



Hi3861V100 / Hi3861LV100 Mesh 软件

开发指南

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档主要介绍Hi3861V100、Hi3861LV100 WiFi Mesh的SDK开发相关内容，包括SDK架构、接口功能与使用说明。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



读者对象

本文档主要适用于以下工程师：



- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2020-04-30	第一次正式版本发布。
00B01	2020-04-10	第一次临时版本发布。



目录

前言.....	i
1 概述.....	1
1.1 背景.....	1
1.2 功能特性.....	1
1.3 SDK 架构.....	2
1.4 拓扑及节点角色.....	2
2 原始接口.....	4
2.1 二层接口.....	4
2.1.1 MBR/MG 节点功能.....	4
2.1.1.1 概述.....	4
2.1.1.2 节点启动.....	4
2.1.1.3 扫描.....	6
2.1.1.4 连接相关.....	7
2.1.2 Mesh STA 节点.....	9
2.1.2.1 概述.....	9
2.1.2.2 节点启动.....	9
2.1.2.3 扫描.....	10
2.2 三层入网接口.....	11
2.2.1 竞选.....	12
2.2.1.1 启动竞选.....	12
2.2.1.2 通知节点入网角色.....	12
2.2.1.3 停止竞选.....	13
2.2.1.4 示例代码.....	13
2.2.2 设置厂家 OUI.....	14
3 自动组网功能.....	15
3.1 概述.....	15
3.2 开发流程.....	16
3.3 注意事项.....	17
3.4 编程实例.....	17



1 概述

1.1 背景

1.2 功能特性

1.3 SDK架构

1.4 拓扑及节点角色

1.1 背景

无线Mesh网络作为一种对传统无线局域网（WLAN）的革命性技术创新，从拓扑结构上看，是一种多对多网络，它通过多跳的方式解决传统无线局域网无法实现大范围高速通信的痛点，同时还具有自我组网、自我修复等特性。它将WLAN的应用范围从“热点”扩展到“热区”，提升了WLAN的作用范围，并减少了对有线网络的依赖。无线Mesh网络是一种多跳网络，它与传统的单跳网络的最大不同之处在于无线Mesh网络中的AP（Access Point）不仅提供用户接入功能，还可以转发无线数据。多个AP构成一个网状结构，信号在网内从一个AP路由到另一个AP，最后通过与固定线路相连的AP传送到有线网络。

1.2 功能特性

Hi3861V100/Hi3861LV100上的WiFi-Mesh方案是基于IoT芯片的一种采用三层路由（Route-Over），支持低功耗节点与常规路由节点的网状拓扑方案，提供了以下功能点：

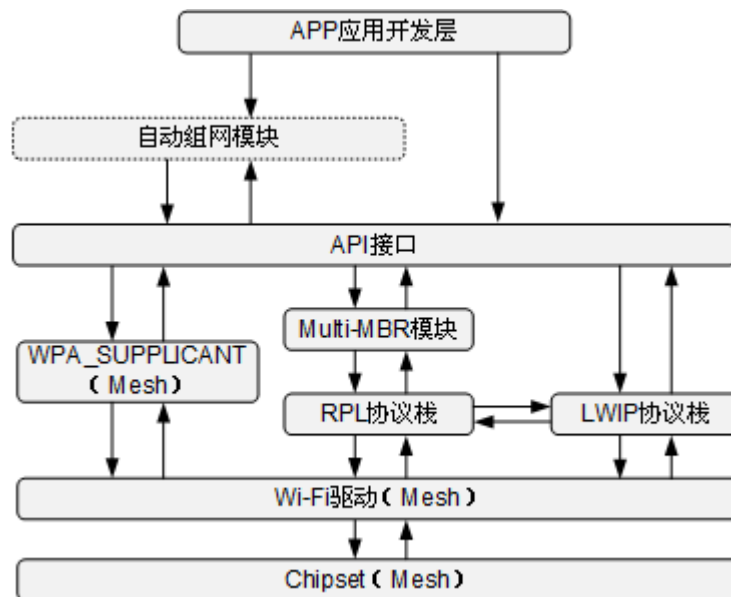
- 自动组网
- 路由快速切换
- 支持IPv6
- 分组控制
- 加密
- 支持节能（仅叶节点）
- 支持Multi-Portal
- 环路避免、检测、修复

- 时钟同步
- Beacon冲突检测

1.3 SDK 架构

Mesh SDK软件架构如图1-1所示。

图 1-1 Mesh SDK 软件架构示意图



Mesh SDK软件架构说明：

- APP应用开发层：用户基于API接口的二次开发。
- 自动组网模块：SDK提供的基于原始API接口开发的自动组网模块。
- API接口：Mesh SDK提供的标准接口。
- WPA_SUPPLICANT：WiFi管理模块，提供Mesh相关的扫描管理功能和关联状态机。
- Multi-MBR模块：提供Mesh MBR的仲裁、竞选、多MBR同步等功能。
- RPL协议栈：RFC6550, IPv6 Routing Protocol for Low-Power and Lossy Networks的简称，Mesh核心路由协议。
- LWIP协议栈：网络协议栈。
- Wi-Fi驱动：802.11协议实现模块，提供Mesh相关的链路层扫描关联等功能。
- Chipset：芯片，包括MAC、PHY、RF等模块。

1.4 拓扑及节点角色

本Mesh组网方案拥有三种不同的节点角色：

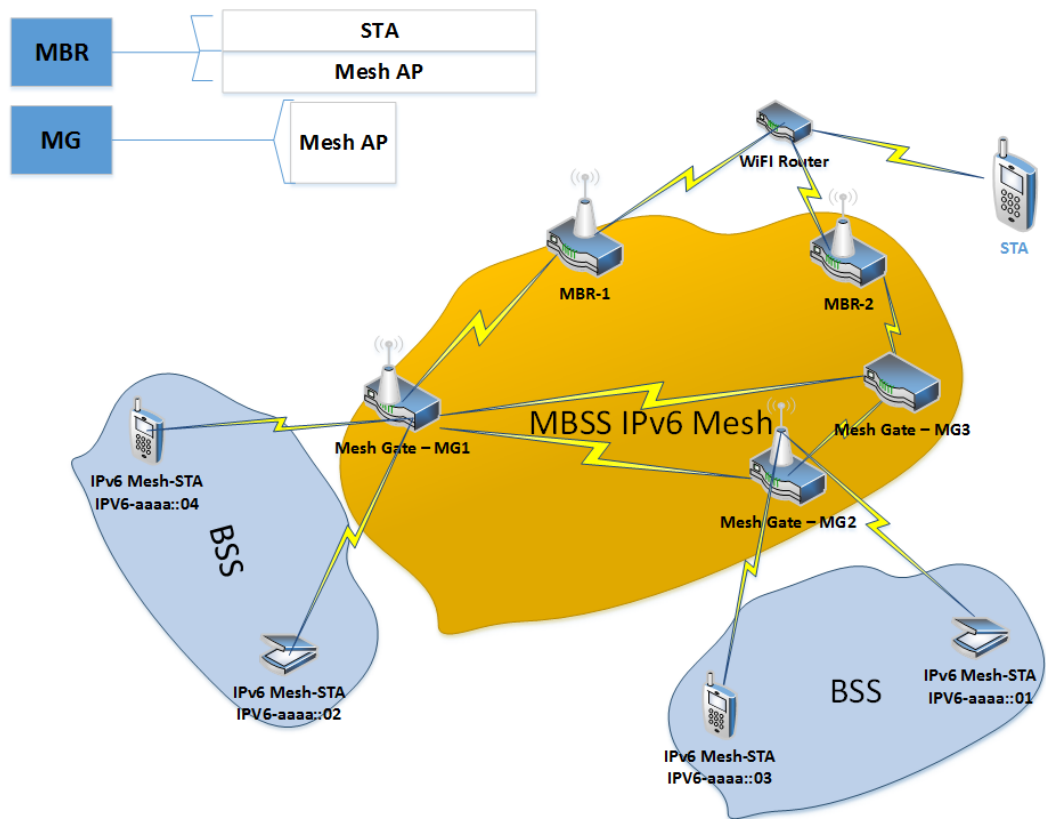
- MBR (Mesh Border Router)
Mesh网络与外网交互的节点，由一个直连路由器的STA和参与Mesh骨干网组网的Mesh AP组成。

- Mesh Gate (MG)
Mesh Gate同样参与Mesh骨干网组网，既可以与其他Mesh AP或MBR直连，也可以被Mesh STA搜索连接组成基本BSS网络，它作为基本BSS和Mesh骨干网中的一个桥，适用于对功耗需求较低、时延要求较高的节点。
- Mesh STA
采用802.11关联流程关联到MG上，通过MG获取IP地址入网，支持低功耗。

说明

MBR/MG属于Mesh骨干网；Mesh STA作为支持低功耗节点，只作为叶节点，不属于Mesh骨干网范围，不参与Mesh路由构建/转发。

图 1-2 Mesh 拓扑结构图





2 原始接口

2.1 二层接口

2.2 三层入网接口

2.1 二层接口

2.1.1 MBR/MG 节点功能

2.1.1.1 概述

MBR/MG是指组成Mesh骨干网络、拥有路由表的Mesh骨干网节点，这些节点适用于对低功耗需求较低、对时延要求较高的设备。

说明

本节仅介绍MBR/MG节点二层启动/扫描/关联的流程，Mesh网络首先需要启动一个MBR节点，之后才能继续拓展网络挂接MG/MSTA节点。二层节点启动成功后，即可开始三层网络组建，三层网络组建流程请参见《Hi3861V100 / Hi3861LV100 RPL 开发指南》。

2.1.1.2 节点启动

使用场景

Mesh节点需要以MBR/MG角色组建Mesh网络或加入某个已经存在的Mesh网络时。

功能

Mesh SDK提供的MBR/MG节点启动功能接口如表2-1所示。

表 2-1 MBR/MG 节点启动功能接口描述

接口名称	描述
hi_wifi_mesh_disconnect	Mesh断开与指定MAC地址的用户连接。



接口名称	描述
hi_wifi_mesh_start	启动Mesh接口。
hi_wifi_mesh_stop	停止Mesh接口。
hi_wifi_protocol_chn_switch_enable	设置Mesh整网信道切换使能或者去使能。
hi_wifi_protocol_chn_switch	设置Mesh整网信道切换的参数。
hi_wifi_get_mesh_node_info	查询Mesh节点信息。

开发流程

启动MBR节点典型流程：

步骤1 调用hi_wifi_sta_start，启动共存的STA。

步骤2 调用hi_wifi_mesh_start，启动共存的Mesh AP。

----结束

启动MG节点典型流程：

步骤1 调用hi_wifi_mesh_start，启动Mesh AP。

----结束

代码示例

示例1：启动MBR节点（STA+Mesh AP）

```
hi_char g_meshid[12] = {"mesh"};
hi_s32 g_router_chan = 11;
hi_s32 hi_mbr_start(hi_s32 argc, const hi_char* argv[])
{
    hi_s32 ret;
    hi_s32 len_sta;
    hi_s32 len_mesh;
    hi_char sta_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    hi_char mesh_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    errno_t rc;
    hi_wifi_mesh_config wpa_mesh_start = {0};

    ret = hi_wifi_sta_start(sta_ifname, &len_sta);
    if (ret != HISI_OK) {
        printf("hi_wifi_sta_start fail!\n");
    }
    hi_wifi_register_event_callback(hi_wifi_wpa_event_cb);
    ...
    wpa_mesh_start.auth = HI_WIFI_SECURITY_OPEN;
    wpa_mesh_start.channel = (hi_uchar)g_router_chan;
    memcpy_s(wpa_mesh_start.ssid, HI_WIFI_MAX_SSID_LEN + 1, g_meshid, strlen(g_meshid)
+ 1);

    ret = hi_wifi_mesh_start(&wpa_mesh_start, mesh_ifname, &len_mesh);
    if (ret != HISI_OK) {
```



```
    printf("mesh start fail!\n");  
  }  
  printf("mesh start succ!\n");  
}
```

示例2：启动MG节点（Mesh AP）

```
hi_char g_meshid[12] = {"mesh"};  
hi_s32 g_router_chan = 11;  
hi_s32 hi_mg_start(hi_s32 argc, const hi_char* argv[])  
{  
    hi_s32 ret;  
    hi_s32 len_mesh;  
    hi_char mesh_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};  
    errno_t rc;  
    hi_wifi_mesh_config wpa_mesh_start = {0};  
  
    ...  
    ...  
    wpa_mesh_start.auth = HI_WIFI_SECURITY_OPEN;  
    wpa_mesh_start.channel = (hi_uchar)g_router_chan;  
    memcpy_s(wpa_mesh_start.ssid, HI_WIFI_MAX_SSID_LEN + 1, g_meshid, strlen(g_meshid)  
+ 1);  
  
    ret = hi_wifi_mesh_start(&wpa_mesh_start, mesh_ifname, &len_mesh);  
    if (ret != HISI_OK) {  
        printf("mesh start fail!\n");  
    }  
    hi_wifi_register_event_callback(hi_wifi_wpa_event_cb);  
    printf("mesh start succ!\n");  
}
```

2.1.1.3 扫描

使用场景

MBR/MG节点扫描当前节点范围内存在其他可用的Mesh节点。

功能

Mesh SDK提供的MBR/MG节点扫描功能接口如表2-2所示。

表 2-2 MBR/MG 节点扫描功能接口描述

接口名称	描述
hi_wifi_mesh_scan	启动Mesh AP扫描。
hi_wifi_mesh_advance_scan	启动Mesh AP高级扫描。
hi_wifi_mesh_scan_results	获取Mesh AP扫描网络的结果。

开发流程

MBR节点典型扫描流程：



- 步骤1** 调用hi_wifi_sta_scan，共存STA发起扫描。
- 步骤2** 调用hi_wifi_sta_scan_results，查看STA扫描结果。

----结束

MG节点典型扫描流程：

- 步骤1** 调用hi_wifi_mesh_scan或hi_wifi_mesh_advance_scan（特殊扫描参数ssid、bssid、ssid长度、信道、是否为前缀扫描），Mesh AP发起扫描。
- 步骤2** 调用hi_wifi_mesh_scan_results，查看Mesh AP扫描结果。

----结束

代码示例

示例：Mesh AP节点进行扫描及查询扫描结果

```
hi_void hi_mr_scan(hi_void)
{
    ...
    pst_results = malloc(sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT);
    if (pst_results == HI_NULL) {
        printf("Alloc mem fail!\n");
    }
    ret = hi_wifi_mesh_scan(HI_NULL);
    if (ret != HISI_OK) {
        printf("fail to scan!\r\n");
    }
    memset_s(pst_results, (sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT),
        0, (sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT));
    if (hi_wifi_mesh_ap_scan_results(pst_results, &num) != HISI_OK) {
        printf("Fail to get mesh scan results!\r\n");
    }
    printf("Scan Result num:%d\n", num);
    for (ul_loop = 0; (ul_loop < num) && (ul_loop < WIFI_SCAN_AP_LIMIT); ul_loop++) {
        printf("Scan Result:%s, %d\n", pst_results[i].ssid, pst_results[i].rssi);
    }
    free(pst_results);
}
```

2.1.1.4 连接相关

使用场景

MBR/MG节点关联指定的目标Mesh节点。

功能

Mesh SDK提供的MBR/MG节点扫描功能接口如表2-3所示。

表 2-3 MBR/MG 节点连接相关功能接口描述

接口名称	描述
hi_wifi_mesh_connect	连接指定MAC地址的Mesh节点。



接口名称	描述
hi_wifi_mesh_set_accept_peer	设置Mesh节点支持/不支持Mesh peer连接。
hi_wifi_mesh_set_accept_sta	设置Mesh节点支持/不支持Mesh STA连接。
hi_wifi_mesh_get_connected_peer	Mesh获取已连接节点的信息。
hi_wifi_add_usr_app_ie	在Mesh管理帧中添加用户IE字段。
hi_wifi_delete_usr_app_ie	在Mesh管理帧中删除用户IE字段。
hi_wifi_mesh_disconnect	设置Mesh节点去关联指定节点。

开发流程

被关联节点典型流程：

- 步骤1** 调用hi_wifi_mesh_set_accept_peer，设置Mesh AP接收对端节点关联。
- 步骤2** 调用hi_wifi_mesh_set_accept_sta，设置Mesh AP接收Mesh STA节点关联。
- 步骤3** （可选）调用hi_wifi_add_usr_app_ie，设置Mesh AP添加用户自定义IE字段。
- 步骤4** 调用hi_wifi_mesh_get_connected_peer，获取Mesh AP关关节点的信息。

----结束

关联节点典型流程：

- 步骤1** （可选）调用hi_wifi_add_usr_app_ie，设置Mesh AP添加用户自定义IE字段。
- 步骤2** 调用hi_wifi_mesh_connect，节点发起关联。
- 步骤3** 调用hi_wifi_mesh_get_connected_peer，获取Mesh节点关关节点的信息。

----结束

去关联典型流程：

- 步骤1** 调用hi_wifi_mesh_disconnect，设置Mesh节点去关联对端节点。

----结束

代码示例

示例1：关联某Mesh AP节点及查询用户数

```
hi_void hi_mr_connect(hi_void)
{
    hi_u8 peer_num;
    hi_s32 ret;
    hi_u8 *mac;
    ...
    ret = hi_wifi_mesh_disconnect(mac, 6);
    if (ret != HISI_OK) {
        printf("mesh_disconnect failed.\n");
    }
    ...
}
```



```
hi_wifi_mesh_peer_info *peer_list = malloc(sizeof(hi_wifi_mesh_peer_info) * peer_num);
if (peer_list == NULL) {
    printf("malloc res fail!\n");
}
ret = hi_wifi_mesh_get_connected_peer(peer_list, &peer_num);
if (ret != HISI_OK) {
    printf("get connect peer fail!\n");
}
printf("Get connected peer succ,peer num[%d]\n", peer_num);
}
```

示例2：去关联某Mesh AP节点

```
hi_void hi_mr_disconnect(hi_void)
{
    hi_s32 ret;
    hi_u8 *mac;
    ...
    ret = hi_wifi_mesh_disconnect(mac, 6);
    if (ret != HISI_OK) {
        printf("mesh_disconnect failed.\n");
    }
    ...
}
```

2.1.2 Mesh STA 节点

2.1.2.1 概述

Mesh STA是指能够作为末端节点加入Mesh组网、具有Mesh能力的STA，本质上与普通STA没有差异，根据使用场景通常应用于具有低功耗要求的设备。

说明

本章节仅描述普通STA开启Mesh能力后新增/更改的接口，其余STA接口说明请参见《Hi3861V100/Hi3861LV100 WiFi软件 开发指南》。

2.1.2.2 节点启动

使用场景

需要以Mesh STA角色启动组建Mesh网络或加入指定已经存在的Mesh网络时。

功能

Mesh SDK提供的MBR/MG节点扫描功能接口如表2-4所示。

表 2-4 Mesh STA 节点启动相关功能接口描述

接口名称	描述
hi_wifi_set_mesh_sta	设置普通STA支持Mesh网络连接。



开发流程

典型Mesh STA启动流程：

步骤1 调用hi_wifi_set_mesh_sta，设置普通STA支持Mesh组网。

步骤2 调用hi_wifi_sta_start，启动STA。

----结束

代码示例

示例：启动Mesh STA

```
hi_void hi_msta_start(hi_void)
{
    hi_s32 ret;
    hi_s32 len = 0;
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};

    ret = hi_wifi_set_mesh_sta(1);
    if (ret != HISI_OK) {
        printf("Set mesh sta flag fail!\n");
    }

    ret = hi_wifi_sta_start(ifname, &len);
    if (ret != HISI_OK) {
        printf("Mesh sta start fail!\n");
    }
    printf("Mesh sta start succ!\n");
}
```

2.1.2.3 扫描

使用场景

Mesh STA节点扫描当前节点范围内存在的其他可关联Mesh节点。

功能

Mesh SDK提供的Mesh STA节点扫描功能接口如表2-5所示。

表 2-5 Mesh STA 节点扫描相关功能接口描述

接口名称	描述
hi_wifi_mesh_sta_scan	Mesh STA启动扫描。
hi_wifi_mesh_sta_advance_scan	Mesh STA添加限制条件的特殊扫描。
hi_wifi_mesh_sta_scan_results	Mesh STA查看扫描结果。

开发流程

典型Mesh STA扫描流程：



步骤1 调用hi_wifi_mesh_sta_scan或hi_wifi_mesh_sta_advance_scan（限制条件扫描），启动Mesh STA发起扫描。

步骤2 调用hi_wifi_mesh_sta_scan_results，查看Mesh STA扫描结果。

----结束

代码示例

示例1：普通扫描

```
hi_void hi_msta_scan(hi_void)
{
    hi_s32 ret;
    hi_wifi_mesh_scan_result_info *pst_results;
    hi_u32 num = WIFI_SCAN_AP_LIMIT;
    ...
    ret = hi_wifi_mesh_sta_scan();
    if (ret != HISI_OK) {
        printf("Scan fail!\n");
    }

    pst_results = malloc(sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT);
    if (pst_results == HI_NULL) {
        printf("Alloc mem fail!\n");
    }

    if (hi_wifi_mesh_sta_scan_results(pst_results, &num) != HISI_OK) {
        free(pst_results);
        printf("Get scan result fail!\n");
    }

    printf("Get scan result succ!\n");
    free(pst_results);
    return;
}
```

示例2：特殊扫描（指定信道）

```
hi_void channel_mesh_scan(hi_void)
{
    int ret;
    hi_wifi_scan_params scanParams = {0};

    scanParams.channel = 11; /* 指定11信道扫描 */
    ret = hi_wifi_mesh_advance_scan(&scanParams);
    if (ret != HISI_OK) {
        printf("channel_mesh_scan fail.\n");
    }
}
```

2.2 三层入网接口



2.2.1 竞选

说明

- Mesh网络中只有一个MBR，不能所有节点都作为MBR，所以需要竞选。当设备触发竞选时，会向关联到同一个路由器的设备发送竞选消息，竞选消息中带有厂家OUI编码，对于同一个厂商的设备间竞选，如果已经存在MBR，新节点只能作为MG。
- 启动竞选算法后，竞选模块会根据当前网络状态为节点分配一个合适的角色（MBR或MG）。

2.2.1.1 启动竞选

节点关联WiFi路由器后，先判断RSSI是否达到阈值，达到后可调用竞选接口参与竞选，首个接入Mesh中的MBR节点如果没有其他节点与之竞选则默认为MBR。

接口定义	hi_s32 mesh_election_start(mbr_election_info* info);
描述	当设备本身具备MBR的条件（位置固定、非电池供电、对耗电要求不高），但又不知道网络中MBR数是否达到上限（8个），可执行此函数来确定设备能否作为MBR。
输入参数	info：竞选信息结构体变量，成员如下： <ul style="list-style-type: none">call_back：竞选结果出来后的回调函数，回调函数入参是节点类型。router_rssi：WiFi路由器RSSI值。
输出参数	无
返回值	<ul style="list-style-type: none">HISI_OK：运行成功。HISI_FAIL：运行失败。
注意事项	<ul style="list-style-type: none">函数在已确定关联WiFi路由器且获取到IP地址后执行，否则没有与其他节点竞选。竞选完成后，三层会调用入口回调函数。传入的参数为节点类型。如果节点类型是MG，需要解除和WiFi路由器的关联后，重新以MG的身份关联MBR。竞选函数非阻塞式，函数执行完，竞选过程并没有执行完成，竞选过程完成后会调用参数func传入的回调函数，回调函数的入口是竞选结果。用户可更据竞选结果选择节点在Mesh网络中的入网角色。

2.2.1.2 通知节点入网角色

通过竞选得到节点身份后，可调用此函数通知竞选模块节点加入Mesh网络所处的角色。

加入 Mesh 网络

接口定义	hi_s32 mesh_election_notify_role(uint8_t mode)
描述	确定期望设备加入Mesh网络中的角色后，通知竞选模块节点的角色。



输入参数	mode: 接入类型MBR/MG。
输出参数	无
返回值	<ul style="list-style-type: none">• HISI_OK: 运行成功。• HISI_FAIL: 运行失败。
注意事项	需要在关联WiFi 路由器后执行。

2.2.1.3 停止竞选

如果节点确定以非MBR角色入网，可停止竞选，释放竞选模块使用的资源。

接口定义	hi_s32 mesh_election_stop(hi_void)
描述	停止节点竞选。
输入参数	无
输出参数	无
返回值	<ul style="list-style-type: none">• HISI_OK: 运行成功。• HISI_FAIL: 运行失败。
注意事项	MBR如果需保持MBR身份，请勿调用此接口。

2.2.1.4 示例代码

示例：OUI设置、节点竞选启动过程。

```
hi_void election_call(const hi_u8 mode)
{
    if (mode == HI_MBR) { // MBR
        mesh_election_notify_role(HI_MBR);
    }
    if (mode == HI_MG) { // MG
        printf("election result is mg");
    }
    return;
}

hi_void demo_join_mesh(hi_void)
{
    // 此处需要先关联WiFi路由器
    mbr_election_info info = {0};
    info.call_back = hi_mesh_election_callback;
    info.router_rssi = g_connected_router_rssi;
    mesh_election_start(&info);
    return;
}

hi_void mbr_main_thread(hi_void) /* 设备启动入口 */
{
    hi_u8 oui[OUI_LENGTH] = {1, 2, 3};
```



```
hi_mesh_set_oui(oui, OUI_LENTH);  
demo_join_mesh();  
return;  
}
```

2.2.2 设置厂家 OUI

通过此接口可以设置产品所属厂家的OUI（Organizationally Unique Identifier，组织唯一标识），不同OUI的设备不会组建在同一个Mesh网络中。

接口定义	hi_s8 hi_mesh_set_oui(const hi_u8 *oui, hi_u8 oui_len)
描述	设置厂家OUI。
输入参数	oui：3byte长度的OUI编码。 oui_len：OUI编码长度，固定3byte。
输出参数	无
返回值	<ul style="list-style-type: none">• HISI_OK：运行成功。• HISI_FAIL：运行失败。
注意事项	本接口在节点组网前调用，否则组网结果可能不正确。



3 自动组网功能

3.1 概述

3.2 开发流程

3.3 注意事项

3.4 编程实例

3.1 概述

自动组网模块是Mesh SDK中提供的一个基于标准Mesh API接口进行开发的用于节点角色仲裁的模块，主要包含以下功能：

- 节点启动
 - 用户指定：MBR、MG和Mesh-STA
 - 自动竞选：Mesh-Auto（仅限MBR和MG，Mesh-STA不支持自动组网竞选）
- 节点角色仲裁
- 节点二层关联选择（关联优选算法）
- 节点关联
 - 直连路由器
 - 直连MG
 - 组建Mesh骨干网
- Mesh-STA漫游处理
- 异常场景处理
 - 路由器异常场景
 - MBR异常/掉线场景
 - MG中间节点异常场景

用户可基于此模块，利用该模块提供的接口，直接搭建Mesh网络进行通信。

📖 说明

该模块可供用户直接使用进行Mesh网络搭建，也可作为Mesh SDK提供的原始API接口调用示例代码。



3.2 开发流程

功能

Mesh SDK自动组网功能提供的接口如表3-1所示。

表 3-1 Mesh SDK 自动组网功能接口描述

接口名称	描述
hi_mesh_start_autolink	启动Mesh自动组网。
hi_mesh_exit_autolink	退出Mesh自动组网。
hi_mesh_autolink_set_router_rssi_thres hold	设置MBR关联路由器的RSSI阈值。
hi_mesh_autolink_get_router_rssi_thres hold	获取MBR关联路由器的RSSI阈值。
hi_mesh_autolink_set_bw	设置自动组网模块的带宽。
hi_mesh_autolink_get_bw	获取自动组网模块的带宽。
hi_mesh_autolink_register_event_callb ack	注册自动组网模块的回调。

开发流程

利用自动组网模块API搭建Mesh网络的典型流程：

步骤1 根据接入路由器的参数，填写自组网的配置参数，调用hi_mesh_start_autolink，启动Mesh自动组网。

步骤2 调用hi_mesh_autolink_register_event_callback，向自动组网模块注册回调，接收角色回调，并获取节点接口。

----结束

退出自动组网模块的典型流程：

步骤1 调用hi_mesh_exit_autolink，退出Mesh自动组网。

----结束

返回值

Mesh Autolink模块返回的错误码如表3-2所示。



表 3-2 Autolink 模块返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

3.3 注意事项

- hi_mesh_start_autolink为非阻塞式接口，命令下发成功后需要等待一段时间才会有节点角色通过事件回调返回。
- 支持重复通过调用hi_mesh_start_autolink启动自动组网模块，每次调用该接口会检查是否当前已有进程，如果有，则会将之前的进程停止，重新启动自动组网流程。
- 调用自组网的接口时，须保证无任何业务VAP启动。如果有业务VAP启动，则须先调用对应关闭接口，关闭该业务VAP后再调用自组网接口。
- 启动Mesh自动组网模块时，节点角色设置支持HI_MESH_USRCONFIG_MBR、HI_MESH_USRCONFIG_MR、HI_MESH_USRCONFIG_MSTA、HI_MESH_USRCONFIG_AUTO共四种，其中HI_MESH_USRCONFIG_AUTO仅能竞选出MBR/MR，如果需要节点成为叶节点Mesh STA角色，需直接指定节点为HI_MESH_USRCONFIG_MSTA角色。
- 搭建Mesh网络时，须至少有一个节点以HI_MESH_USRCONFIG_MBR/HI_MESH_USRCONFIG_AUTO角色启动。
- 自动组网模块设置超时机制，如果在超时时间内未返回有效Mesh节点角色结果，则会在超时时间10min到期时退化到HI_MESH_AUTOLINK_ROUTER_MSTA角色连接路由器，如需重新入网，需重新调用hi_mesh_start_autolink启动入网。
- 退出自动组网模块只能使用hi_mesh_exit_autolink退出。

3.4 编程实例

自动竞选启动

```
#include "hi_mesh_autolink_api.h"

hi_void mesh_autolink_cb(const hi_mesh_autolink_role_cb *role_cb)
{
    dprintf("mesh role[%d]\n", role_cb->role);
    dprintf("ifname_first[%s], len_first[%d]\n", role_cb->info.ifname_first, role_cb->info.len_first);
    dprintf("ifname_second[%s], len_second[%d]\n", role_cb->info.ifname_second, role_cb->info.len_second);

    return;
}

hi_void example_start_autolink(hi_void)
{
    /*****Mesh Auto Network*****/
    hi_mesh_autolink_config mesh_auto_config = {
        "meshnetwork", HI_MESH_AUTH, "123456789", HI_MESH_USRCONFIG_AUTO};
}
```



```
hi_mesh_start_autolink(&mesh_auto_config);  
  
hi_mesh_autolink_register_event_callback(mesh_autolink_cb);  
}
```

结果验证:

```
mesh role[0]  
ifname_first[mesh0], len_first[5]  
ifname_second[wlan0], len_second[5]
```

指定 MR 启动

```
#include "hi_mesh_autolink_api.h"  
hi_void mesh_autolink_cb(const hi_mesh_autolink_role_cb *role_cb)  
{  
    dprintf("mesh role[%d]\n", role_cb->role);  
    dprintf("ifname_first[%s], len_first[%d]\n", role_cb->info.ifname_first, role_cb->info.len_first);  
    dprintf("ifname_second[%s], len_second[%d]\n", role_cb->info.ifname_second, role_cb->info.len_second);  
  
    return;  
}  
  
hi_void example_start_autolink(hi_void)  
{  
    /*****Mesh Auto Network*****/  
    hi_mesh_autolink_config mesh_auto_config = {  
        "meshnetwork", HI_MESH_AUTH, "123456789", HI_MESH_USRCONFIG_MR};  
  
    hi_mesh_start_autolink(&mesh_auto_config);  
  
    hi_mesh_autolink_register_event_callback(mesh_autolink_cb);  
}
```

结果验证:

```
mesh role[1]  
ifname_first[mesh0], len_first[5]  
ifname_second[], len_second[0]
```