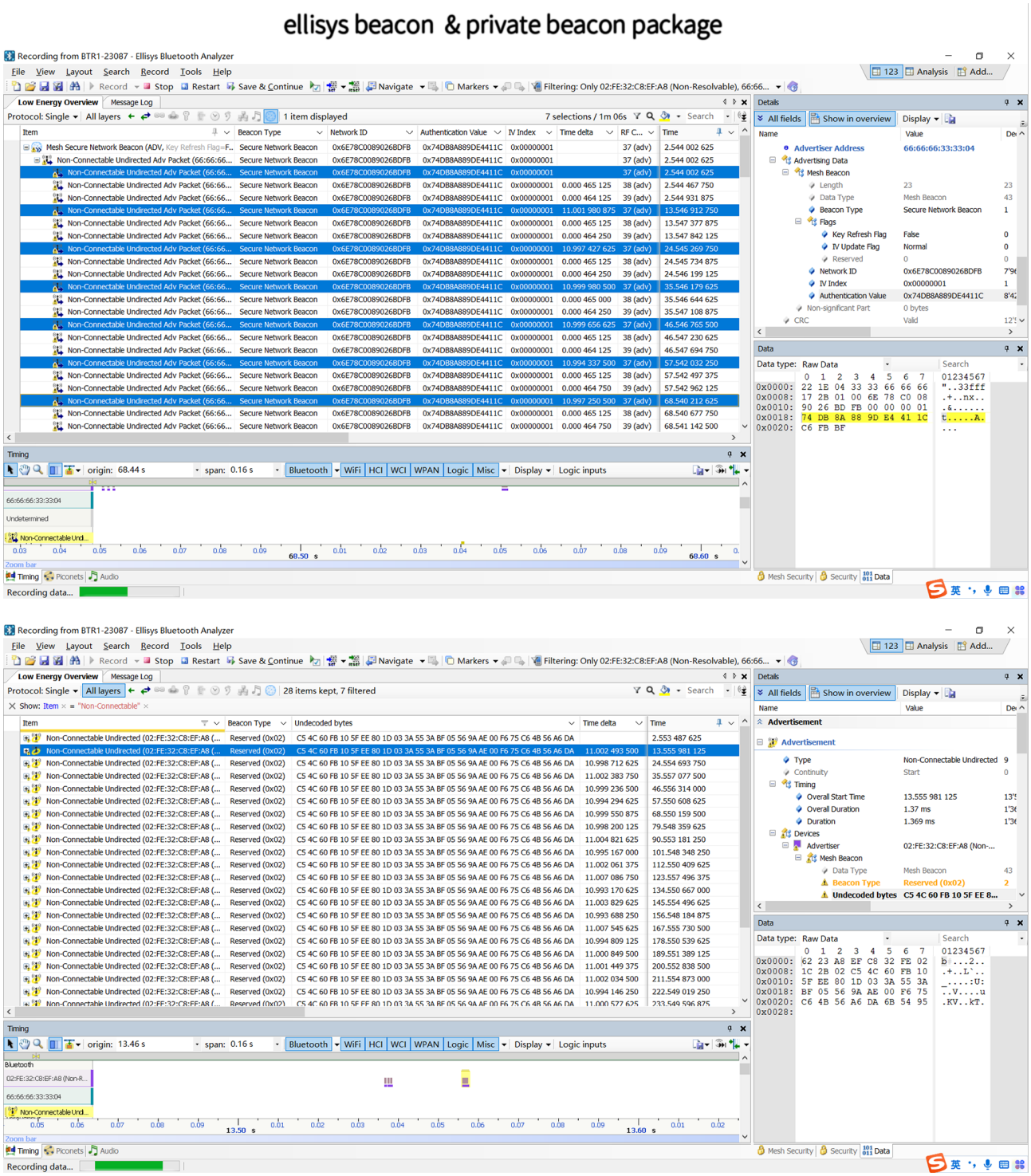


Figure 33.56: Beacon packets received by ellisys after opening beacon + private beacon

33.6 Setting Interface

The Setting interface allows for Manage Network, OOB Database, Root Cert, settings (Enable Log, Enable Private Mode, Provision Mode, Enable Subscription Level Service, Enable Log, Enable Private Mode, Pro-

vision Mode, Enable Subscription Level Service, Extend Bearer Mode, Use No-OOB Automatically, Share Import Complete Action, Online Status, Reset Settings), How To Import Bin File, Get More Telink Apps.

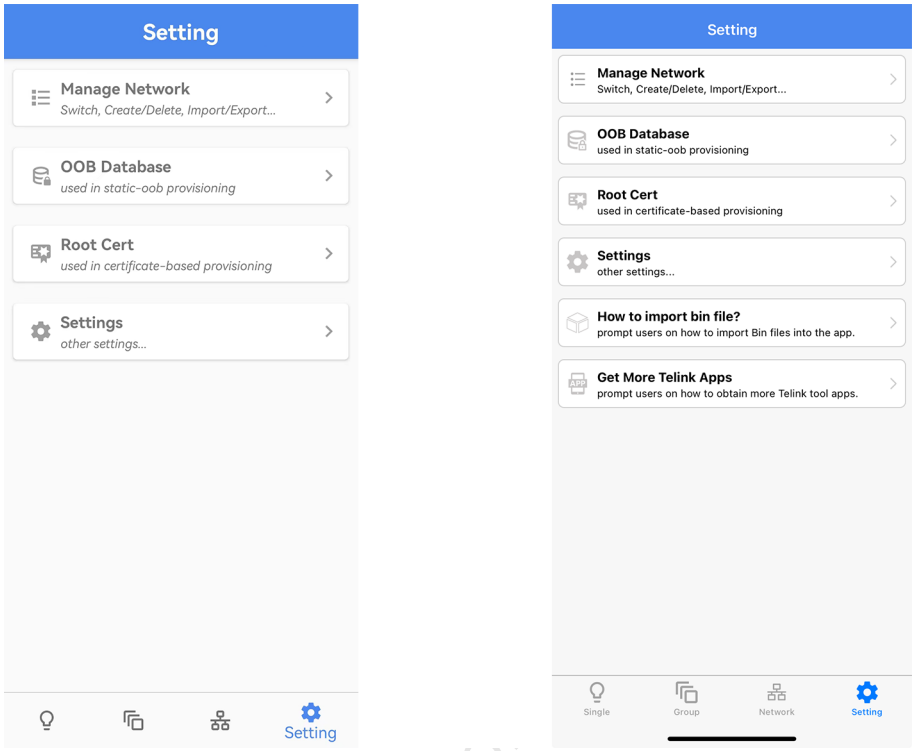


Figure 33.57: Android & iOS setting interface

33.6.1 Manage Network

Difference between Manage Network and the Network interface at the bottom of the app: in Manage Network we can view, manage, apply and share different networks, while the Network at the bottom of the app is for the current network to perform Mesh OTA, Scene and other operations. The network selected in the blue box is the current network, click the + sign in the upper right corner to create a new network.

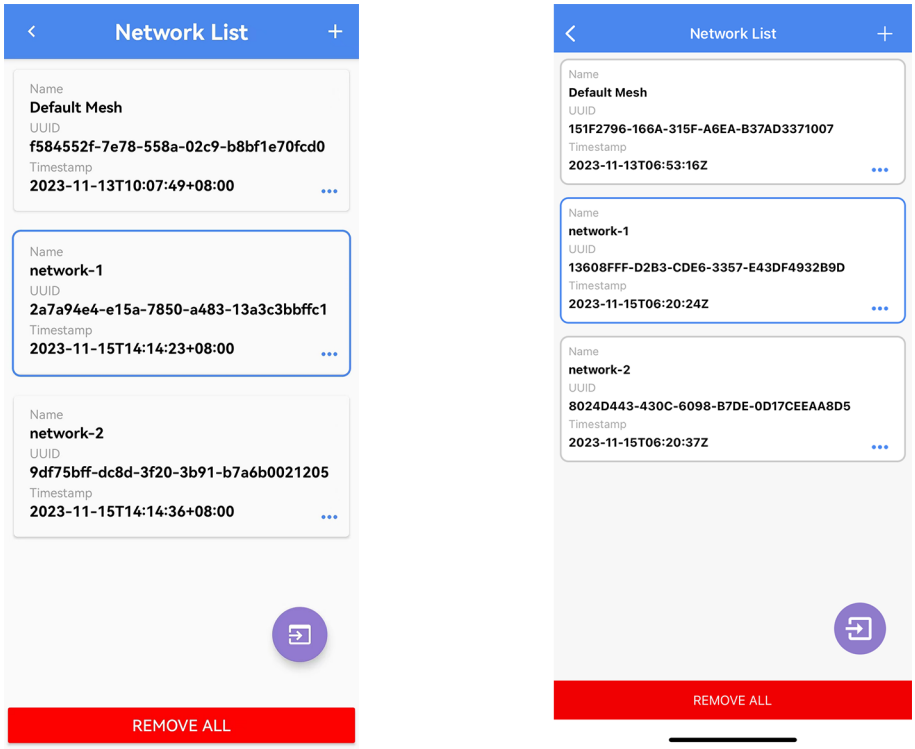


Figure 33.58: Android & iOS manage network interface

33.6.1.1 Show Detail

Click on the Network you want to view and select Show detail to view the details of the Network (see section 4.1 for the Mesh Info interface).

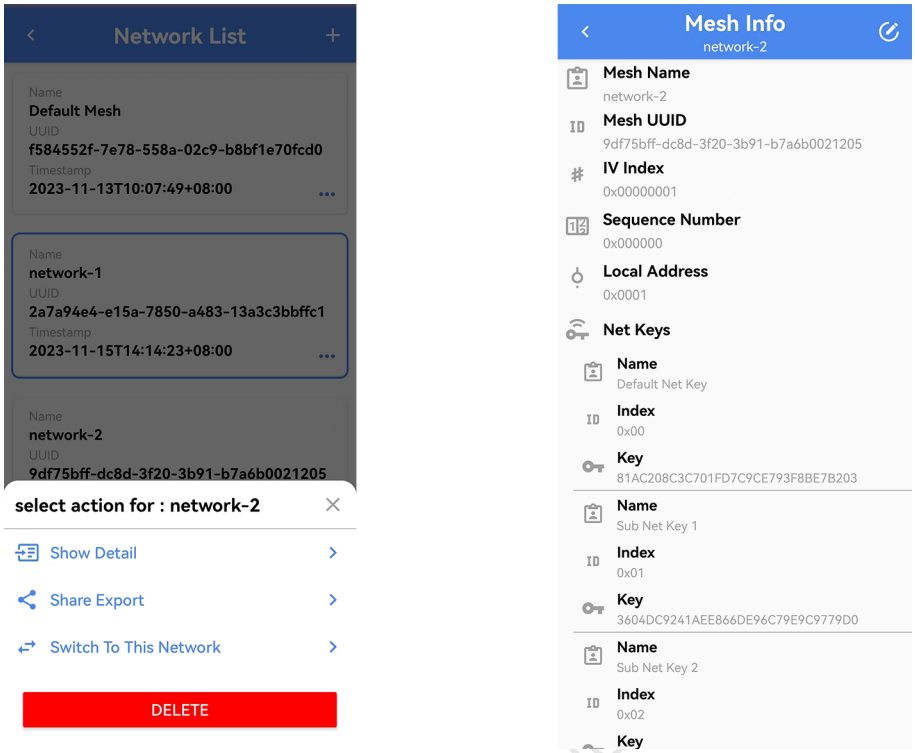


Figure 33.59: Show detail interface

33.6.1.2 Share Export

(1) Export by File

At Network List interface, click on a shared Network and select Share Export, select the Net Key you need to share, select Json File for the sharing method and click on the export button to export the json file, the Android version of the exported json file is saved in storage/emulated/0/TelinkBleMesh directory, and the iOS version of the exported json file is shared through the iTunes file inside .

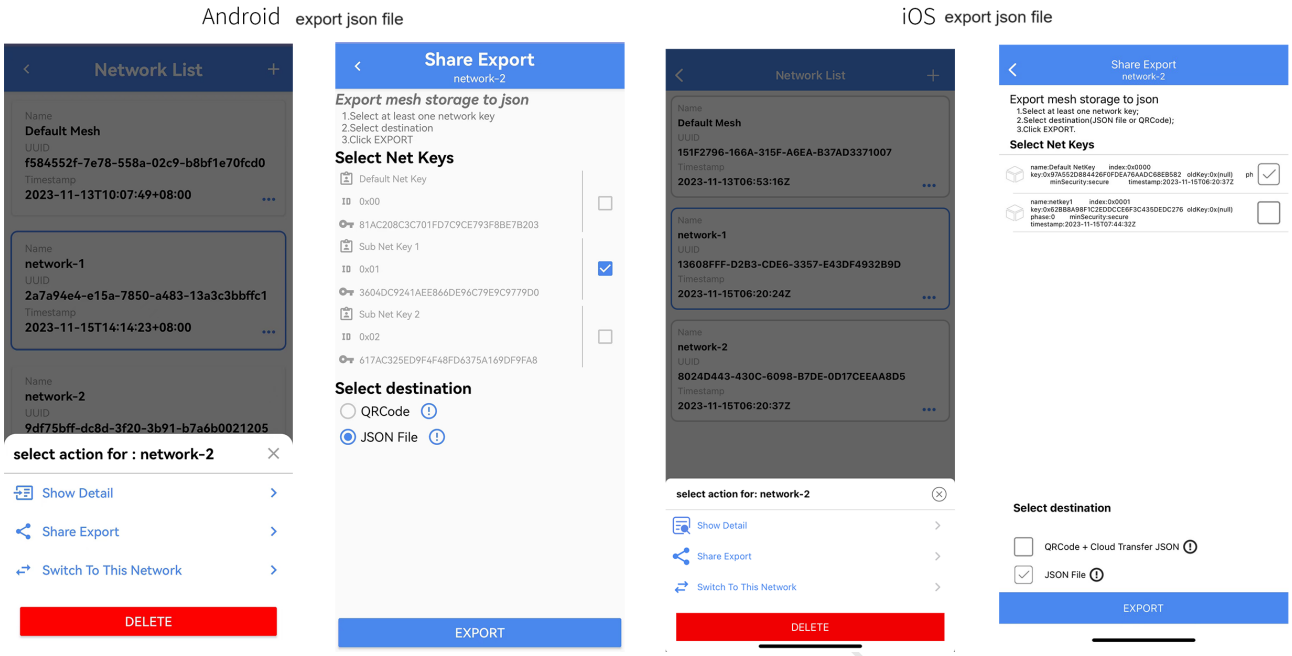


Figure 33.60: Android & iOS export json file

(2) Export by QR Code

At Network List interface, click on a shared Network and select Share Export, select the Net Key to be shared, select QRCode for the sharing method and click on the export button to display the json information in the form of QR code (the QR code has a time limit and will expire after 300 seconds).

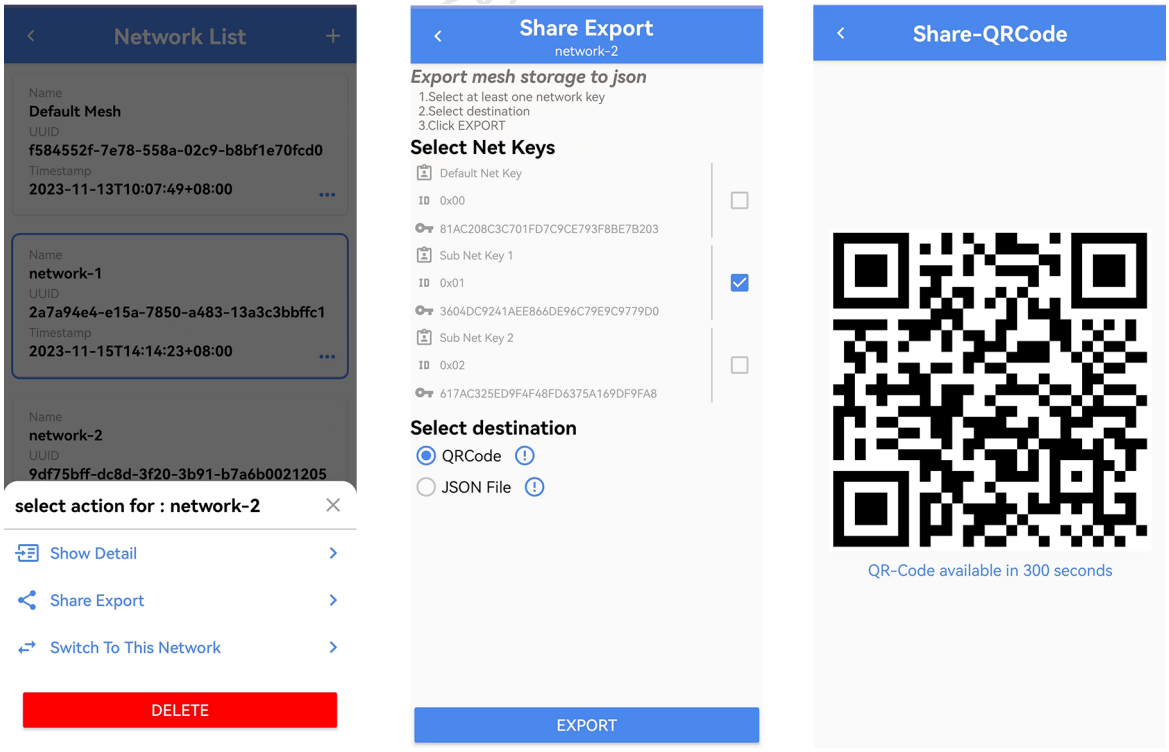


Figure 33.61: Android QR code export

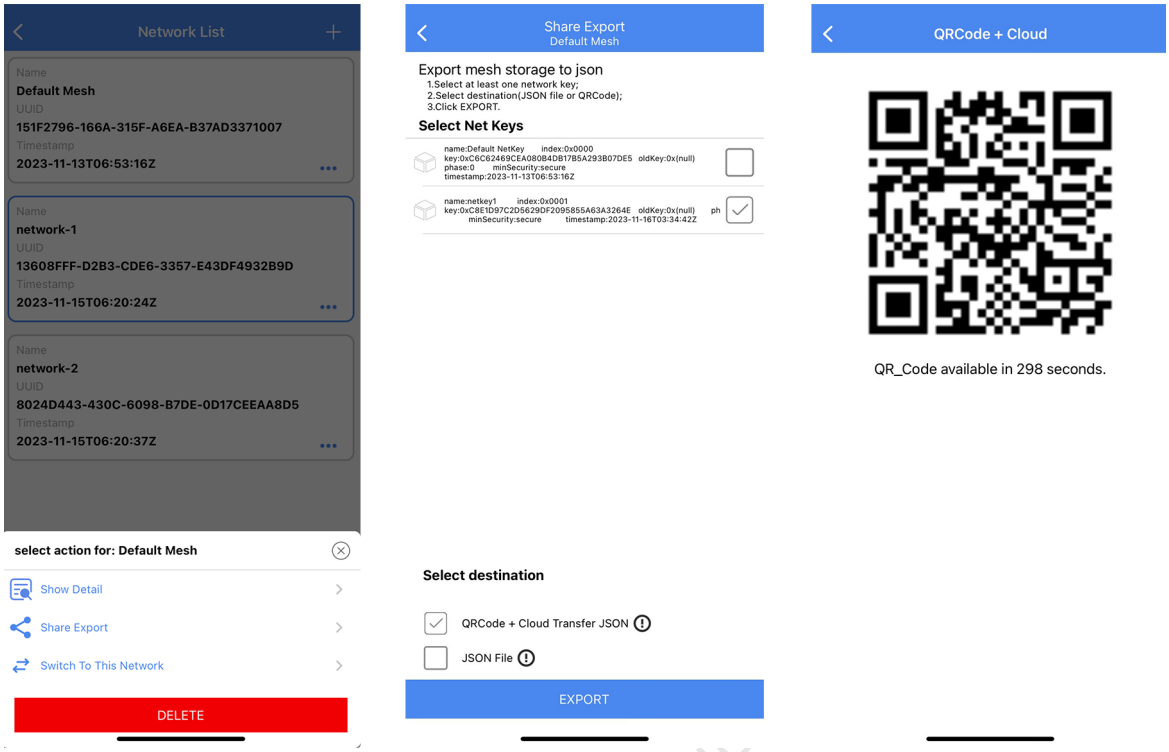


Figure 33.62: iOS QR code export

33.6.1.3 Switch To This Network

Click the specified Network to select Switch to this network to switch to the selected Network (the blue box synchronizes the switch to the specified Network).

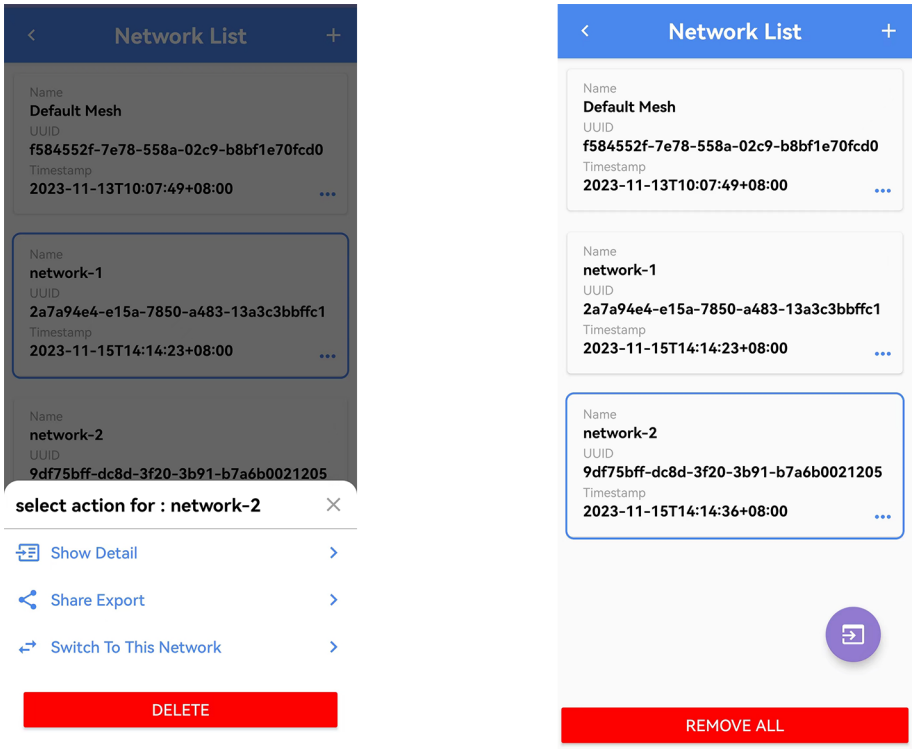


Figure 33.63: Switch to this network

33.6.1.4 Import mesh

Click the Import button at the bottom right of the Network List interface to import a network.

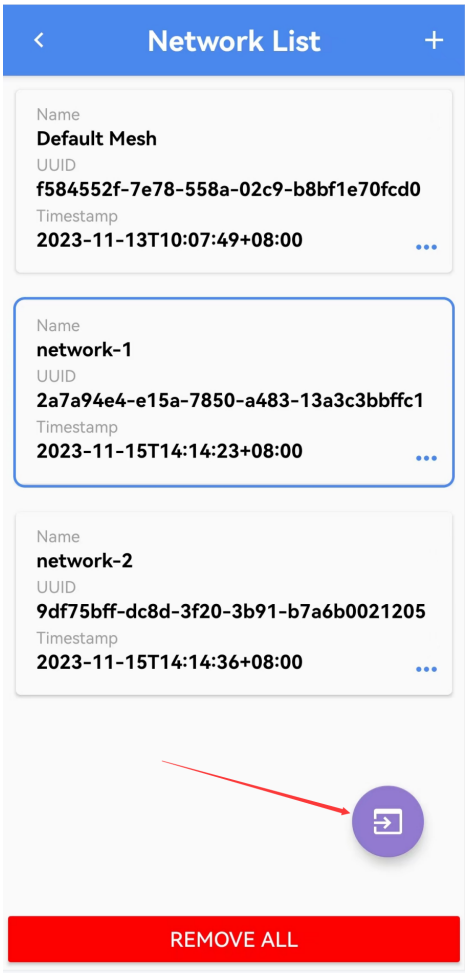


Figure 33.64: Network list interface import button

(1) Import by Json File

Android APP:

At Network List interface, click the Import button in the lower right corner, Select source to choose Json File, Select File to choose the json file to be imported, select the json file, Preview button to preview the information of the json file, Import button to import Network, after successful import, return to Network List interface by default pop-up asking whether to switch to the just imported Network, according to the need not to switch or switch (the previous Network will be retained), you can also set the import automatically switches to the imported Network, the setting path: APP Home – Setting in the right bottom corner – Settings – Share Import Complete Action in the lower right corner, select Auto Switch.

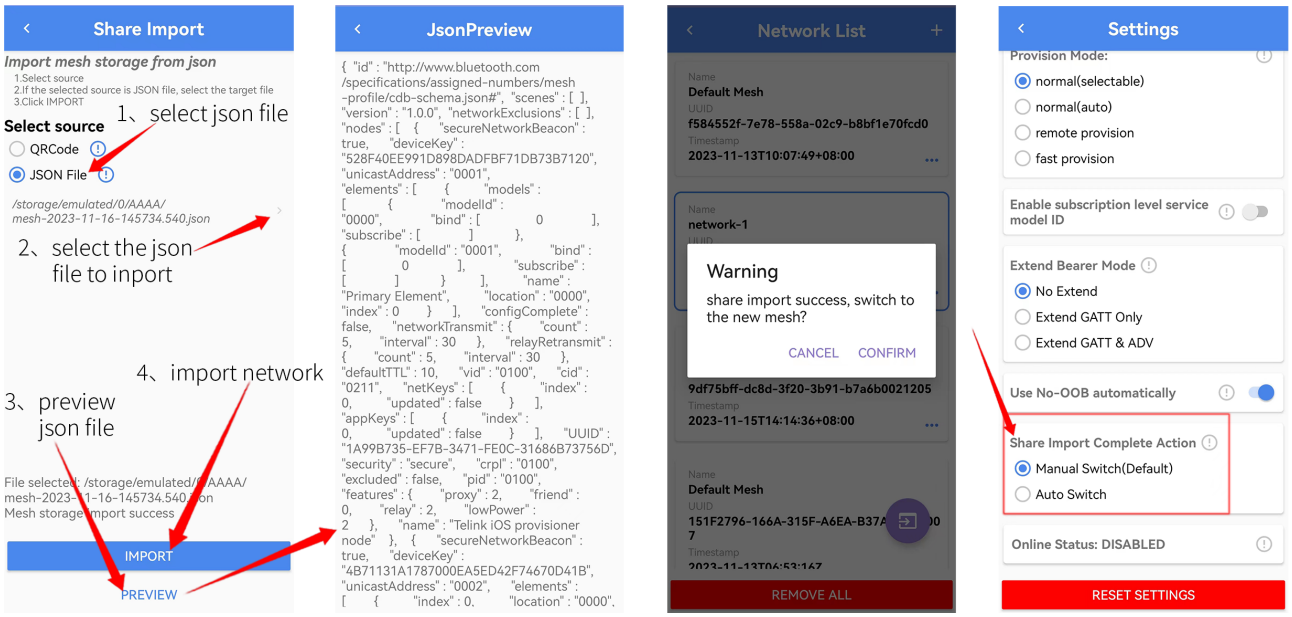


Figure 33.65: Android json file import

iOS APP:

Put the json file into TelinkSigMesh APP through iTunes, click import button in the lower right corner of Network List interface, select the json file inside the select sorce, click Import button to select the json file to be imported, click import, a prompt box will pop up to indicate whether to import the json data or not, clicking cancel not to import network, clicking confirm to import network. After importing network, the default popup will ask whether to switch to the just-imported network, you can not switch or switch according to your needs (the previous network will be retained), you can also set the network to automatically switch to the imported network after importing, the setting path: APP home page – Setting in the right bottom corner – Settings – Share Import Complete Action and select Auto Switch.

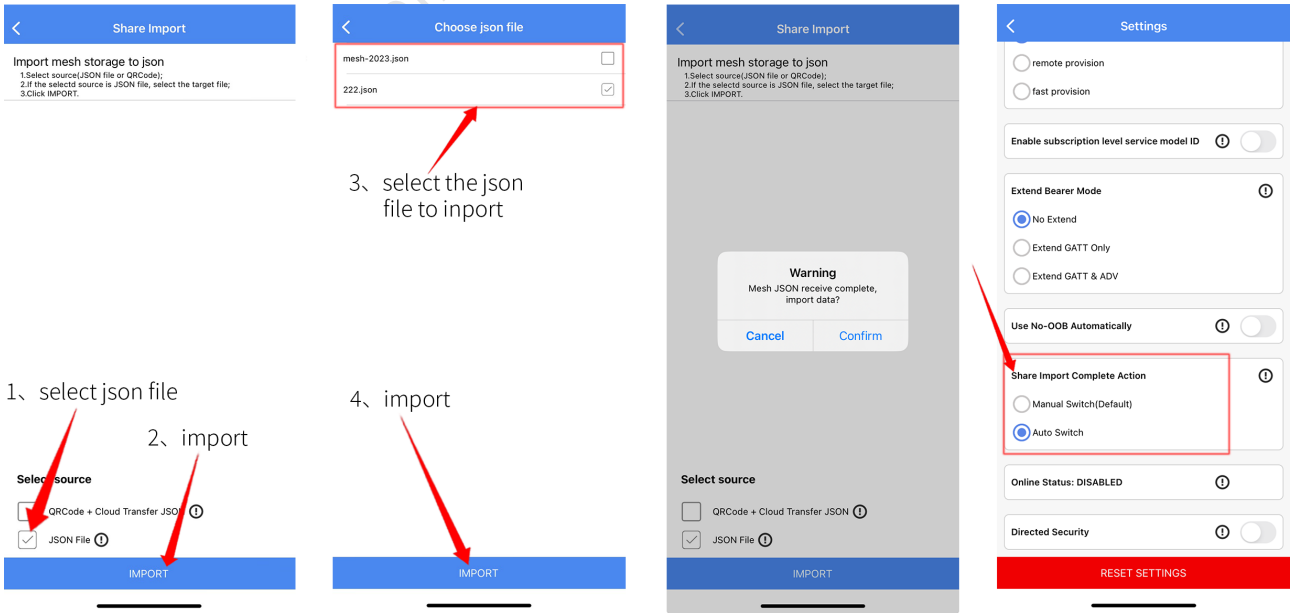


Figure 33.66: iOS json file import

(2) Import by QRCode

Android APP:

At Network List interface, click on the lower right corner of the import button, Select source choose QRCode, click Import button, scan the QR code of the other party's shared Network, it will pop up whether to import Network, after the import is successful, the default automatic pop-up whether to switch to the just imported Network, not switch or switch according to the needs (the previous Network will be retained), you can also set the import automatically switch to the imported Network, set the path: APP home page – Settings in the right bottom corner – Setting – Share Import Complete Action, choose Auto Switch.

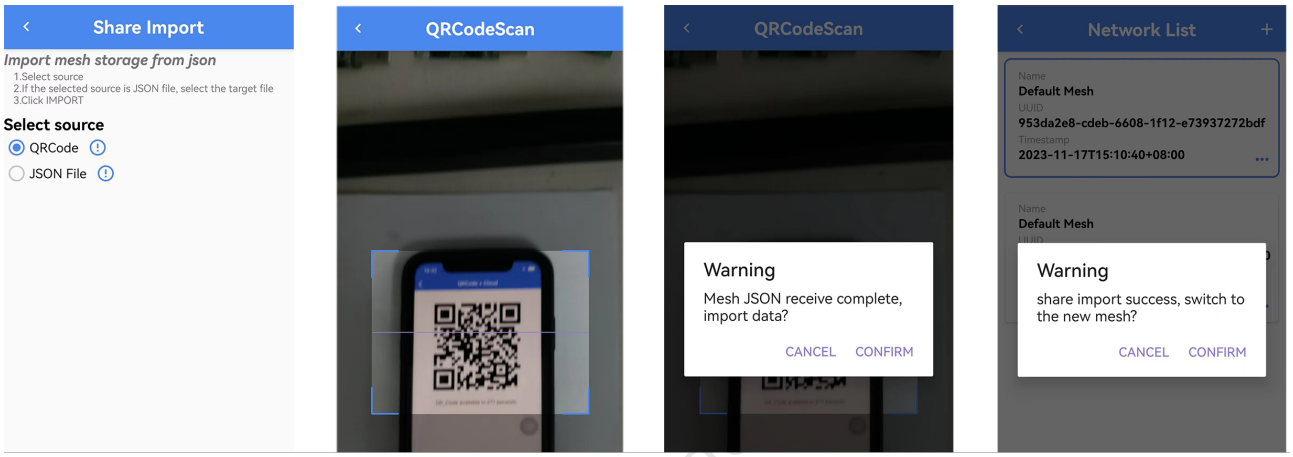


Figure 33.67: Android QR code import network interface

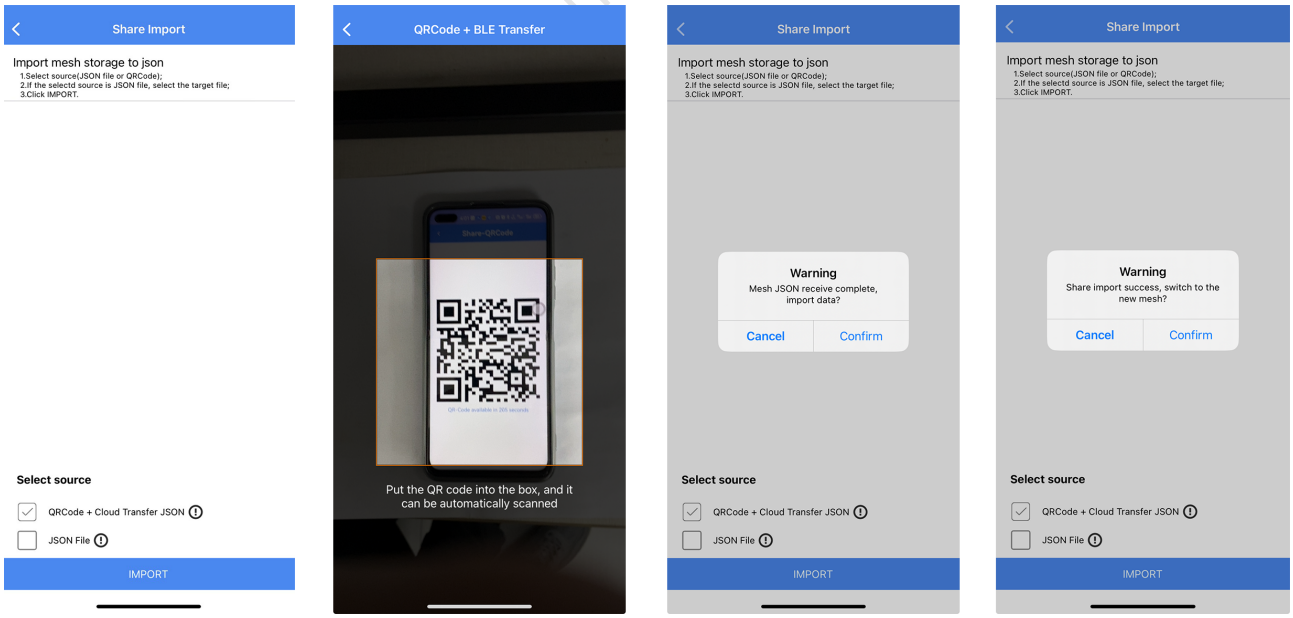


Figure 33.68: iOS QR code import network interface

33.6.1.5 Delete Network

At Network List interface, click to select the network to be deleted, select DELETE, a prompt box will pop up whether to delete the network, you can delete the network according to the need, need to pay attention to is the current network can not be deleted, you can switch to other network and then delete it.

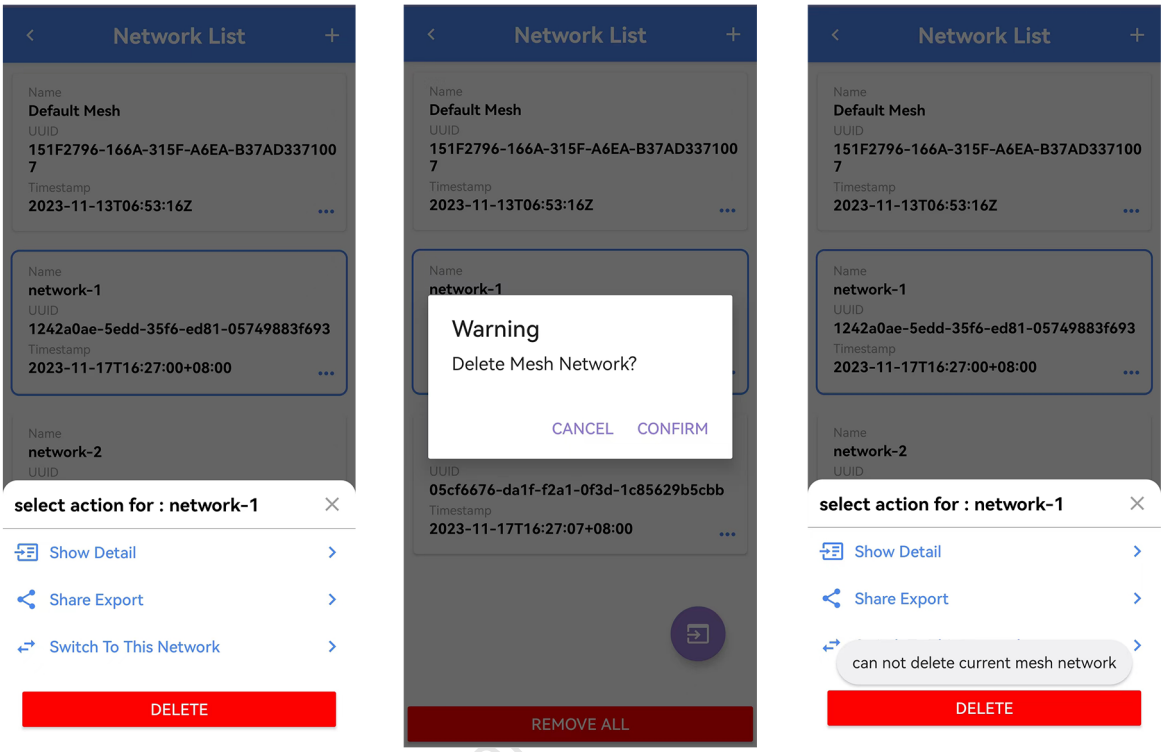


Figure 33.69: Delete the network interface

33.6.1.6 Clear All Network

At Network List, click Remove All at the bottom to clear all Networks.

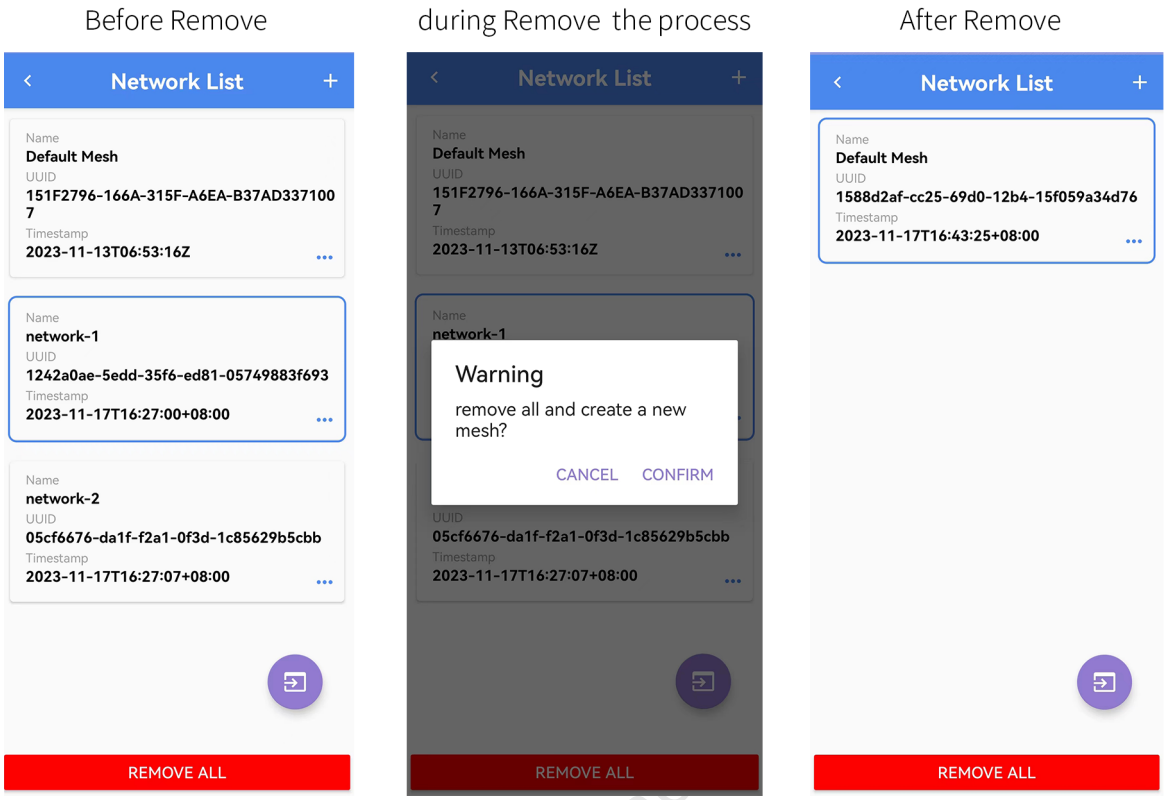


Figure 33.70: Clear all network interface

33.6.2 OOB Database

The OOB Database is used for the App to find the corresponding Auth Value of the device when the device supports static-oob method for provisioning.

When the App is looking for Auth Value, it will use deviceUUID as the key to look up the table from database. If writes OOB data to the the device, you need to enter the corresponding UUID and OOB data in APP in order to normalize networking, and vice versa, the networking will fail.

Currently it supports 16-bit and 32-bit OOB data, which can be written at flash location 77800 as needed.

33.6.2.1 Add an OOB Database Manually

At OOB List interface, click the + sign in the upper right corner and select Manual input to enter UUID and OOB data.

The UUID and OOB queries are as follows:

OOB: Burn 8258 mesh, 8269 mesh, 8278 mesh and other projects, write 16 or 32 bit OOB data at location 77800.

UUID: The device to be networked state with the universal light blue APP to view the device's UUID, specific operation: APP scans the device that needs to obtain the UUID (shown as below) – view the Complete list of 16-bit Service UUID, the yellow area in the figure is UUID.

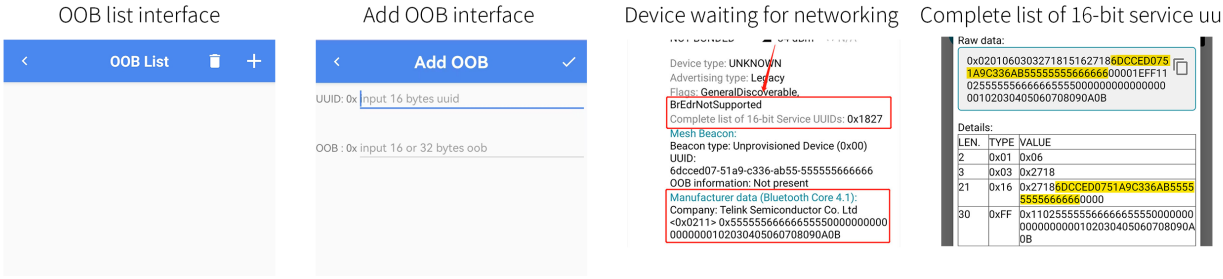


Figure 33.71: Add OOB database manually

33.6.2.2 Import OOB Database via Txt File

Create a new txt document – enter 16 bytes UUID, an empty space to enter 16 or 32 bytes of OOB data and save – at OOB List interface click on the upper-right corner of the + sign to select import from file – select the txt file just saved.

```
// default sample: device uuid + oob(sample MAC: AA 55 55 66 66 66)
6DCCED0751A9C336AB55AA5555666666 11111111222222223333333344444444   16 bytes of OOB data
6DCCED0751A9C336AB55AA5555666666 1111111122222222333333334444444411111111222222223333333344444444   32 bytes of OOB data
```

Figure 33.72: OOB data in TXT format

33.6.2.3 Delete OOB Database

Long press one of the OOB data to delete the OOB data individually, and click the trash can button in the upper right corner to empty all OOB data.

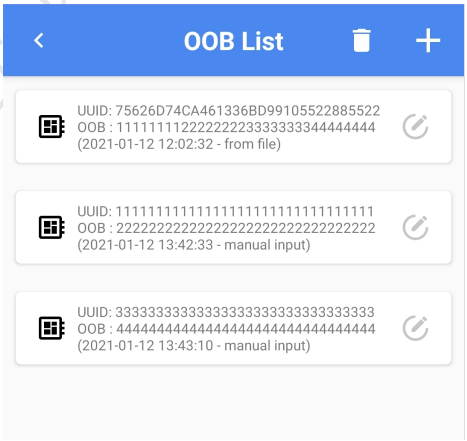


Figure 33.73: OOB List, delete OOB data

33.6.2.4 Use No-OOB Automatically

Use No-OOB Automatically can be added to devices that have OOB data written at the 77800 location but not recorded in the APP (provided that the ENABLE_NO_OOB_IN_STATIC_OOB macro is turned on, and the Use No-OOB Automatically switch is turned on in the APP home page – setting – settings).

33.6.3 Root Cert

When networking, the certificate is verified to determine whether the device is allowed to join the mesh network. When the certificate passes, the network will be successful, but if it does not pass, the network will fail and prompt certificate recordcheck error.

33.6.3.1 Networking by Default Certificate

The V4.1.0.0 version APP comes with a default certificate, when the device uses the default certificate to group the network, it will use the current effective default certificate for verification, you can check it in APP home page – Setting – Root Cert.

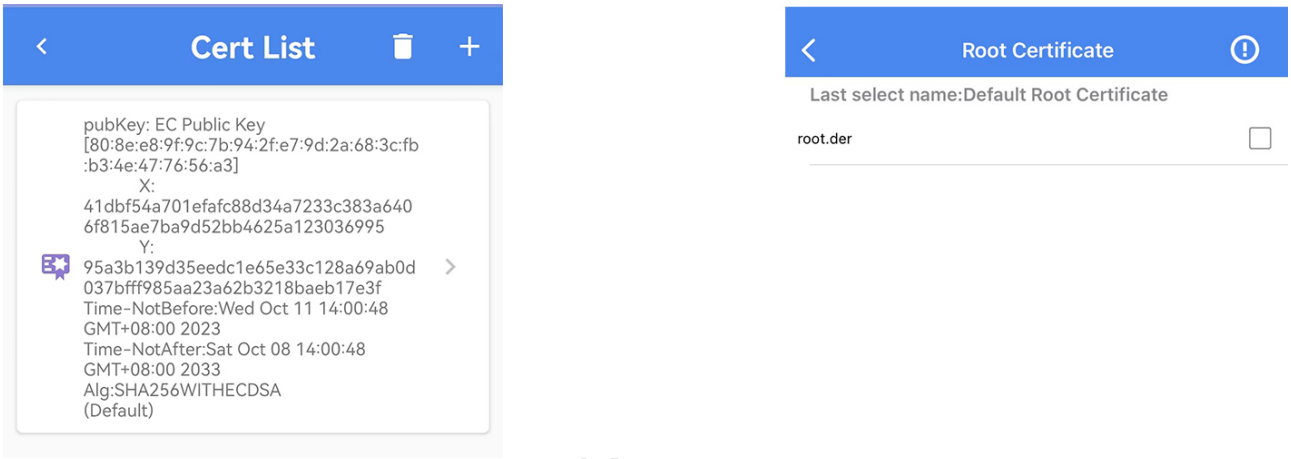


Figure 33.74: Android & iOS cert list interface

Preset conditions:

(1) In the mesh_config.h file turn on the CERTIFY_BASE_ENABLE macro;

(2) The CERT_TYPE in the certify_base_crypto.c file is set to CERTIFY_OOB_BY_DEFAULT_CERT (as shown in the figure below), and the firmware is compiled and burned into the device.

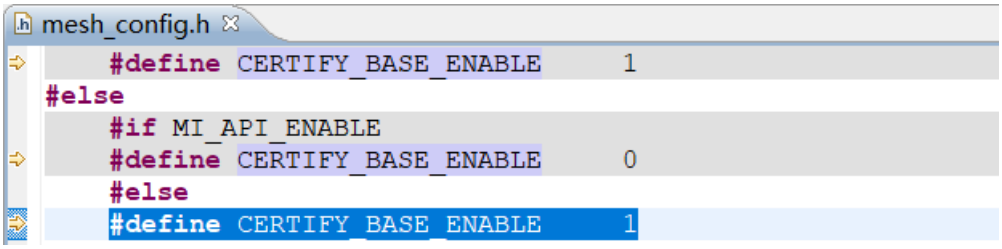


Figure 33.75: Open CERTIFY_BASE_ENABLE

mesh_config.h | certify_base_crypto.c

```
#define CERTIFY_OOB_BY_DEFAULT_CERT         1 // use defaule certificate //check by the rootcert ,and it will not have the oob part
#define CERTIFY_OOB_BY_READING_FLASH        2 // reading certificate from flash.

#if WIN32 // sig_mesh_tool.exe
#define CERTIFY_TYPE  CERTIFY_OOB_BY_DEFAULT_CERT // must 1. because it is for sig_mesh_tool.exe
#else
#define CERTIFY_TYPE  CERTIFY_OOB_BY_DEFAULT_CERT
#endif
```

Figure 33.76: CERT_TYPE set to CERTIFY_OOB_BY_DEFAULT_CERT

Operation Steps:

Manual networking mode:

The certificate icon will be displayed on the upper left corner when the device networking that needs to verify the certificate, and the network will be successfully after the certificate verification passes. If the current device is equipped with other certificates, the APP is validated by other certificates, or the certificate corresponding to the device is deleted, it will prompt that the verification fails: Provision – intermediate cert verify fail.

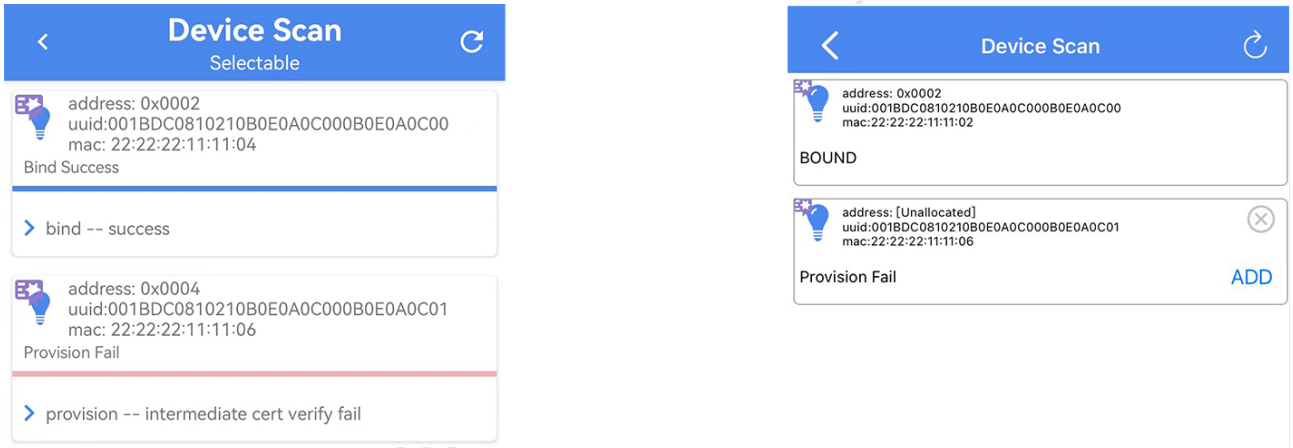


Figure 33.77: Android & iOS manual networking certify tips

Automatic Networking Mode

Compared to devices that do not verify certificates, the devices that need to verify certificates will display the certificate icon in the upper left corner of the device when networking in the Device Scan (Auto) interface. The networking will be successful after the certificate verification passes, while the verification fails will prompt Provision – intermediate cert verify fail.

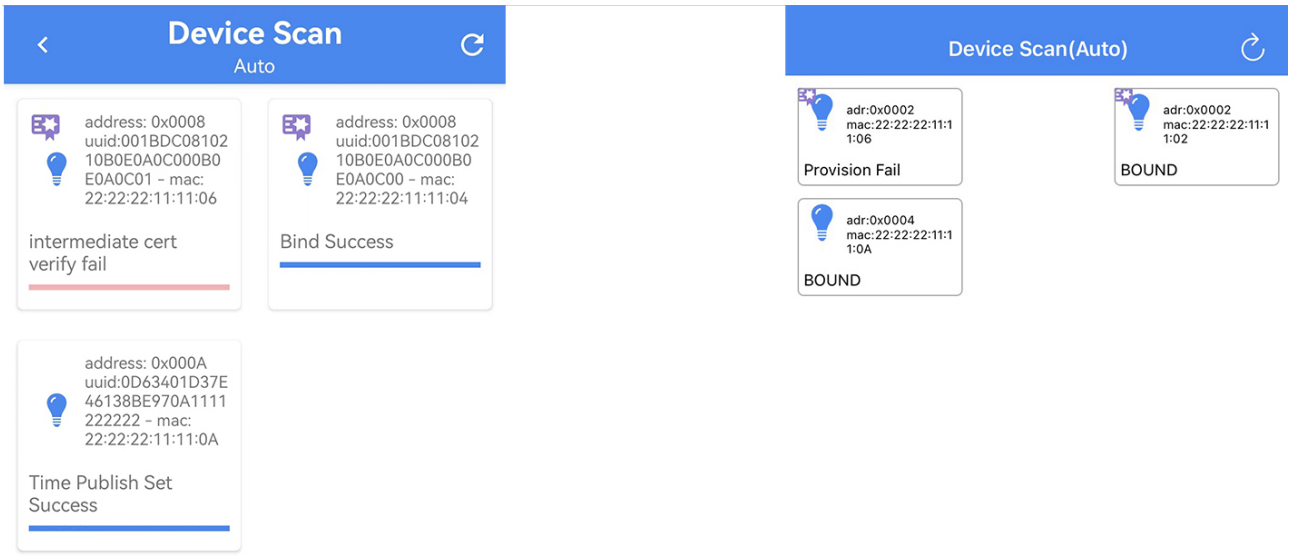


Figure 33.78: Android & iOS auto-networking certify tips

33.6.3.2 Generate and Import New Certificate for Networking

Generate new certificate pre-conditions:

(1) The computer needs to be updated to the latest version of git and TortoseGit.

(2) In the mesh_config.h file, turn on the CERTIFY_BASE_ENABLE macro, and set the CERT_TYPE in the certify_base_crypto.c file to CERTIFY_OOB_BY_READING_FLASH.

Generate a new certificate procedure:

(1) Right-click and select Open Git Bash here in the sdk_git_lab/tools/bash-certifybase directory.

(2) Type ./gen-root.bash – hit enter to generate the "root.der" certificate.

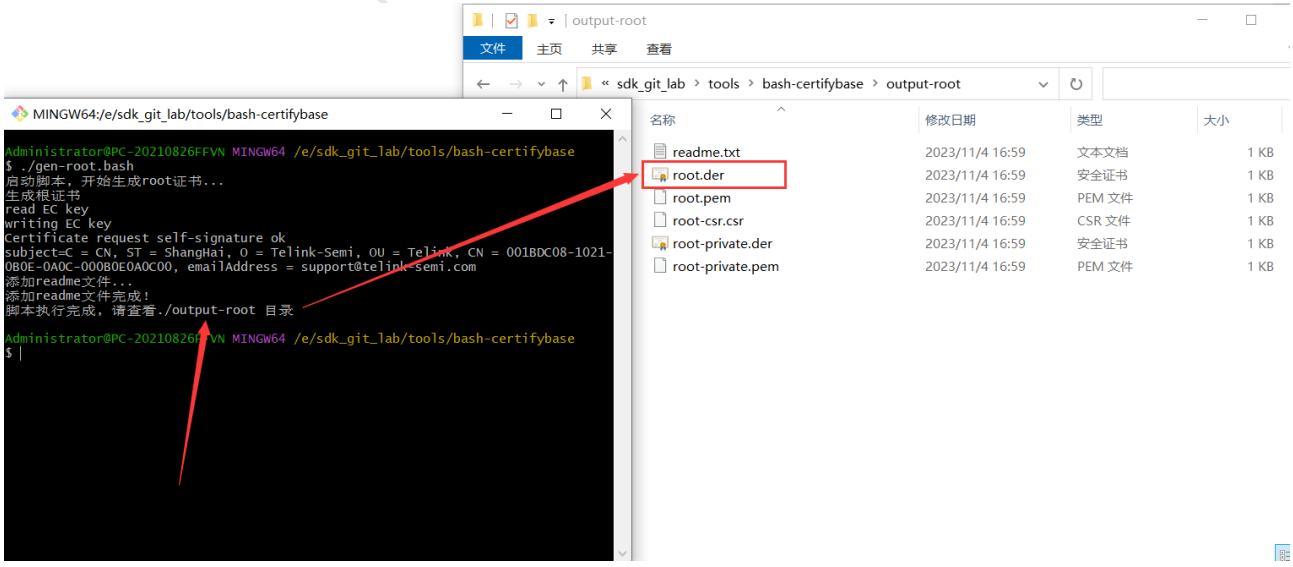


Figure 33.79: Generated "root.der" certificate

(3) APP import root.der certificate

In Android version, import root.der certificate: at APP home page click Setting – root cert – click the upper right corner of the + sign – select the root.der file.

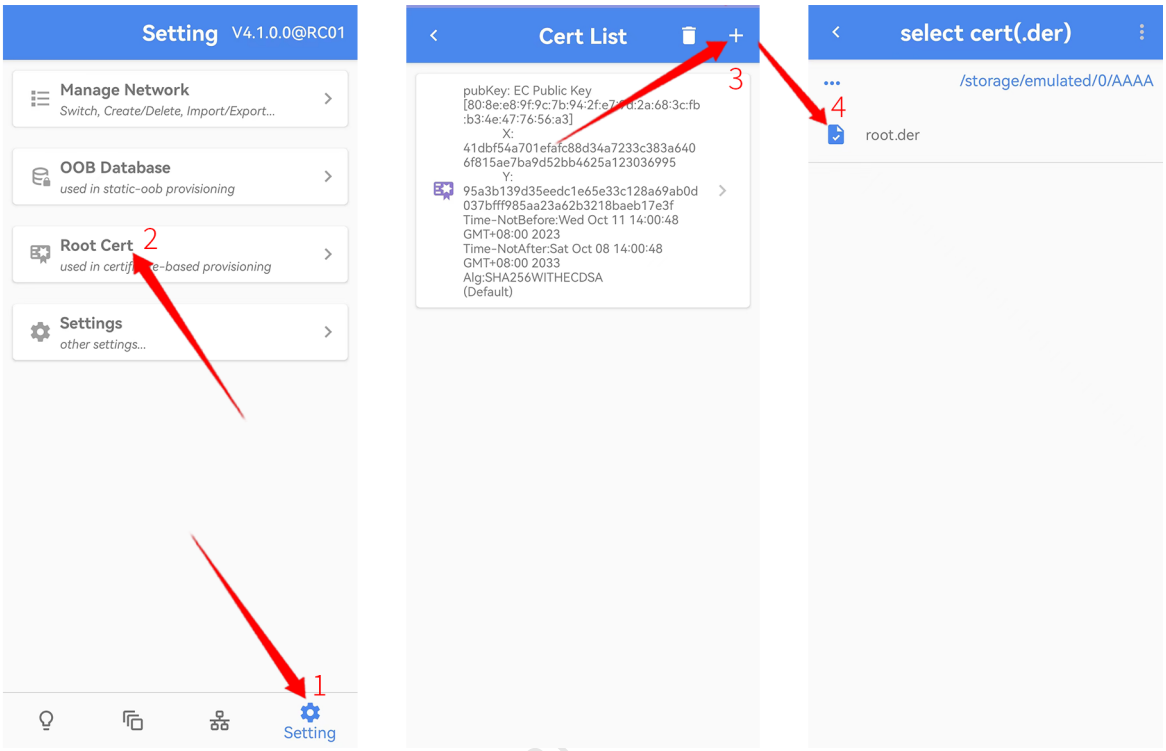


Figure 33.80: Import root.der certificate for Android

Import root.der certificate for iOS version: in iOS system, it needs to put the root.der file into TelinkSigMesh APP through iTunes, and then click Setting – Root Cert – Check the certificate file – Save (Note: you need to save it after checking it to make it effective).

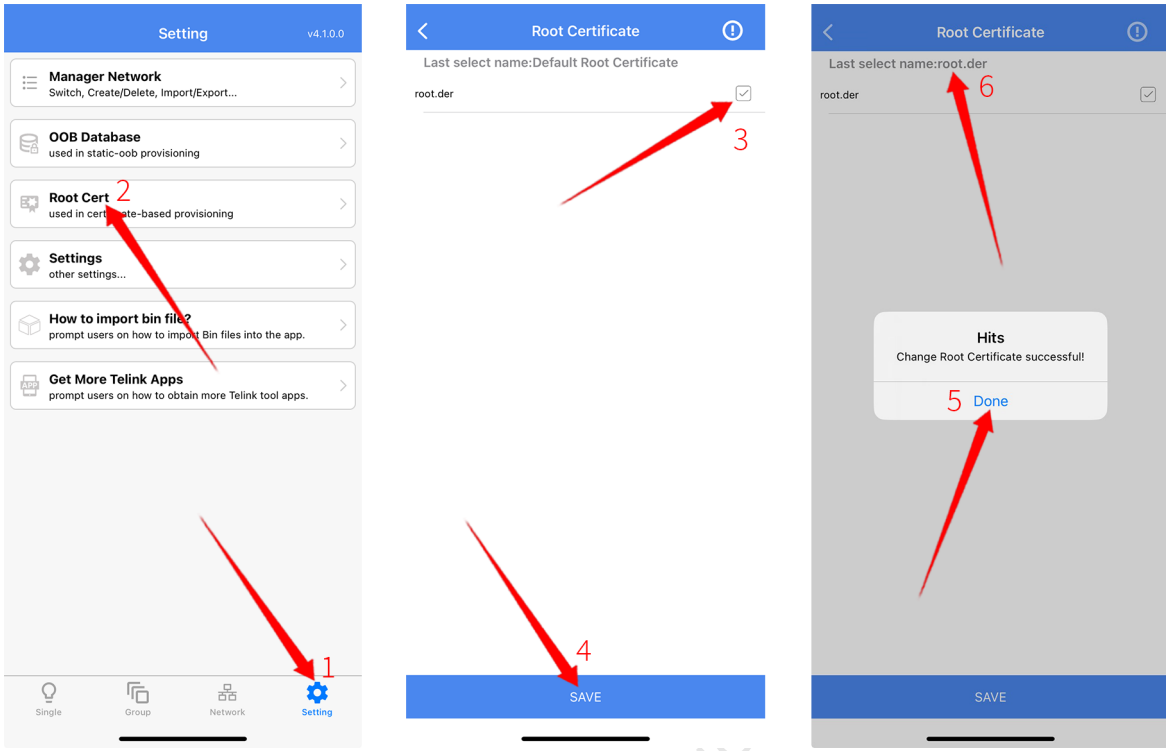


Figure 33.81: Import root.der certificate for iOS

(4) Type ./gen-intermediate.bash – hit enter to generate the intermediate certificate

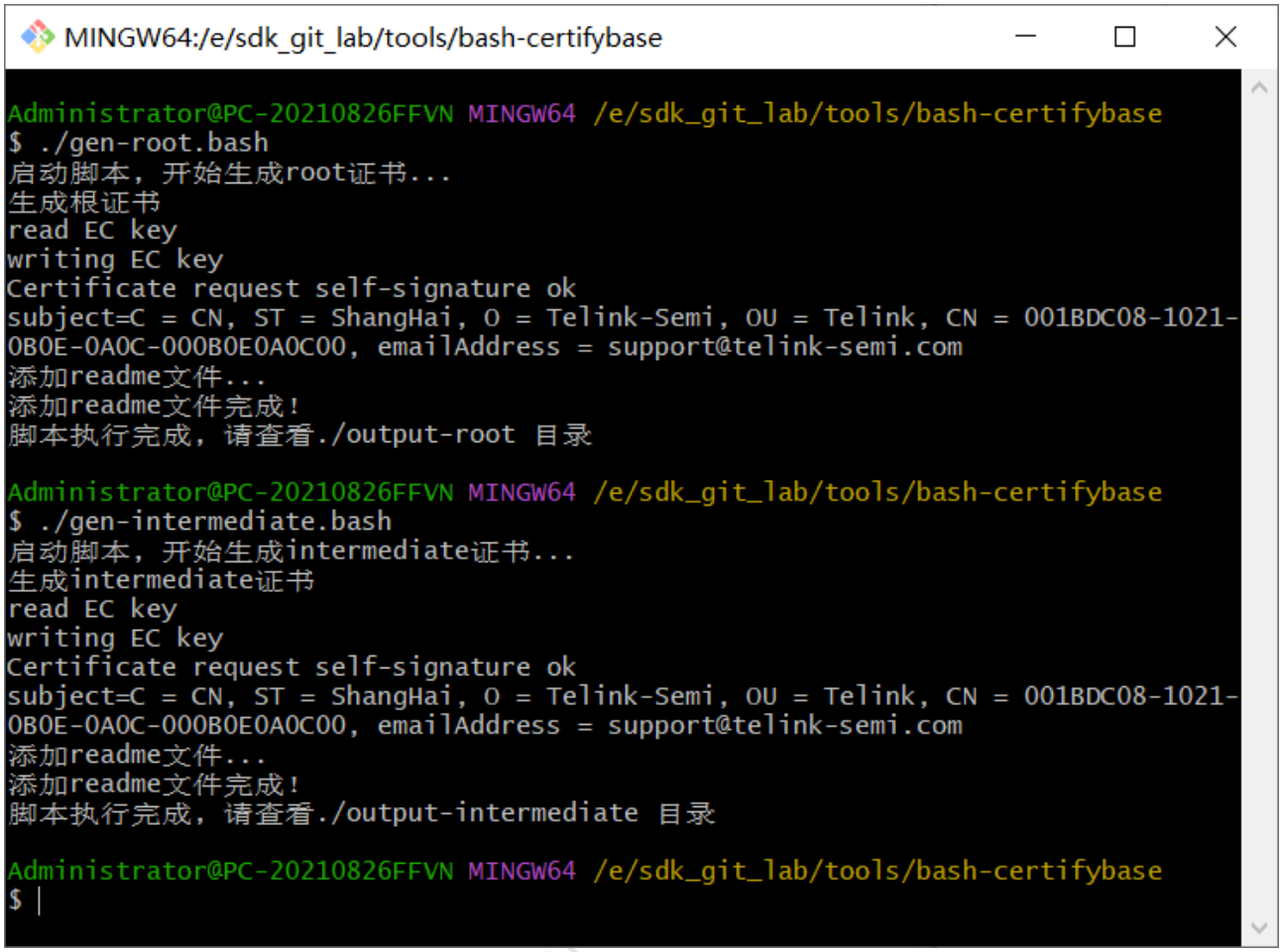


Figure 33.82: Generat intermediate certificates

(5) Edit UUID, CID, PID in gen-device.config

```
[req]
prompt = no
distinguished_name = req_distinguished_name

[req_distinguished_name]
#Country
C=CN
#State
ST=ShangHai
#Organization
O=Telink-Semi
#Organization Unit
OU=Telink
#CN(common name): use device uuid, CID(Company ID, big endianness), PID(Product ID, big endianness) of unprovision node.
CN=001BDC08-1021-0B0E-0A0C-000B0E0A0C01 BCID:0211 BPID:0001
#emailAddress
emailAddress=support@telink-semi.com

[v3_req]
authorityKeyIdentifier = keyid
subjectKeyIdentifier = hash
basicConstraints = CA:FALSE
keyUsage = Certificate Sign, CRL Sign
#TODO: static oob
#2.25.234763379998062148653007332685657680359 = DER:31:7a:6f:16:58:44:72:74:15:10:33:62:5a:fb:c4:f1
certificatePolicies = critical,@pol

[pol]
policyIdentifier = 2.16.840.1.101.3.2.1.48.1
```

Figure 33.83: Change the UUID and the corresponding CID and PID

(6) Type ./gen-device.bash – hit enter to generate a 4K size "device.bin" device certificate.

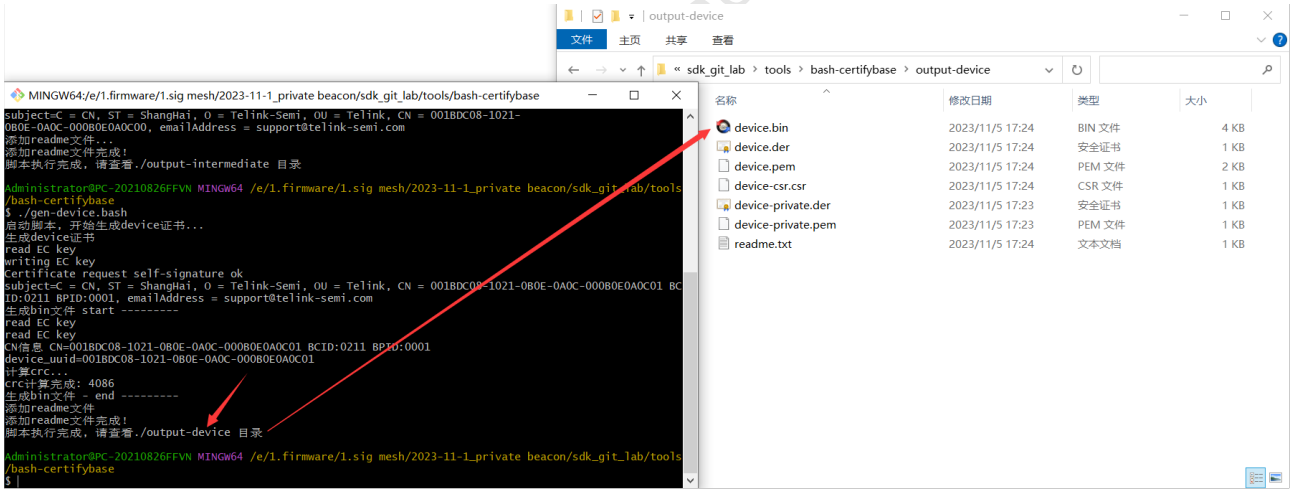


Figure 33.84: Generate device certificate bin file

(7) Burn the device certificate "device.bin" to flash 78000 and use APP to network the device.

33.6.3.3 Switch Certify Base Certificates

Long press the specified certificate, select Set As ROOT Cert to switch to the certificate, the purple certificate icon is the current effective certificate, gray is the saved but not effective certificate.

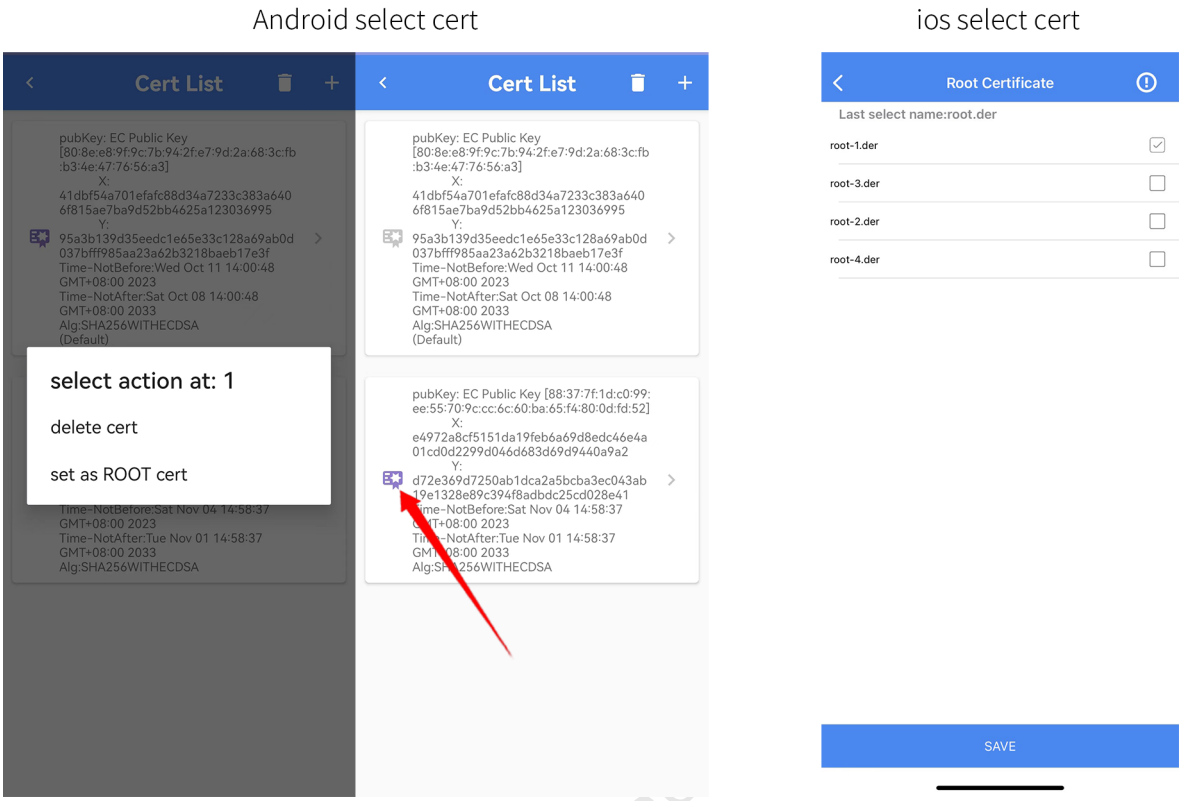


Figure 33.85: Switch the certify base certificate interface

33.6.3.4 Delete Certify Base Certificate

Long press the specified certificate and select Delete Cert to delete the certificate, and click the trash can button in the upper right corner to clear all certificates.

33.6.4 Settings

In the Settings, we can process Enable Log, Enable Private Mode, Enable Provision Mode, Enable Subscription Level Servicemodel Id, Extend Bearer Mode, Use No -Oob Automatically, Share Import Complete Action, Online Status, Reset Settings.

Note: There are differences between Android and iOS, see below for details:

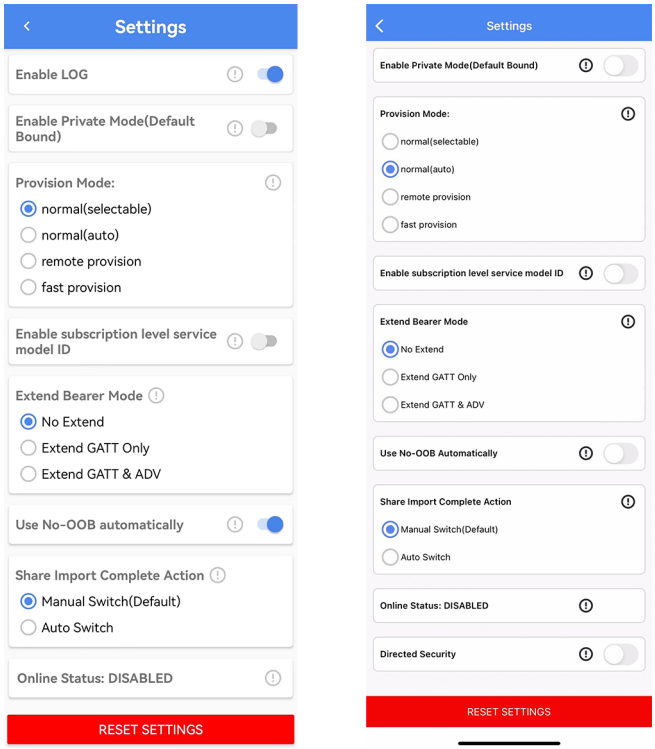


Figure 33.86: Android & iOS Setting/Settings interface

33.6.4.1 Enable Log

Android APP:

Turn on Enable Log to record the log information when controlling the mesh, this item is turned off by default, and can be turned on as needed (please refer to the section 2.5).

iOS APP:

There is no Log switch, the app turns on the Log function by default (please refer to the section 2.5).

33.6.4.2 Enable Privare Mode（Default Bound）

Enable Default Bound is the default binding mode, which needs to be supported by the device. In this mode, the app key binding process can be completed only if the app key add is executed successfully, and the device will automatically bind the app key to all the modes that need to be bound.

33.6.4.3 Provision Mode

In Provision Mode, the default setting is Normal (Selectable) manual networking mode (please refer to section 1.1), Normal (Auto) automatic networking mode (please refer to section 1.2), remote provision and fast provision, which can be turned on according to your needs, and the following is an introduction to remote provision and fast provision.

(1) Remote Provision

Remote provision is to add multi-hop range devices one by one when networking, able to add devices at a longer distance, and with relay function.

Specific operation: Normal (selectable or auto) network a device that supports Remote Provision (i.e., open the MD_REMOTE_PROV macro) – App home page click on the setting – settings – Select Remote Provision in Provision Mode – Click the + sign on the home page of the app to carry out Remote Provision.

Note: Remote provision is turned off by default and requires the device to open the MD_REMOTE_PROV macro.

(2) Fast Provison

In Fast provision batch networking mode, you can network multiple devices that are not networked within the multi-hop range at the same time. The device key used is generated according to certain rules based on the mac address and does not need to be assigned individually. Steps: App home page click on Setting – Settings – Provision Mode and select Fast Provision – at APP home page click on the + sign for Fast Provision.

Note: Fast provision is turned off by default and requires the device to open FAST_PROVISION_ENABLE macro.

33.6.4.4 Enable Subscription Level Service model ID

Enable Subscription Level Servicemodel ID can support to enable the Level control function of the grouping, you can control the Lum Level, Temp Level, Hue Level, Sat Level of the grouping individually. Before grouping nodes, you need to turn on Extend SubscriptionLevel Service Model ID in APP Home – setting – settings (there is a note in section 3.2.3).

33.6.4.5 Enable DLE Mode Extend Bearer

Enable DLE Mode Extend Bearer is an option for sending long packets, requires device support. When enabled, the maximum length of short packets at the access layer is changed from 11 to 225.

33.6.4.6 Online Status

Online Status allows you to check whether the current connected-only device supports the Online Status function and report the status when the device status changes.

33.6.4.7 Reset Settings

Reset settings restores all options in the settings screen to their default state.

34 Common API

This chapter introduces the commonly used APIs for mesh SDK development. For an introduction to the parameters of the APIs listed in this chapter, please refer to the comments on the API functions in mesh SDK.

34.1 Provisioning Callbacks

34.1.1 Provision Event Callback

34.1.1.1 void mesh_node_prov_event_callback(u8 evt_code)

This function is callback function for provision event of mesh node in each provision state.

34.1.1.2 u8 is_provision_success()

This function get whether the node is at provision success state.

34.1.1.3 rf_link_light_event_callback (u8 status)

This function is callback function to define LED indication event, such as how to do LED indication when provision success.

34.1.2 Provisioning Message Handle

34.1.2.1 PB_ADV

void mesh_node_rc_data_dispatch(pro_PB_ADV *p_adv)

This function is for a unprovisioned device to be provisioned through PB_ADV bearer, and handle all messages during provision flow.if a provison message need to be assembled, it was assembled in mesh_provison_process() before.

34.1.2.2 PB_GATT

void dispatch_pb_gatt(u8 *p ,u8 len)

This function is for a unprovisioned device to be provisioned through PB_GATT bearer, and handle all messages during provision flow.if a provison message need to be assembled, it was assembled in pkt_pb_gatt_data() before.

34.2 Proxy Server API

34.2.1 Provision Service

34.2.1.1 Int pb_gatt_Write (void *p)

This function server to process Proxy PDU of provision service from GATT master, such as cell phone.

34.2.2 Proxy Service

34.2.2.1 Int proxy_gatt_Write(void *p)

This function server to process Proxy PDU of proxy service from GATT master, such as cell phone.

34.3 Configuration Callbacks API

This section introduce APIs of all opcodes of configuration server and client model.

Most of them are located within config_model.c. and processing callback function when receiving a opcode are defined by mesh_cmd_sig_func[].cb.

Take opcode of "APPKEY_ADD" for example:

34.3.1 Int mesh_cmd_sig_cfg_appkey_set()

```
* @brief       This function will be called when receive the opcode of "Config AppKey Add"
 * @param[in]   par     - parameter of this message
 * @param[in]   par_len - parameter length
 * @param[in]   cb_par  - parameters output by callback function which handle the opcode
 ↪  received.
 * @return      0: success; others: error code of tx_errno_e
 * @note

int mesh_cmd_sig_cfg_appkey_set(u8 *par, int par_len, mesh_cb_fun_par_t *cb_par)
```

34.4 model_enable

This section introduce APIs of all opcodes of other SIG server and client model.

all processing callback function when receiving a opcode are defined by mesh_cmd_sig_func[].cb.

34.4.1 MD_SAR_EN

SAR Configuration Server model and client model. SAR means segmentation and reassembly.

34.4.2 MD_ON_DEMAND_PROXY_EN

On Demand Private Proxy Server model and client model.

34.4.3 MD_OP_AGG_EN

Opcodes Aggregator Server model and client model.

34.4.4 MD_LARGE_CPS_EN

Large Composition Data Server model and client model.

34.4.5 MD_SOLI_PDU_RPL_EN

Solicitation PDU RPL Configuration Server model and client model. RPL means Replay Protection List.

34.4.6 MD_DF_CFG_SERVER_EN and MD_DF_CFG_CLIENT_EN

directed forwarding server model and client model.

34.4.7 MD_SBR_CFG_SERVER_EN and MD_SBR_CFG_CLIENT_EN

subnet bridge server model and client model.

34.4.8 MD_REMOTE_PROV

Remote Provisioning Server model and client model.

34.4.9 MD_PRIVACY_BEA

Mesh Private Beacon Server model and client model

34.4.10 MD_BATTERY_EN

Generic Battery Server and client model

34.4.11 MD_LOCATION_EN

Generic Location Server and client model

34.4.12 MD_LEVEL_EN

Generic Level Server and client model

34.4.13 MD_DEF_TRANSIT_TIME_EN

Generic Default Transition Time Server and client model

34.4.14 MD_POWER_ONOFF_EN

Generic Power OnOff Server model and client model

34.4.15 MD_SCENE_EN

Scene Server and client model

34.4.16 MD_TIME_EN

Time Server model and client model

34.4.17 MD_SCHEDULE_EN

Scheduler Server model and client model

34.4.18 MD_SENSOR_EN

Sensor Server model and client model

34.4.19 MD_MESH_OTA_EN

device firmware update Server model and client model

Mesh Binary Large Object Transfer Server model and client model

34.4.20 MD_LIGHTNESS_EN

Light Lightness Server model and client model

34.4.21 MD_LIGHT_CONTROL_EN

Lighting control models model and client model

34.4.22 LIGHT_TYPE_CT_EN

Light CTL Server model and client model

34.4.23 LIGHT_TYPE_HSL_EN

Light HSL Server model and client model

34.4.24 LIGHT_TYPE_XYL

Light xyL Server model and client model

34.4.25 LIGHT_TYPE_POWER

Generic Power Level Server model and client model

34.4.26 MD_PROPERTY_EN

Generic User Property Server model and client model

Generic Admin Property Server model and client model

Generic Manufacturer Property Server model and client model

Generic Client Property Server model and client model

34.5 Light CT and RGB PWM Output API

34.5.1 Void light_dim_refresh(int idx)

Refresh the light status once the current lightness, CT and HSL, etc is changed, include during transition process.

In this function, user can get the lightness and CT, HSL, etc, and then user can redefine how to driver PWM output which is depend on hardware, if it is needed.

34.6 Vendor Model Client and Server API

This section introduce APIs of all opcodes of other Vendor server and client model.

all processing callback function when receiving a opcode are defined by mesh_cmd_vd_func[].cb.

VENDOR_OP_MODE_SEL is set to VENDOR_OP_MODE_DEFAULT as default. Include model of VENDOR_MD_LIGHT_S which is server model and VENDOR_MD_LIGHT_C which is client model

34.7 Firmware Update and Blob Transfer API

This section introduce APIs of all opcodes of firmware update and blob transfer server and client model.

all processing callback function when receiving a opcode are defined by mesh_cmd_sig_func[].cb.

And they are enable by set MD_MESH_OTA_EN to 1.

35 QA

This section contains some frequently asked questions.

Q1. Where is the callback for successful networking?

Inside the LED indication callback function rf_link_light_event_callback(),use the LGT_CMD_SET_MESH_INFO branch. LGT_CMD_SET_MESH_INFO which is the same as LGT_CMD_PROV_SUC_EVE, indicates the flashing event on the light node side, after the provision is successful.

Q2. How to determine if a node is in a provision success state?

is_provision_success(); Returns 1 to indicate that the app has been provisioned. Note that this refers to the provisioning process, not include the subsequent app key add and key bind processes.

There is no single way to judge the completion of a key bind that is applicable to all scenarios. This is because logically, it is not necessary to perform app key add and key bind immediately after the app networking process. If we take our app process as an example, after the app networking process is completed, app key add and key bind will be performed immediately after the app key add and key bind, and at this point, we can judge node_binding_tick inside main_loop() as follows. tick, as shown below:

```
void main_loop ()
{
    ......

    if(is_provision_success() && node_binding_tick && clock_time_exceed(node_binding_tick,
    ↪  3*1000*1000)){
        // key bind success here
    }

    ......
}
```

LPN is the judgement by the above logic.

Q3. Can I use the group number as the destination address when deleting or kicking out nodes?

To remove a device from the network, it is necessary to kick out one by one. The reason is that kick out sends the command "NODE_RESET", which belongs to the configuration model and requires device key to encryption and decryption, and the device key of each node is not the same, so there is no way to send the command using multicast, unless the client customises the vendor command to do the remove action.

Q4. Do 16k and 32k retention need to switch cstartup.S and boot.link files?

Not required. See the "Startup file cstartup.s and link file boot.link" section for more information.

Q5. After grouping the onoff model of a node and sending onoff set or lightness set command to the destination address 0xc000, the node is controlled normally but the vendor command is not, what is the reason?

See "Grouping Features and Share-modell" section for a detailed description of the theory analysis.

The UI operation of the sig_mesh_tool.exe only sends "Config Model Subscription Add" to the onoff model by default, so we need to send Subscription Add to the vendor model via INI, etc. in addition to the onoff

model. If you want to add private practices, such as sharing the group number between onoff model and vendor model, you can turn on SHARE_ALL_LIGHT_STATE_MODEL_EN. See the code for this macro.

Q6. What is the reason for the error "get ut tx buffer failed: tx segment busy" when sending long packets (segment) continuously?

When a long packet is sent, it needs to be segment. If the destination address is a unicast address, after sending all the sub-packets, you need to wait for the receiver to reply block Ack to see if all the segment packets have been received, and if not, the missing packet will be retransmited. Until the other party has finished receiving or exceeded the maximum number of retransmissions.

Currently, the SDK only supports one state machine to manage the segment process, so when the previous segment sends a long packet without completing it, the send packet function sends a long packet again, and it will report an error "get ut tx buffer failed: tx segment busy".

Even if the destination address is a multicast address, you need to wait until all the sub-packets have been sent before you are allowed to send the next long packet. Because the process of sending packets is to send a packet first, and when the packet is finished (about 200ms), then send the next packet, until all the packets are sent, then the segment busy will be cleared to zero. The reason why we don't continuously press the packets into the send fifo is that our tx buffer is not set that big, and if we continuously press it, the buffer will probably be insufficient to cause an error.

The suggested solution is to judge the return value of the send packet function, if it is not 0, wait for some time before sending, or call is_busy_mesh_tx_cmd() before sending to judge if it is currently in busy state. Or enable the private extended broadcast packet mode, see "Telink Customized Mode for Sending Mesh Messages via Extended Broadcast Package extend_adv" for details.

Q7. How to get the rssi of the current message?

In a message callback function, such as mesh_cmd_sig_g_onoff_set(), to read the global variable rssi_pkt to get rssi of the current message, which is assigned a value before calling app_event_handler().

Q8. How to get the ttl of the current message?

In a message callback function, such as mesh_cmd_sig_g_onoff_set(), to read cb_par->p_nw.ttl to get the ttl of the current message.

Q9. Do the step resolution and number of steps in the transition time commands, such as on/off command, define the amount of change each time the light fades?

No, Transition Time just defines how long it takes for the light to finish changing. The amount of each transition is customised by the hardware, if you need to change it, just change the value of LIGHT_ADJUST_INTERVAL.

Q10. Why add AS_PWM_SECOND type inside driver?

In the mesh SDK, the PWM property only retains AS_PWM and adds AS_PWM_SECOND, because we hope that when the customer modifies the PWM port of the light to another IO port, the customer only needs to modify the GPIO, and the rest is done automatically by the code, no need to configure the PWM ID, invert property and so on.

Since some pins support two PWM IDs, such as GPIO_PC1, GPIO_PC4, GPIO_PC5 of B85m. Therefore, when customers configure the PWM attribute in mesh SDK, they need to check the GPIO table in the datasheet, if the GPIO does not belong to one of these 3 pins, then all of them will be set to AS_PWM, because all of

them have only one PWM index for use. If the GPIO belongs to one of these three, then if you select the first PWM index, then set it to AS_PWM, if you select the second PWM index, then set it to AS_PWM_SECOND.

Q11. Which compilation option should I choose for low-power devices?

For lower power product, if you want to only receive commands during the provision process or during receive mode which is actively entered, and other times just need to send command, it is recommended to use the 8258_mesh_Switch compilation option.

If you want to receive data at irregular intervals, i.e. another node may send data from time to time and require a low-power node to receive the data, then you need to use the LPN mode defined in the SIG mesh spec, which corresponds to the compilation option 8258_mesh_LPN, and which requires a Friend node. You can also consider the private spirit LPN compilation option, which does not require a Friend node, but requires that the sender sends commands continuously for 1 second at short intervals. This is because spirit LPNs wake up periodically for a short period of time to receive data. The power consumption is higher than standard mesh LPN. For details, please refer to Spirit LPN.

Q12. Suggested improvement for flash over 192KB?

When the firmware is optimised and still exceeds 192KB, the first recommendation is to replace the chip with a flash of 1MB.

If you don't want to change it, and you don't exceed it by much, say a dozen k sizes, you can consider the following optimisation directions at this point:

- Rearrange the flash map and remove the parameter areas corresponding to some unused functions (but evaluate whether they will be used later), such as FLASH_ADR_MD_TIME_SCHEDULE, which is not used by default. FLASH_ADR_MD_LIGHT_LC, FLASH_ADR_MD_SENSOR, FLASH_ADR_MD_LIGHT_HSL, FLASH_ADR_MD_PROPERTY(FLASH_ADR_MD_DF_SBR), FLASH_ADR_MD_G_POWER_ONOFF, FLASH_ADR_MD_SCENE, FLASH_ADR_MESH_TYPE_FLAG, FLASH_ADR_MD_MESH_OTA, FLASH_ADR_MD_MISC_PA
- The md5 algorithm for removing device uuid, the specific function is uuid_create_md5_from_name of uuid_create_by_mac(), which is about 2k.

Note:

> After rearranging the flash, you need to modify FW_SIZE_MAX_K and check if factory_reset() needs to be adjusted.

Q13. How do we get a node's provisioning information?

The provisioning information mainly contains netkey, device key, appkey, iv index, unicast address, which includes:

```
typedef struct{
    u8  net_work_key[16];
    u16  key_index;
    union{
        mesh_ctl_fri_update_flag_t prov_flags;
        u8  flags;
    };
    u8  iv_index[4];
    u16  unicast_address;
```

```
}

u8 device_key[16];

u8 app_key[16];
```

We can get netkey, device key, appkey from variable mesh_key; get iv index from iv_cur in iv_idx_st; get unicast address from ele_adr_primary.

Q14. How do we send a customized advertise packet?

To send a customized advertise packet, you can change the data where p points to in the gatt_adv_prepare_handler function, such as pre_set_beacon_to_adv(), which returns 1 to send, or returns 0 indicating of no data to send.

Q15. Why is the maximum number of schedulers 16?

Because the index field size of Scheduler in mesh model spec occupy only four bits, so the maximum number of schedulers is 16. Please refer mesh_model_spec for details, such as section 5.2.3.4 of "MshMDL_v1.1.pdf". Note that each node supports 16 schedulers, rather than the entire mesh network having only 16 schedulers, so it is also sufficient.

Q16. How do we delete scheduler?

The scheduler can be removed by sending the Scheduler Action Set command and setting the action field of the Scheduler to SCHD_ACTION_NONE which is 0x0F.

```
typedef struct{
    u64 valid_flag_or_idx   : 4;    // flag: when save;  index: in message
    u64 year        : 7;
    u64 month       : 12;
    u64 day         : 5;
    u64 hour        : 5;
    u64 minute      : 6;
    u64 second      : 6;
    u64 week        : 7;    // bit0 means monday,
    u64 action      : 4;
    u64 trans_t     : 8;    // transition time
    u16 scene_id;
    u8 rand_hour;
    u8 rand_min;
    u8 rand_sec;
    u8 rsv;
}scheduler_t;
```