Hi3861V100 / Hi3861LV100 RPL

开发指南

文档版本　01

发布日期　2020-04-30

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 的RPL模块中的API接口说明以及开发实现示例。

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对象：

- 软件开发工程师
- 技术支持工程师

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。
注意	表示如不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。

符号	说明
须知	用于传递设备或环境安全警示信息。如不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

1 API 接口说明

1.1 概述

RPL协议即低功耗有损网络路由协议，用于内部链接和路由器均受限的网络中，该网络下的路由器的处理器功能、内存和系统功耗（电池供电）都可能受到较大的限制，其中的网络连接也具有高丢包率、低数据传输率及不稳定的特性。

说明

本文档描述的接口为RPL协议对外的原始接口，目前Wi-Fi软件上层自组网模块中已经对这些接口进行调用并封装为功能调用接口，便于用户直接使用自组网的功能。 如果用户需要利用RPL协议中这些原始接口进行开发，可通过本文档并参照代码示例进行开发。

当前协议栈中提供的RPL功能接口如**表1-1**所示。

表 1-1 RPL 功能接口描述

接口名称	描述
rpl_init	RPL模块初始化。
rpl_deinit	RPL模块清除。
rpl_start	RPL模块启动。
rpl_stop	RPL模块停止。
rpl_route_entry_count	获取RPL中的路由条目数。
rpl_route_entry_get	获取RPL中的路由表。
rpl_rank_get	获取RPL中节点的rank值。
rpl_rank_threshold_set	设置RPL中节点的rank阈值。
l3_event_msg_callback_reg	注册上层的回调函数。
lwip_nat64_init	初始化nat64。
lwip_nat64_deinit	清除nat64。
netifapi_dhcp_clients_info_get	获取dhcp客户端的信息。
netifapi_dhcp_clients_info_free	释放对应接口dhcp客户端的信息。

1.2 rpl_init

接口定义	int rpl_init(char *name, uint8_t len);
描述	本接口用于初始化RPL（例如：分配内存等操作）。
参数	• name：netif的名称 • len：netif名称的长度（单位：byte）
返回值	• 成功：返回正整数（context index） • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.3 rpl_deinit

接口定义	int rpl_deinit(int ctx);
描述	本接口用于清除RPL（例如：释放内存等操作）。

参数	• ctx：context index，由rpl_init 返回的数值
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.4 rpl_start

接口定义	int rpl_start(int ctx, uint8_t mode, uint8_t inst_id, const rpl_config_t *cfg, const rpl_ip6_prefix_t *prefix);
描述	本接口用于启动RPL。
参数	• ctx：context index，由rpl_init返回的数值 • mode：当前节点入网的角色。当前取值仅有以下2个： – RPL_MODE_MBR：以MBR角色入网 – RPL_MODE_MG：以MG角色入网（后续的参数不使用） • inst_id：instance_id • cfg：DODAG config（当前未使用） • prefix：DODAG prefix（IPv6地址）
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.5 rpl_stop

接口定义	int rpl_stop(int ctx);
描述	本接口用于停止RPL，该操作会清空路由表、停止Timer等。
参数	• ctx：context index，由rpl_init返回的数值
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.6 rpl_route_entry_count

接口定义	int rpl_route_entry_count(int ctx, uint16_t *cnt);
描述	本接口用于获取路由条目数量。
参数	• ctx：context index，由rpl_init返回的数值 • cnt：路由条目数（输出参数）
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.7 rpl_route_entry_get

接口定义	int rpl_route_entry_get(int ctx, rpl_route_entry_t *rte, uint16_t cnt);
描述	本接口用于获取路由表。
参数	• ctx：context index，由rpl_init返回的数值 • rte：复制的路由表内存（输出参数） • cnt：路由表内存rte的条目个数
返回值	• 成功：返回正整数（复制的路由条目数） • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.8 rpl_rank_get

接口定义	int rpl_rank_get(uint16_t *rank);
描述	本接口用于获取节点的rank值。
参数	• rank：用于保存的rank值（输出参数）
返回值	• 成功：返回0 • 失败：返回-1
错误码	-

自从	nStack_N500 1.0.2

1.9 rpl_rank_threshold_set

接口定义	int rpl_rank_threshold_set(uint16_t rank_threshold);
描述	本接口用于设置节点的rank阈值。
参数	• rank_threshold：设置的该节点的rank阈值
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.10 l3_event_msg_callback_reg

接口定义	void l3_event_msg_callback_reg(enum l3_event_msg_type evt_type, app_callback_fn app_callback);
描述	本接口用于注册上层的回调函数。
参数	• evt_type：RPL中上报的事件类型 • app_callback：该事件类型对应的回调函数
返回值	-
错误码	-
自从	nStack_N500 1.0.2

1.11 lwip_nat64_init

接口定义	int lwip_nat64_init(char *name, uint8_t len);
描述	本接口用于初始化nat64。
参数	• name：网络接口名称 • len：接口名称的长度（单位：byte）
返回值	• 成功：0 • 失败：-1
错误码	-

自从	nStack_N500 1.0.2

1.12 lwip_nat64_deinit

接口定义	int lwip_nat64_deinit(void);
描述	本接口用于清除nat64，包括nat64表的清除、dhcp proxy的停止等。
参数	-
返回值	• 成功：返回0 • 失败：返回-1
错误码	-
自从	nStack_N500 1.0.2

1.13 netifapi_dhcp_clients_info_get

接口定义	netifapi_dhcp_clients_info_get(struct netif *netif, struct dhcp_clients_info **clis_info);
描述	本接口用于获取dhcp客户端的信息。
参数	• netif：dhcp client对应的网络接口 • clis_info：输出参数，获取到的dhcp clients的信息
返回值	• 成功：返回ERR_OK（值为0）。 • 失败：返回负值。
错误码	-
自从	nStack_N500 1.0.2

1.14 netifapi_dhcp_clients_info_free

接口定义	netifapi_dhcp_clients_info_free(struct netif *netif, struct dhcp_clients_info **clis_info);
描述	本接口用于释放对应接口dhcp客户端的信息。
参数	• netif：dhcp client对应的网络接口 • clis_info：对应于调用netifapi_dhcp_clients_info_get获取到的dhcp_clients_info信息

返回值	• 成功：返回ERR_OK（值为0） • 失败：返回负值
错误码	-
自从	nStack_N500 1.0.2

2 开发指南

2.1 开发约束

当前RPL模块存在以下规格限制：

- 一个节点最多存储128条下行路由表。
- 每个DAG节点最多支持4个parent。
- 每个RPL instance支持1个DODAG。
- DIO消息支持1个prefix。
- DAO消息中最多支持的单播的target个数是9个。
- RPL协议中，MBR模块需要使用IPv4地址，用于和路由器进行通信，但Mesh网络内部不使用IPv4通信，而是使用IPv6保持数据和机制的同步。

2.2 代码示例

示例1：

1. 初始化并开启一个RPL网络，调用rpl_init、rpl_start接口，开启成功。
2. 中间启动dhcp服务，然后开启nat64功能。
3. 关闭dhcp服务，nat64退出并进行清理。
4. 最后关闭该RPL网络，释放相关资源。

```
#include "lwip/lwip_ripple_api.h"
#include "lwip/netif.h"
#include "lwip/nat64_api.h"
#include "lwip/netifapi.h"
#include "hi_at.h"
#define WIFI_IFNAME_MAX_SIZE  16
#define RPL_MODE_MBR 1 /* Root mode */
#define RPL_MODE_MG 2  /* Router mode */
hi_u8 g_rpl_prefix[8] = {0xFD, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
hi_void hi_rpl_process_example()
```



```
{
    rpl_ip6_prefix_t prefix = {0};
    hi_s32 ctx;
    char ifname[WIFI_IFNAME_MAX_SIZE + 1] = "test_wifi_rpl";

    if (memcpy_s(prefix.prefix, sizeof(prefix.prefix), g_rpl_prefix, 8) != EOK) {
        return;
    }
    prefix.len = 8;

    ctx = rpl_init(ifname, strlen(ifname));
    if (ctx < 0) {
        hi_at_printf("rpl_init fail.\r\n");
        return;
    }

    if (rpl_start(ctx, RPL_MODE_MG, 99, NULL, &prefix) != 0) {
        hi_at_printf("rpl_start fail!\r\n");
        return;
    }
    hi_at_printf("start rpl example succ!\r\n");
...
...
    struct netif *lwip_netif = NULL;
    lwip_netif = netif_find("wlan0");
    if (lwip_netif == NULL) {
        return;
    }
    /* here we start dhcp first, and then init nat64 */
    netifapi_dhcp_start(lwip_netif);
    if (lwip_nat64_init("wlan0", 5) != 0) {
        hi_at_printf("nat64_init fail!\r\n");
        return;
    } else {
        hi_at_printf("nat64_init succ!\r\n");
    }
...
...
    /* here we stop dhcp first, and then deinit nat64 */
    netifapi_dhcp_stop(lwip_netif);
    if (lwip_nat64_deinit() != 0) {
    hi_at_printf("nat64_deinit fail!\r\n");
        return;
    }
...
...
    ret = rpl_deinit(ctx);
    if (ret == -1) {
        hi_at_printf("rpl deinit fail!\r\n");
        return;
    }
    hi_at_printf("mesh mr stop success!\r\n");
    return;
}
```

示例2：

- 根据传入的该RPL网络的ctx，调用rpl_route_entry_count获取其路由条目数。
- 调用rpl_route_entry_get可以将获取的这些条目存放到结构体数组rte中。

```
#include "lwip_ripple_api.h"
#include "hi_at.h"
```



```
int mesh_rpl_get_entry_example(hi_s32 ctx, hi_u8 sock_type)
{
    hi_u16 item_count = 0;
    /* here we get the rpl route entry count */
    if (rpl_route_entry_count(ctx, &item_count) != ERR_OK) {
        hi_at_printf("rpl_route_entry_count failed. nodeCount:%d\r\n", item_count);
        return -1;
    } else {
        hi_at_printf("rpl_route_entry_count succ. nodeCount : %d\r\n", item_count);
    }

    rpl_route_entry_t rte[10] = {0};
    rpl_route_entry_t *rte_ptr = rte;
    /* here we get the rpl route entry */
    if (rpl_route_entry_get(ctx, rte_ptr, item_count) < 0) {
        hi_at_printf("rpl_route_entry_get failed.\r\n");
        return -1;
    } else {
        hi_at_printf("rpl_route_entry_get succ.\r\n");
    }
    return 0;
}
```

示例3：获取节点rank值。

```
#include "lwip/lwip_ripple_api.h"
#include "hi_at.h"
int rpl_rank_get_example(void)
{
    hi_u16 my_rank = 0;
    int ret;
    ret = rpl_rank_get(&my_rank);
    if (ret == -1) {
        hi_at_printf("rpl_rank_get failed!\r\n");
        return -1;
    } else {
        hi_at_printf("rpl_rank_get succ. my_rank is %d\r\n", my_rank);
    }
    return 0;
}
```

示例4：注册l3事件，当事件发生时，执行对应的回调函数，调用l3_event_msg_callback_reg接口。

```
#include <stdio.h>
#include "lwip/l3event.h"
#include "hi_at.h"
static hi_void mesh_lwip_clean_parent_callback(hi_u8 type, hi_void *para)
{
    (hi_void) para;
    hi_at_printf("mesh_lwip_clean_parent_callback event L3_EVENT_PARENT_CLEAN\n");
    return;
}
static hi_void mesh_lwip_rout_change_callback(hi_u8 type, hi_void *para)
{
    (hi_void) para;
    hi_at_printf("mesh_lwip_rout_change_callback event L3_EVENT_ROUTE_CAHNGE\n");
    return;
}
static hi_void mesh_lwip_join_rpl_callback(hi_u8 type, hi_void *para)
{
    (hi_void) para;
    hi_at_printf("mesh_lwip_join_rpl_callback event L3_EVENT_MSG_RPL_JOIN_SUCC\n");
```



```
    return;
}
void l3_event_register_example(void)
{
    /* here we register these three L3_EVENT_MSG, and when L3_EVENT_MSG happend, the
callback function will be executed */
    l3_event_msg_callback_reg(L3_EVENT_MSG_PARENT_CLEAR,
mesh_lwip_clean_parent_callback);
    l3_event_msg_callback_reg(L3_EVENT_MSG_ROUTE_CHANGE,
mesh_lwip_route_change_callback);
    l3_event_msg_callback_reg(L3_EVENT_MSG_RPL_JOIN_SUCC, mesh_lwip_join_rpl_callback);
}
```

示例5：获取dhcp clients的信息，且之后进行释放。

```
#include "lwip/netif.h"
#include "lwip/netifapi.h"
#include "hi_at.h"
hi_void hi_get_dhcp_clients_example(void)
{
    struct netif *netif = netifapi_netif_find("wlan0");
    if (netif == NULL) {
        return;
    }
    struct dhcp_clients_info *clis_info = NULL;
    if (netifapi_dhcp_clients_info_get(netif, &clis_info) != ERR_OK) {
        hi_at_printf("get dhcp clients fail!\r\n");
        return;
    }
    hi_at_printf("get dhcp clients success!\r\n");
    if (clis_info != HI_NULL) {
        if (netifapi_dhcp_clients_info_free(netif, &clis_info)!= ERR_OK) {
            hi_at_printf("free dhcp clients fail!\r\n");
            return;
        }
    hi_at_printf("free dhcp clients success!\r\n");
    }
}
```

A 专有名词解释

- RPL：IPv6 Routing Protocal for Low-Power and Lossy Networks，指在低功耗有损网络（LLN）中使用的IPv6的路由组网协议。
- RPL instance：RPL实例，指由单个或者多个具有相同RPL instance ID的DODAG组成的集合。
- DAG：有向无环图，所有边具有方向性，且不存在回路。
- DODAG：目标导向有向无环图，无出边的DAG跟只有一个的有向无环图称为DODAG。
- DADAG parent：DODAG中某节点的父节点，指该节点通往DODAG根路径上的下一跳。
- DIO：DODAG信息对象，携带一个节点用于发现RPL实例、获取相关配置参数、选择DODAG父节点和管理DODAG的信息，用于构建上行路由。
- DAO：目的地公告对象，沿着DODAG往上传播目的地址信息，用于构建下行路由。
- DIS：DODAG信息请求，用于从一个RPL节点请求一个DIO，通过DIS消息节点可以探测它的邻居节点。
- GACK：私有消息，即从DODAG根节点发到目标节点的消息，该消息携带DAO消息已经到达DODAG根节点的确认信息。

