
沁恒 MESH APP 管理配网 应用手册

V1.1
2022 年 7 月 14 日

目录

1. 工程文件介绍	3
1.1 配网信息存储配置	3
2. 应用层程序介绍	4
2.1 应用层初始化	4
2.2 收发数据接口	5
2.2.1 接收手机数据	5
2.2.2 接收 MESH 数据	5
2.2.3 向手机上报数据	6
2.2.4 通过 MESH 网络发送数据	7
3. 配网与网络管理	8
3.1 手机配置节点入网	8
3.1.1 配网状态广播	8
3.1.2 配网流程	9
3.2 管理网络	11
3.2.1 查询节点在线状态	12
3.2.2 数据传输	14
3.2.3 删除节点	15
3.2.4 订阅与组控制	16
3.2.5 同一个手机管理多个网络方案	17
4. OTA 升级	18
4.1 代码区域分配	18
5. 低功耗与朋友节点	19

1. 工程文件介绍

此应用为固定 BLE LIB，可备份升级 MESH LIB 应用，由三个工程与一个 BLE LIB 文件组成，分别为：

adv_vendor_self_provision_IAP，负责判断当前是否需要搬运，需要则搬运备份区的代码到用户区并且跳转到用户区运行。

adv_vendor_self_provision_JumpIAP，跳转到 IAP 程序的指令

adv_vendor_self_provision_with_peripheral，应用程序主体，也是可升级程序

CH58xBLE_ROM_MESH.hex (wchble_rom_mesh.hex)，固定 BLE LIB 库文件，不同芯片文件不同

烧录固件时需要使用 AssemblingFileTool.exe（在我司官网下载名为 WCH_AssemblingFileTool 的压缩包）将上述三个工程编译出来的目标文件和固定 BLE LIB 库文件合并为一个 BIN 文件，将最终的 BIN 文件烧录到单片机。

手机 APP 可在我司官网下载 WCH_Mesh_Android 安装。

1.1 配网信息存储配置

配网信息存储地址 CONFIG_MESH_NVS_ADDR_DEF 默认为 0，即 dataflash 的首地址，共使用 CONFIG_MESH_SECTOR_COUNT_DEF 默认为 3 块扇区，每块大小 CONFIG_MESH_SECTOR_SIZE_DEF 默认为 4096 字节。删除配网信息会使设备恢复出厂。

配置代码在 app_mesh_config.c 中如下：

```
1.      .nvs_sector_cnt = CONFIG_MESH_SECTOR_COUNT_DEF,
2.      .nvs_write_size = sizeof(int),
3.      .nvs_sector_size = CONFIG_MESH_SECTOR_SIZE_DEF,
4.      .nvs_store_baddr = CONFIG_MESH_NVS_ADDR_DEF,
```

处于开发调试过程中时，在同一个开发板上烧录修改后的固件建议勾选 ISP 工具上的清除 Dataflash 选项，防止因为上一次的配网信息未清除导致调试出现错误。

2. 应用层程序介绍

2.1 应用层初始化

应用层初始化函数包含 BLE 从机初始化，以及 MESH 初始化，同时注册了一个按键回调用于演示。

```
1. void App_Init()
2. {
3.     GAPRole_PeripheralInit();
4.     Peripheral_Init();
5.
6.     App_TaskID = TMOS_ProcessEventRegister(App_ProcessEvent);
7.     vendor_model_srv_init(vnd_models);
8.     blemesh_on_sync();
9.     HAL_KeyInit();
10.    HalKeyConfig(keyPress);
11. }
```

MESH 初始化的最后，会调用 settings_load 函数从 dataflash 中获取配网信息

```
1. #if(CONFIG_BLE_MESH_SETTINGS)
2.     settings_load();
3. #endif /* SETTINGS */
4.
5.     if(bt_mesh_is_provisioned())
6.     {
7.         set_led_state(LED_PIN, TRUE);
8.         APP_DBG("Mesh network restored from flash");
9.     }
10.    #if(CONFIG_BLE_MESH_LOW_POWER)
11.        bt_mesh_lpn_set(TRUE);
12.        APP_DBG("Low power enable");
13.    #endif /* LPN */
14.    }
15.    else
16.    {
17.        set_led_state(LED_PIN, FALSE);
18.        APP_DBG("Waiting for provisioning data");
19.        Peripheral_AdvertData_Privisioned(FALSE);
20.    }
```

此时分两种情况：

如果已配网，会进入 prov_complete 回调函数，并在回调中将广播数据修改为已配网广播。然后回到初始化函数点亮演示 LED 灯。如果未使能低功耗节点功能，随后设备会进入正常工作状态，广播的同时开启 MESH 接收，等待手机连接或者收到 MESH 数据。

```
1. static void prov_complete(uint16_t net_idx, uint16_t addr, uint8_t flags, uint32_t iv_index)
```

如果未配网，则会关掉演示 LED 灯，并将广播数据修改为未配网广播。随后设备进入待机状态，只开启广播，等待手机连接配网。

2.2 收发数据接口

已配网的任意节点之间都可以互相通信，比如当我们需要发送数据给地址为 0x0234 的节点，可以直接控制任意节点通过调用发送函数往指定 0x0234 地址发送，也可以用手机连接任意一个节点，把要发送的数据和地址 0x0234 下发给连接到的节点，随后该节点会将此数据转发给 0x0234 地址的节点，地址为 0x0234 的节点收到数据后应答给转发数据的节点，随后此节点再将应答数据上报给手机，完成一次传输流程。

2.2.1 接收手机数据

默认手机会向从机的 0xFFE1 属性发送数据，设备收到数据会进入 simpleProfileChangeCB 回调。

```
1. static void simpleProfileChangeCB(uint8_t paramID, uint8_t *pValue, uint16_t len)
2. {
3.     switch(paramID)
4.     {
5.         case SIMPLEPROFILE_CHAR1:
6.             {
7.                 uint8_t newValue[SIMPLEPROFILE_CHAR1_LEN];
8.                 tmos_memcpy(newValue, pValue, len);
9.                 PRINT("profile ChangeCB CHAR1.. \n");
10.                App_peripheral_reveived(newValue, len);
11.                break;
12.            }
13. ...
```

在回调里调用应用层的处理函数 App_peripheral_reveived。在函数里会把数据按照命令格式解析，并做相对应的处理。

```
1. void App_peripheral_reveived(uint8_t *pValue, uint16_t len)
2. {
3.     tmos_memcpy(&app_mesh_manage, pValue, len);
4.     APP_DBG("CMD: %x", app_mesh_manage.data.buf[0]);
5.     switch(app_mesh_manage.data.buf[0])
6.     {
7.         // 配网信息命令
8.         case CMD_PROVISION_INFO:
9.             {
10. ...
```

2.2.2 接收 MESH 数据

协议栈收到 MESH 数据后，会调用 vendor_model_srv_rsp_handler 回调函数通知应用层

```
1. static void vendor_model_srv_rsp_handler(const vendor_model_srv_status_t *va
1)
```

```

2. {
3.     if(val->vendor_model_srv_Hdr.status)
4.     {
5.         // 有应答数据传输 超时未收到应答
6.         APP_DBG("Timeout opcode 0x%02x", val-
>vendor_model_srv_Hdr.opcode);
7.         return;
8.     }
9.     if(val-
>vendor_model_srv_Hdr.opcode == OP_VENDOR_MESSAGE_TRANSPARENT_MSG)
10.    {
11.        // 收到透传数据
12.        APP_DBG("len %d, data 0x%02x from 0x%04x", val-
>vendor_model_srv_Event.trans.len,
13.                val->vendor_model_srv_Event.trans.pdata[0],
14.                val->vendor_model_srv_Event.trans.addr);
15.        App_trans_model_reveived(val-
>vendor_model_srv_Event.trans.pdata, val->vendor_model_srv_Event.trans.len,
16.                val->vendor_model_srv_Event.trans.addr );
17.        // // 转发给主机(如果已连接)
18.        // peripheralChar4Notify(val-
>vendor_model_srv_Event.trans.pdata, val->vendor_model_srv_Event.trans.len);
19.    }
20. ...

```

在回调中会调用 App_trans_model_reveived 把数据按照命令格式解析，并做相对应的处理。

```

1. void App_trans_model_reveived(uint8_t *pValue, uint16_t len, uint16_t addr )
2. {
3.     tmos_memcpy(&app_mesh_manage, pValue, len);
4.     switch(app_mesh_manage.data.buf[0])
5.     {
6.         // 判断是否为删除命令
7.         case CMD_DELETE_NODE:
8.         ...

```

2.2.3 向手机上报数据

通过调用 peripheralChar4Notify 函数，将数据上报给 BLE 主机（手机）。

```

1. void peripheralChar4Notify(uint8_t *pValue, uint16_t len)
2. {
3.     attHandleValueNoti_t noti;
4.     if(peripheralConnList.connHandle != GAP_CONNHANDLE_INIT)
5.     {
6.         noti.len = len;

```

```
7.      noti.pValue = GATT_bm_alloc(peripheralConnList.connHandle, ATT_HANDLE_VALUE_NOTI, noti.len, NULL, 0);
8.      tmos_memcpy(noti.pValue, pValue, noti.len);
9.      if(simpleProfile_Notify(peripheralConnList.connHandle, -i) != SUCCESS)
10.     {
11.         PRINT("Notify ERR \n");
12.         GATT_bm_free((gattMsg_t *)-i, ATT_HANDLE_VALUE_NOTI);
13.     }
14. }
15. }
```

2.2.4 通过 MESH 网络发送数据

通过调用 `vendor_model_srv_send` 函数，可以将数据发送给指定地址，可以是组地址也可以是单播地址。

例程的应用层发送次数为 5 次，如果实际使用环境较复杂，可以提高发送次数，保证传输成功率。

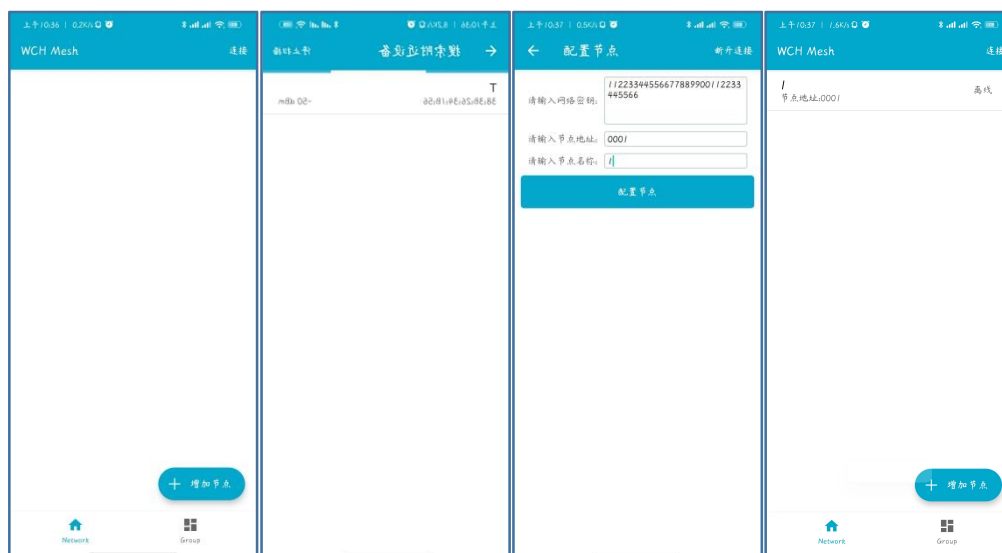
注意这里的 TTL 是默认是 3，如果网络较大，应把默认值根据需要转发的次数改大。

```
1. static int vendor_model_srv_send(uint16_t addr, uint8_t *pData, uint16_t len)
2. {
3.     struct send_param param = {
4.         .app_idx = vnd_models[0].keys[0], // 此消息使用的 app key，如无特定则使用第 0 个 key
5.         .addr = addr, // 此消息发往的目的地地址，例程为发往订阅地址，包括自己
6.         .trans_cnt = 0x05, // 此消息的用户层发送次数
7.         .period = K_MSEC(500), // 此消息重传的间隔，建议不小于 (200+50*TTL)ms，若数据较大则建议加长
8.         .rand = (0), // 此消息发送的随机延迟
9.         .tid = vendor_srv_tid_get(), // tid，每个独立消息递增循环，srv 使用 128~191
10.        .send_ttl = BLE_MESH_TTL_DEFAULT, // ttl，无特定则使用默认值
11.    };
12.    // return vendor_message_srv_indicate(&param, pData, len); // 调用自定义模型服务的有应答指示函数发送数据，默认超时 2s
13.    vendor_message_srv_trans_reset();
14.    return vendor_message_srv_send_trans(&param, pData, len); // 或者调用自定义模型服务的透传函数发送数据，只发送，无应答机制
15. }
```

3. 配网与网络管理

3.1 手机配置节点入网

打开 APP 进入主界面后点击增加节点，APP 会扫描并显示未配网的设备的名称和 MAC 地址，点击将进入配网界面，再配网界面中填写网络密钥，网络地址，以及配网后 APP 显示的名称。注意同一个网络的网络密钥必须相同，网络地址必须不能重复。



当配置成功后，已配网的节点将显示在主界面，可以再次连接节点以管理网络。

3.1.1 配网状态广播

APP 对于设备的广播做了限制，只有广播内容的第 7~12 个字节匹配才会显示，并且设备在配网前和配网后的广播内容有差异，配网前广播内容的第 7~12 个字节为 0x05FFD7070000，配网后广播内容的第 7~12 个字节为 0x05FFD7070100，APP 添加节点时，只会显示未配网设备；点击连接节点时只会显示已配网的设备。

当配网状态发生改变时，程序会调用 `Peripheral_AdvertData_Privisioned` 切换当前广播内容。

```
1. void Peripheral_AdvertData_Privisioned(uint8_t privisioned)
2. {
3.     uint8_t advertising_enable;
4.     if(privisioned)
5.     {
6.         advertData[11]=0x01;
7.         advertData[12]=0x00;
8.     }
9.     else
10.    {
11.        advertData[11]=0x00;
12.        advertData[12]=0x00;
13.    }
14.    GAPRole_GetParameter(GAPROLE_ADVERT_ENABLED, &advertising_enable);
```



```
15.     if(advertising_enable)
16.     {
17.         advertising_enable = FALSE;
18.         GAPRole_SetParameter(GAPROLE_ADVERT_ENABLED, sizeof(uint8_t), &advertising_enable);
19.         GAPRole_SetParameter(GAPROLE_ADVERT_DATA, sizeof(advertData), advertData);
20.         advertising_enable = TRUE;
21.         GAPRole_SetParameter(GAPROLE_ADVERT_ENABLED, sizeof(uint8_t), &advertising_enable);
22.     }
23.     else
24.     {
25.         GAPRole_SetParameter(GAPROLE_ADVERT_DATA, sizeof(advertData), advertData);
26.     }
27. }
```

3.1.2 配网流程

配网流程共包含两条命令，首先手机会发送 CMD_PROVISION_INFO，下发 iv_index 参数，然后发送 CMD_PROVISION，下发网络密钥和网络地址参数，节点收到全部三个参数后，会启动 APP_NODE_PROVISION_EVT 配网事件。

```
1.     // 配网信息命令
2.     case CMD_PROVISION_INFO:
3.     {
4.         if(len != PROVISION_INFO_DATA_LEN)
5.         {
6.             APP_DBG("Privisioning info data err!");
7.             return;
8.         }
9.         // 判断是设置还是查询
10.        if( app_mesh_manage.provision_info.set_flag )
11.        ...
```

由于 iv_index 参数在网络中是会改变的，所以每次连接已配网设备时，都要获取当前网络的 iv_index 参数，并在下次配网新设备时，将最新的 iv_index 参数下发给未配网设备。

```
1.     // 配网命令 包含 1 字节命令码+16 字节网络密钥+2 字节网络地址
2.     case CMD_PROVISION:
3.     {
4.         if(len != PROVISION_DATA_LEN)
5.         {
6.             APP_DBG("Privisioning data err!");
7.             return;
8.         }
```

```

9.          tmos_memcpy(self_prov_net_key, app_mesh_manage.provision.net_key
, PROVISION_NET_KEY_LEN);
10.          self_prov_addr = app_mesh_manage.provision.addr[0]|(app_mesh_manage.provision.addr[1]<<8);
11.          tmos_start_task(App_TaskID, APP_NODE_PROVISION_EVT, 160);
12.          break;
13.      }

```

在 APP_NODE_PROVISION_EVT 事件中，会调用 bt_mesh_provision 将完成配网，随后进入 prov_complete 配网完成回调

```

1.      if(events & APP_NODE_PROVISION_EVT)
2.      {
3.          if( self_prov_addr )
4.          {
5.              int err;
6.              err = bt_mesh_provision(self_prov_net_key, self_prov_net_idx, self_prov_flags,
7.                                     self_prov_iv_index, self_prov_addr, self_prov_dev_key);
8.              if(err)
9.              {
10.                 APP_DBG("Self Privisioning (err %d)", err);
11.                 self_prov_addr = 0;
12.                 app_mesh_manage.provision_ack.cmd = CMD_PROVISION_ACK;
13.                 app_mesh_manage.provision_ack.addr[0] = app_nodes[0].node_addr&0xFF;
14.                 app_mesh_manage.provision_ack.addr[1] = (app_nodes[0].node_addr>>8)&0xFF;
15.                 app_mesh_manage.provision_ack.status = STATUS_INVALID;
16.                 // 通知主机(如果已连接)
17.                 peripheralChar4Notify(app_mesh_manage.data.buf, PROVISION_ACK_DATA_LEN);
18.             }
19.         }
20.         return (events ^ APP_NODE_PROVISION_EVT);
21.     }

```

在 prov_complete 回调中，点亮 LED 指示灯，并把广播设置为已配网广播。由于此刻为第一次配网，所以需要执行配置本地网络信息流程（添加和绑定应用密钥），在回调最后启动 APP_NODE_EVT 事件。

```

1. static void prov_complete(uint16_t net_idx, uint16_t addr, uint8_t flags, uint32_t iv_index)
2. {
3.     int err;
4.     node_t *node;
5.

```

```
6.     APP_DBG("");
7.
8.     node = node_cfg_process(node, net_idx, addr, ARRAY_SIZE(elements));
9.     if(!node)
10.    {
11.        APP_DBG("Unable allocate node object");
12.        return;
13.    }
14.    set_led_state(LED_PIN, TRUE);
15.    Peripheral_AdvertData_Privisioned(TRUE);
16.
17.    // 如果未配置过网络信息, 则退出回调后, 执行配置本地网络信息流程
18.    if( vnd_models[0].keys[0] == BLE_MESH_KEY_UNUSED )
19.    {
20.        tmos_start_task(App_TaskID, APP_NODE_EVT, 160);
21.    }
22. }
```

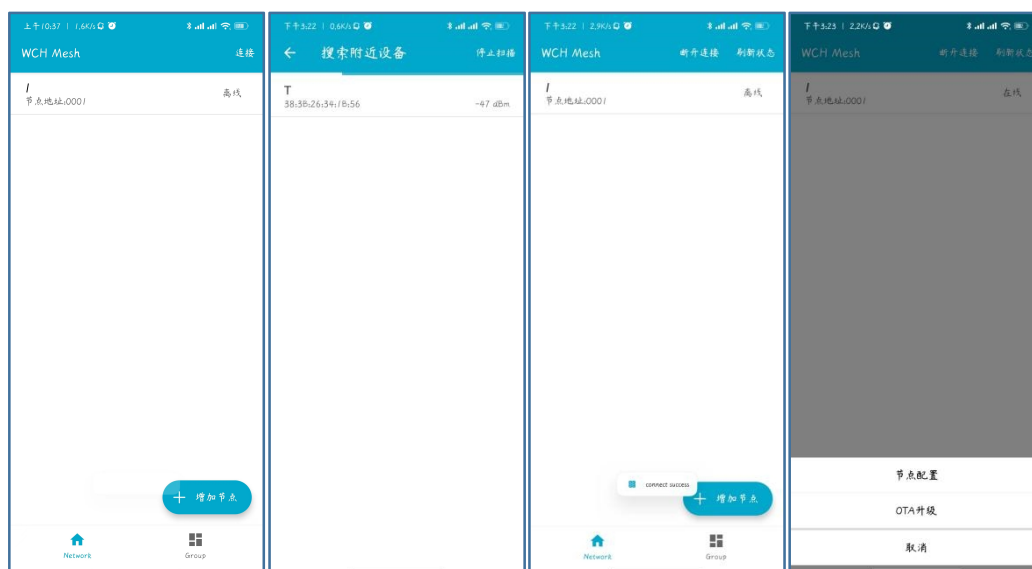
在 APP_NODE_EVT 事件中, 会调用 `cfg_local_net_info` 函数, 在此函数中设置自身的应用密钥。

```
1. static void cfg_local_net_info(void)
    最后上报应答数据给手机, 通知此时配网完成。
2.     // 节点配置任务事件处理
3.     if(events & APP_NODE_EVT)
4.     {
5.         cfg_local_net_info();
6.         app_mesh_manage.provision_ack.cmd = CMD_PROVISION_ACK;
7.         app_mesh_manage.provision_ack.addr[0] = app_nodes[0].node_addr&0xFF;
8.
9.         app_mesh_manage.provision_ack.addr[1] = (app_nodes[0].node_addr>>8)&0xFF;
10.        app_mesh_manage.provision_ack.status = STATUS_SUCCESS;
11.        // 通知主机(如果已连接)
12.        peripheralChar4Notify(app_mesh_manage.data.buf, PROVISION_ACK_DATA_LEN);
13.        return (events ^ APP_NODE_EVT);
14.    }
```

3.2 管理网络

在主界面点击右上角连接按钮, APP 会扫描并显示已在网络中的设备的名称和 MAC 地址, 任意选择一个设备连接即可对整个网络进行控制。

当成功连接到当前网络后, 再点击需要进行操作的设备名称, 即可选择进行配置或者升级。



3.2.1 查询节点在线状态

当连接到当前网络后，点击右上角“刷新状态”按钮，即会开始对网络中所有设备进行轮询刷新其在线状态。

轮询按配网顺序进行，采用一问一答的方式，只有上一次问答流程完成或者超时后，才会进行下一次问答流程。这里的超时是由设备程序控制的。

当与手机连接的设备端收到查询命令后，会解析要查询节点的地址，随后将命令通过 MESH 网络发往对应地址的设备（可以是自己），同时开启一个超时任务，如果一段时间后还未收到对应地址的应答，则上报查询超时。如果收到应答，则取消超时任务并上报应答数据。

默认超时时间为 10 秒，如果网络中节点相隔较远，需要中继次数很多，则需要把超时时间加长。

设备收到手机发送的查询状态命令处理代码：

```

1.      case CMD_ASK_STATUS:
2.      {
3.          if(len != ASK_STATUS_DATA_LEN)
4.          {
5.              APP_DBG("ask status data err!");
6.              return;
7.          }
8.          uint16 remote_addr;
9.          int status;
10.         remote_addr = app_mesh_manage.ask_status.addr[0]|(app_mesh_manage.ask_status.addr[1]<<8);
11.         APP_DBG("CMD_ASK_STATUS %x ",remote_addr);
12.         status = vendor_model_srv_send(remote_addr, app_mesh_manage.data.buf, ASK_STATUS_DATA_LEN);
13.         if(status)
14.         {
15.             APP_DBG("ask_status failed %d", status);
16.         }

```

```
17.         else
18.         {
19.             ask_status_node_address = remote_addr;
20.             // 定时，未收到应答就超时
21.             tmos_start_task(App_TaskID, APP_ASK_STATUS_NODE_TIMEOUT_EVT,
APP_CMD_TIMEOUT);
22.         }
23.         break;
24.     }
```

设备收到 MESH 网络发往自己的查询节点命令处理代码（应答状态）：

```
1.         // 判断是否为查询节点信息命令
2.         case CMD_ASK_STATUS:
3.         {
4.             if(len != ASK_STATUS_DATA_LEN)
5.             {
6.                 APP_DBG("ask status data err!");
7.                 return;
8.             }
9.             int status;
10.            app_mesh_manage.ask_status_ack.cmd = CMD_ASK_STATUS_ACK;
11.            app_mesh_manage.ask_status_ack.status = STATUS_SUCCESS; // 用户自
定义状态码
12.            status = vendor_model_srv_send(addr, app_mesh_manage.data.buf, A
SK_STATUS_ACK_DATA_LEN);
13.            if(status)
14.            {
15.                APP_DBG("send ack failed %d", status);
16.            }
17.            break;
18.        }
```

设备收到查询节点命令应答处理代码：

```
1.         // 判断是否为查询节点信息命令应答
2.         case CMD_ASK_STATUS_ACK:
3.         {
4.             if(len != ASK_STATUS_ACK_DATA_LEN)
5.             {
6.                 APP_DBG("ask status data err!");
7.                 return;
8.             }
9.             tmos_stop_task(App_TaskID, APP_ASK_STATUS_NODE_TIMEOUT_EVT);
10.            APP_DBG("ask status complete");
11.            vendor_message_srv_trans_reset();
12.            // 通知主机(如果已连接)
```

```
13.         peripheralChar4Notify(app_mesh_manage.data.buf, ASK_STATUS_ACK_D
ATA_LEN);
14.         break;
15.     }
```

3.2.2 数据传输

点击节点进入节点配置后，即可在发送数据的界面填入数据发送给指定节点。



与查询节点一样，设备收到手机下发的数据传输命令 CMD_TRANSFER 后，会转发给对应的节点，对应节点收到数据后，会调用应用数据处理函数 app_trans_process，在处理函数中，可以解析用户自定义的协议，例如演示的开关状态设置。

```
1. void app_trans_process(uint8_t *pValue, uint8_t len, uint16_t src_Addr, uint
16_t dst_Addr)
2. {
3.     uint16_t opcode = (pValue[0]<<8)|pValue[1];
4.     switch( opcode )
5.     {
6.         case BLE_MESH_MODEL_OP_GEN_ONOFF_SET:
7.             {
8.                 set_led_state(LED_PIN, pValue[2]);
9.                 break;
10.            }
11.
12.         default:
13.             {
14.                 break;
15.            }
16.     }
17. }
```

数据传输命令没有超时，但也可以应答，是否应答由用户自行开发，应答数据上报给手机时，APP 下方会弹出提示框。例程演示了应答功能：当设备从 MESH 网络中收到数据地址为单播地址（非群发群组地址）时，会原路把收到的数据第一字节加一返回。

```
1.  if( BLE_MESH_ADDR_IS_UNICAST(dst_addr) )
2.  {
3.      // 收到单播信息，这里演示原路把收到的数据第一字节加一返回
4.      app_mesh_manage.transfer_receive.cmd = CMD_TRANSFER_RECEIVE;
5.      app_mesh_manage.transfer_receive.data[0]++;
6.      status = vendor_model_srv_send(addr, app_mesh_manage.data.buf, len);
7.      ...
```

3.2.3 删除节点

在节点配置界面点击右上角“删除节点”按钮，即可将当前节点从网络中删除。

删除命令有应答超时，超时也是在设备端实现。当待删除节点收到删除命令后，会发送应答和清除节点信息命令，其中应答发往连接手机的设备；清除节点信息命令发往网络中所有的设备，让所有设备删除已存储的关于待删除节点的信息。

由于当节点被删除后就无法发送应答和清除节点命令，所以收到删除命令后，不会立即重置自身配网信息，而是启动 APP_DELETE_LOCAL_NODE_EVT 任务，等待数据发送结束后再执行删除。

```
1.      // 判断是否为删除命令
2.      case CMD_DELETE_NODE:
3.      {
4.          if(len != DELETE_NODE_DATA_LEN)
5.          {
6.              APP_DBG("Delete node data err!");
7.              return;
8.          }
9.          int status;
10.         APP_DBG("receive delete cmd, send ack and start delete node del
ay");
11.         app_mesh_manage.delete_node_ack.cmd = CMD_DELETE_NODE_ACK;
12.         app_mesh_manage.delete_node_ack.status = STATUS_SUCCESS;
13.         status = vendor_model_srv_send(addr, app_mesh_manage.data.buf,
DELETE_NODE_ACK_DATA_LEN);
14.         if(status)
15.         {
16.             APP_DBG("send ack failed %d", status);
17.         }
18.         // 即将删除自身，先发送 CMD_DELETE_NODE_INFO 命令
19.         APP_DBG("send to all node to let them delete stored info ");
20.         app_mesh_manage.delete_node_info.cmd = CMD_DELETE_NODE_INFO;
21.         status = vendor_model_srv_send(BLE_MESH_ADDR_ALL_NODES,
app_mesh_manage.data.buf, DELET
E_NODE_INFO_DATA_LEN);
22.         if(status)
23.         {
24.             APP_DBG("send ack failed %d", status);
25.         }
26.     }
```

```

27.         tmos_start_task(App_TaskID, APP_DELETE_LOCAL_NODE_EVT, APP_DELETE_LOCAL_NODE_DELAY);
28.         break;
29.     }

```

网络中的其余设备接收到清除节点信息命令后，同样不能立即删除，需要延迟等待被删除节点的数据执行本地删除后再清除其节点信息，防止刚删除又收到其数据重复存储下来。

```

1.         // 判断是否为有节点被删除，需要删除存储的节点信息
2.         case CMD_DELETE_NODE_INFO:
3.         {
4.             if(len != DELETE_NODE_INFO_DATA_LEN)
5.             {
6.                 APP_DBG("Delete node info data err!");
7.                 return;
8.             }
9.             delete_node_address = addr;
10.            tmos_start_task(App_TaskID, APP_DELETE_NODE_INFO_EVT, APP_DELETE_NODE_INFO_DELAY);
11.            break;
12.        }

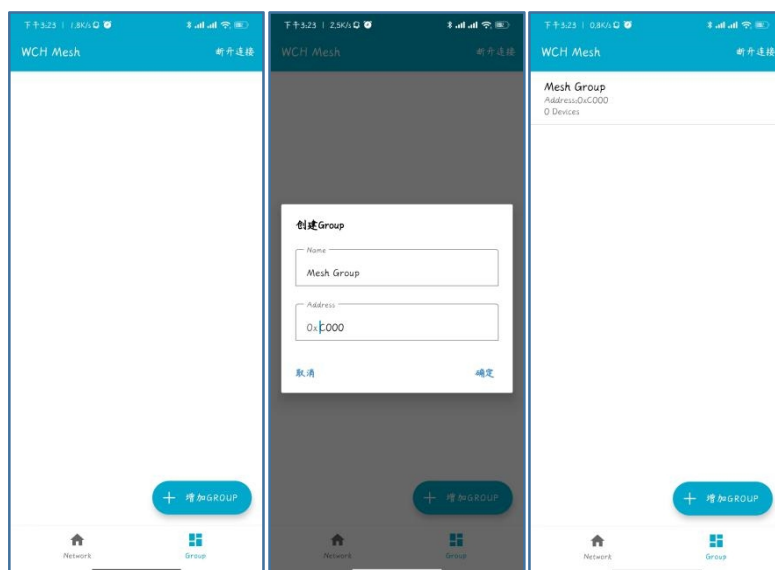
```

注意：已删除节点对应的网络地址不建议重复使用，因为如果删除节点时，有当前网络中的其他节点不在线，会导致未能成功清除其存储的节点信息，后续可能无法收到使用之前删除节点网络地址的新设备的数据。

3.2.4 订阅与组控制

在 APP 的 Group 界面可以创新新的组，组里可以添加多个节点，实现群发的效果，组消息也是通过 CMD_TRANSFER 传输，可以通过消息目标地址类型来区分一个消息时组消息还是单发消息。

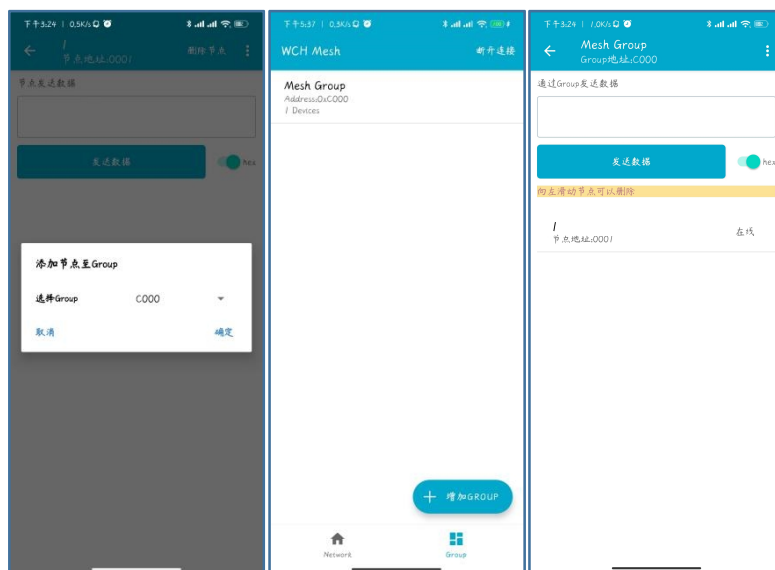
注意，不推荐群发消息带有应答机制，由于群发节点会同时收到数据，如果同时应答必定会出现漏收应答包的情况。



选中需要加入组的节点，进入节点配置界面，点击右上角选项，可以添加节点至已创

建的组中，之后点击组即可群发消息。

如果要删除某个组，需要将组内所有成员删除后才能删除组。



3.2.5 同一个手机管理多个网络方案

Demo APP 没有做多网络管理功能，如果需要同时管理多个网络，可以在主界面之前再添加一个创建网络的选项，每个网络需要存储其网络密钥和 `iv_index`。注意每个网络必须有独立不同的网络密钥。

4. OTA 升级

连接至网络后，点击需要升级的节点，选择 OTA 升级即可进入升级界面，获取硬件信息后，选择文件进行升级。升级后无需重新配网，网络信息不变。

OTA 升级流程共四个步骤：获取硬件信息、发送升级程序、校验升级程序、发送升级完成。所有过程包括每一包升级数据都是带应答机制，必须收到上一包应答才会发送下一包，例程默认超时为 10 秒，超时未收到应答会返回失败。

由于升级时间较长，建议 APP 端在 DEMO 的基础上添加如果收到超时再重试几次上一包数据的机制，防止因为偶然性丢包导致升级失败。

4.1 代码区域分配

CH58X 整个 codeflash 共 448K，从 0 地址开始分配：

起始地址	结束地址	代码区域名称	大小
0x0	0xFFFF	adv_vendor_self_provision_JumpIAP	4k
0x1000	0x26FFF	adv_vendor_self_provision_with_peripheral	152k
0x27000	0x4CFFF	升级固件存放区	152k
0x4D000	0x4DFFF	adv_vendor_self_provision_IAP	4k
0x4E000	0x6FFFF	CH58xBLE_ROM_MESH.hex (BLE 固定库)	136k

CH32F&V208 整个 codeflash 共 448K，从 0 地址开始分配：

起始地址	结束地址	代码区域名称	大小
0x0	0x1FFF	adv_vendor_self_provision_IAP	8k
0x2000	0x27FFF	adv_vendor_self_provision_with_peripheral	152k
0x28000	0x4DFFF	升级固件存放区	152k
0x4E000	0x6FFFF	wchble_rom_mesh.hex (BLE 固定库)	136k

5. 低功耗与朋友节点

例程默认使能了朋友功能，且默认一个朋友节点只支持跟一个低功耗节点建立朋友关系。如果要使能低功耗功能，只需要把朋友功能关闭，使能低功耗功能即可。

```
1. // 朋友节点功能
2. #define CONFIG_BLE_MESH_FRIEND 1
3. // 低功耗节点功能
4. #define CONFIG_BLE_MESH_LOW_POWER 0
```

使能低功耗功能以后，设备仍然可以和手机连接，如果需要进一步降低功耗，可以调用下述代码关闭广播，设备将只保留 MESH 网络通信，依旧可以支持所有的命令。

```
1. advertising_enable = FALSE;
2. GAPRole_SetParameter(GAPROLE_ADVERT_ENABLED, sizeof(uint8_t), &advertising_enable);
```

注意：低功耗功能在配网之前不可用。