Read the Docs Template Documentation

发布 *v1.0-156-gcf50274*

Read the Docs

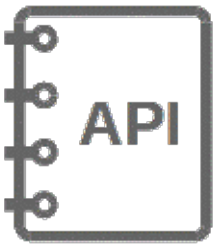
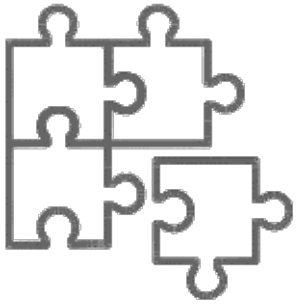
2023 年 03 月 09 日

Contents

[English]

这里是乐鑫 ESP-WIFI-MESH 开发框架 (esp-mdf) 的文档。

本文档提供不同语言的翻译版本 (English, 中文版, 如何切换语言？), 如有出入请以英文版本为准。

		
快速入门	API 参考	H/W 参考
		
API 指南	贡献代码	相关资源

CHAPTER 1

快速入门

[English]

本文档旨在指导用户创建 ESP-MDF 的软件环境。ESP-MDF 是基于 ESP-IDF 封装的 ESP-WIFI-MESH 开发构架，因此 ESP-MDF 的软件环境搭建与 ESP-IDF 基本相同，本文将主要阐述 ESP-MDF 与 ESP-IDF 环境区别和注意事项。在您使用 ESP-MDF 开发前，请先详细阅读 ESP-IDF 快速入指南。

1.1 准备工作

开发 ESP32 应用程序需要准备：

- **路由器**：用于连接外部网络
- **手机**：安装 ESP-WIFI-MESH 配网 app
- **ESP32 开发板**：运行 ESP-WIFI-MESH 至少需要两块以上的 ESP32 开发板，以构成一个网络

1.2 开发板指南

为了方便 ESP-WIFI-MESH 的开发和测试，我们专为 ESP-WIFI-MESH 开发测试而设计的开发板

1.2.1 ESP32-Buddy

ESP32-Buddy 是专为 ESP-WIFI-MESH 开发测试而设计的开发板。体积小，采用 USB 供电，方便做大数量设备的测试及距离测试。

- 购买链接：即将上架
- **功能：**
  - 16 MB 的 flash：存储运行日志
  - OLED 屏：显示当前设备所在的层级、连接状态等信息
  - LED：运行状态指示
  - 温湿度传感器：数据采集

1.2.2 ESP32-MeshKit

ESP32-MeshKit 包含一整套完整的 [ESP-Mesh 的照明解决方案](https://www.espressif.com/zh-hans/products/software/esp-mesh/overview)，可配套 ESP-Mesh App 使用，用于调研和了解 ESP-WIFI-MESH，也可以进行二次开发。

![image.png-50.7kB][3]

- 购买链接：[淘宝](https://item.taobao.com/item.htm?spm=a230r.1.14.1.55a83647K8jlrh&id=573310711489&ns=1&abbucket=3#detail)
- **产品：**
  - ESP32-MeshKit-Light：RGBW 智能灯，直观反应控制结果，可用于测试组网时间、响应速度、距离测试、稳定性测试等。
  - ESP32-MeshKit-Sense：带有光强传感器和温湿度传感器，可用于功耗测量和低功耗应用的开发，可配套使用 ESP-Prog 进行固件烧录和 Debug。
  - ESP32-MeshKit-Button：作为开关控制，用于低功耗应用的开发，可配套使用 ESP-Prog 进行固件烧录和 Debug

1.3 获取 ESP-MDF

工具链（包括用于编译和构建应用程序的程序）安装完后，你还需要 ESP32 相关的 API/库。API/库在 ESP-MDF 仓库 中。

获取本地副本：打开终端，切换到你要存放 ESP-MDF 的工作目录，使用 `git clone` 命令克隆远程仓库:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-mdf.git
```

ESP-MDF 将会被下载到 `~/esp/esp-mdf` 目录下。

注解: 此命令将克隆 master 分支，该分支保存着 ESP-MDF 的最新版本，它功能齐全，每周都会更新一些新功能并修正一些错误。

注解: GitHub 中”下载 zip 文档”的功能不适用于 ESP-MDF

1.4 设置 ESP-MDF 路径

工具链程序使用环境变量 `MDF_PATH` 来访问 ESP-MDF。这个变量应该设置在你的 PC 中，否则工程将不能编译。其设置方法与 `IDF_PATH` 相同。

1.5 配置

在终端窗口中，输入 `cd ~/esp/get-started` 进入 `get-started` 所在目录，然后启动工程配置工具 `menuconfig`:

```
cd ~/esp/get-started
make menuconfig
```

- `examples` 的配置在 `Example Configuration` 子菜单下
- 功能模块的配置在 `Component config` 中以 `MDF` 开头的子菜单下

1.6 编译和烧写

您可以通过 *make menuconfig* 来配置串口，也可以通过直接在命令行上使用 `ESPPORT` 和 `ESPBAUD` 环境变量来指定串口和波特率:

```
make erase_flash flash -j5 monitor ESPBAUD=921600 ESPPORT=/dev/ttyUSB0
```

1.7 工具

您也可以选择使用 `tools` 目录下的脚本来简化开发流程。

`gen_misc.sh` 进行编译和烧写，它在 `make monitor` 基础上加入了时间戳和日志保存:

```
cp $MDF_PATH/tools/gen_misc.sh .
./gen_misc.sh /dev/ttyUSB0
```

multi_downloads.sh 和 multi_downloads.sh 进行批量设备的烧录:

```
cp $MDF_PATH/tools/multi_*.sh .
./multi_downloads.sh 49
./multi_open_serials.sh 49
```

format.sh 进行代码格式化:

```
$MDF_PATH/tools/format.sh .
```

CHAPTER 2

API Reference

2.1 Mcommmon API

`Mcommon` (Mesh common) is a module shared by all ESP-MDF components, and it features the followings:

1. Memory Management: manages memory allocation and release, and can help find a memory leak;
2. Error Codes: checks the error codes of all the modules, and therefore can help find out what could possibly go wrong;
3. Event Loop: uses an identical function for all the modules to deal with an event.
4. Data persistence: provide an API to save any type of data on Flash.

2.1.1 Memory Management

Header File

- mcommon/include/mdf_mem.h

Functions

void **mdf_mem_add_record**(void *_ptr_, int _size_, **const** char *_tag_, int _line_)

Add to memory record.

Parameters

- `ptr`: Memory pointer
- `size`: Memory size
- `tag`: Description tag
- `line`: Line number

void **mdf_mem_remove_record**(void *_ptr_, **const** char *_tag_, int _line_)

Remove from memory record.

Parameters

- `ptr`: Memory pointer
- `tag`: Description tag
- `line`: Line number

void **mdf_mem_print_record**(void)

Print the all allocation but not released memory.

Attention Must configure CONFIG_MDF_MEM_DEBUG == y annd esp_log_level_set(mdf_mem, ESP_LOG_INFO);

void **mdf_mem_print_heap**(void)

Print memory and free space on the stack.

void **mdf_mem_print_task**(void)

Print the state of tasks in the system.

Macros

MDF_MEM_DEBUG

< _cplusplus CONFIG_MDF_MEM_DEBUG

CONFIG_MDF_MEM_DBG_INFO_MAX

CONFIG_MDF_MEM_DBG_INFO_MAX

MDF_MEM_DBG_INFO_MAX

MALLOC_CAP_INDICATE

MDF_MALLOC(size)

Malloc memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `size`: Memory size

`MDF_CALLOC`(n, size)

Calloc memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `n`: Number of block
- `size`: Block memory size

`MDF_REALLOC`(ptr, size)

Reallocate memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `ptr`: Memory pointer
- `size`: Block memory size

`MDF_REALLOC_RETRY`(ptr, size)

Reallocate memory, If it fails, it will retry until it succeeds.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `ptr`: Memory pointer
- `size`: Block memory size

`MDF_FREE`(ptr)

Free memory.

Parameters

- `ptr`: Memory pointer_cplusplus **MDF_MEM_H**

2.1.2 Error Codes

Header File

- mcommon/include/mdf_err.h

Functions

`const` char *`mdf_err_to_name`(*mdf_err_t code*)

Returns string for mdf_err_t error codes.

This function finds the error code in a pre-generated lookup-table and returns its string representation.

The function is generated by the Python script tools/gen_mdf_err_to_name.py which should be run each time an mdf_err_t error is modified, created or removed from the IDF project.

Return string error message

Parameters

- `code`: mdf_err_t error code

Macros

`MDF_OK`

mdf_err_t value indicating success (no error)

`MDF_FAIL`

Generic mdf_err_t code indicating failure

`MDF_ERR_NO_MEM`

Out of memory

`MDF_ERR_INVALID_ARG`

Invalid argument

`MDF_ERR_INVALID_STATE`

Invalid state

`MDF_ERR_INVALID_SIZE`

Invalid size

`MDF_ERR_NOT_FOUND`

Requested resource not found

`MDF_ERR_NOT_SUPPORTED`
: Operation or feature not supported

`MDF_ERR_TIMEOUT`
: Operation timed out

`MDF_ERR_INVALID_RESPONSE`
: Received response was invalid

`MDF_ERR_INVALID_CRC`
: CRC or checksum was invalid

`MDF_ERR_INVALID_VERSION`
: Version was invalid

`MDF_ERR_INVALID_MAC`
: MAC address was invalid

`MDF_ERR_NOT_INIT`
: MAC address was invalid

`MDF_ERR_BUF`
: The buffer is too small

`MDF_ERR_MWIFI_BASE`
: Starting number of MWIFI error codes

`MDF_ERR_MESPNOW_BASE`
: Starting number of MESPNOW error codes

`MDF_ERR_MCONFIG_BASE`
: Starting number of MCONFIG error codes

`MDF_ERR_MUPGRADE_BASE`
: Starting number of MUPGRADE error codes

`MDF_ERR_MDEBUG_BASE`
: Starting number of MDEBUG error codes

`MDF_ERR_MLINK_BASE`
: Starting number of MLINK error codes

`MDF_ERR_CUSTOM_BASE`
: Starting number of COUSTOM error codes

`CONFIG_MDF_LOG_LEVEL`
: CONFIG_MDF_LOG_LEVEL

`MDF_LOG_LEVEL`

`MDF_LOG_FORMAT`(letter, format)

`MDF_LOGE`(format, ...)

`MDF_LOGW`(format, ...)

`MDF_LOGI`(format, ...)

`MDF_LOGD`(format, ...)

`MDF_LOGV`(format, ...)

`MDF_ERROR_CHECK`(con, err, format, ...)

Macro which can be used to check the error code, and terminate the program in case the code is not MDF_OK. Prints the error code, error location, and the failed statement to serial output.

Disabled if assertions are disabled.

`MDF_ERROR_ASSERT`(err)

Macro serves similar purpose as `assert`, except that it checks `esp_err_t` value rather than a `bool` condition. If the argument of `MDF_ERROR_ASSERT` is not equal `MDF_OK`, then an error message is printed on the console, and `abort()` is called.

Note If `IDF monitor` is used, addresses in the backtrace will be converted to file names and line numbers.

Return [description]

Parameters

- `err`: [description]

`MDF_ERROR_GOTO`(con, lable, format, ...)

`MDF_ERROR_CONTINUE`(con, format, ...)

`MDF_ERROR_BREAK`(con, format, ...)

`MDF_PARAM_CHECK`(con)

_cplusplus **MDF_ERR_H**

Type Definitions

`typedef` int `mdf_err_t`

< _cplusplus

2.1.3 Event Loop

Header File

- mcommon/include/mdf_event_loop.h

Functions

mdf_err_t **mdf_event_loop_init**(*mdf_event_loop_cb_t cb*)

Initialize event loop, create the event handler and task.

Attention Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `cb`: Application specified event callback, it can be modified by call mdf_event_loop_set

mdf_err_t **mdf_event_loop_deinit**()

Deinitialize event loop, delete the event handler and task.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdf_event_loop**(*mdf_event_loop_t event*, void **ctx*)

Call event callback function directly.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Event type defined in this file
- `ctx`: Reserved for user

mdf_event_loop_cb_t **mdf_event_loop_set**(*mdf_event_loop_cb_t cb*)

Set event loop callback function.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `cb`: Set application event callback

mdf_err_t **mdf_event_loop_send**(*mdf_event_loop_t event*, void **ctx*)

Send the event to the event handler.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Generated events
- `ctx`: Reserved for user

mdf_err_t **mdf_event_loop_delay_send**(*mdf_event_loop_t event*, void **ctx*, TickType_t *delay_ticks*)

Delay send the event to the event handler.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Generated events
- `ctx`: Reserved for user
- `delay_ticks`: Delay time

Macros

CONFIG_EVENT_QUEUE_NUM

CONFIG_EVENT_QUEUE_NUM

EVENT_QUEUE_NUM

CONFIG_MDF_EVENT_TASK_NAME

CONFIG_MDF_EVENT_TASK_NAME

MDF_EVENT_TASK_NAME

CONFIG_MDF_EVENT_TASK_STACK_SIZE

CONFIG_MDF_EVENT_TASK_STACK

MDF_EVENT_TASK_STACK

`CONFIG_MDF_EVENT_TASK_PRIOTY`
CONFIG_MDF_EVENT_TASK_PRIOTY

`MDF_EVENT_TASK_PRIOTY`

`MDF_EVENT_MWIFI_BASE`

`MDF_EVENT_MESPNOW_BASE`

`MDF_EVENT_MCONFIG_BASE`

`MDF_EVENT_MUPGRADE_BASE`

`MDF_EVENT_MDEBUG_BASE`

`MDF_EVENT_MLINK_BASE`

`MDF_EVENT_CUSTOM_BASE`

Type Definitions

`typedef` uint32_t **`mdf_event_loop_t`**

`typedef` *mdf_err_t* (*`mdf_event_loop_cb_t`)(*mdf_event_loop_t* event, void *ctx)
Application specified event callback function.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Event type defined in this file
- `ctx`: Reserved for user

2.1.4 Info Store

Header File

- mcommon/include/mdf_info_store.h

Functions

esp_err_t **`mdf_info_init`**(void)
Initialize the default NVS partition.

Return

- ESP_FAIL
- ESP_OK

esp_err_t **mdf_info_save**(**const** char *_key_, **const** void *_value_, size_t _length_)

save the information with given key

Return

- ESP_FAIL
- ESP_OK

Parameters

- `key`: Key name. Maximal length is 15 characters. Shouldn' t be empty.
- `value`: The value to set.
- `length`: length of binary value to set, in bytes; Maximum length is 1984 bytes (508000 bytes or (97.6% of the partition size - 4000) bytes whichever is lower, in case multi-page blob support is enabled).

esp_err_t **__mdf_info_load**(**const** char *_key_, void *_value_, size_t _len_, uint32_t _type_)

esp_err_t **mdf_info_erase**(**const** char *_key_)

Macros

`MDF_SPACE_NAME`

`LENGTH_TYPE_NUMBER`

Load the information, esp_err_t mdf_info_load(const char *key, void *value, size_t *length); esp_err_t mdf_info_load(const char *key, void *value, size_t length);.

Attention The interface of this api supports size_t and size_t * types. When the length parameter of the pass is size_t, it is only the length of the value. When the length parameter of the pass is size_t *, the length of the saved information can be obtained.

Return

- ESP_FAIL
- ESP_OK

Parameters

- `key`: The corresponding key of the information that want to load
- `value`: The corresponding value of key
- `length`: The length of the value, Pointer type will return length

LENGTH_TYPE_POINTER

mdf_info_load(key, value, len)

2.2 Mconfig API

Mconfig (Mesh Network Configuration) is a network configuration solution for ESP-WIFI-MESH, which sends network configuration information to ESP-WIFI-MESH devices in a convenient and efficient manner.

2.2.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mconfig, which includes:

- Connect to the external network: This can be achieved by the root node via MQTT and HTTP.

2.2.2 Mconfig Blufi

Header File

- mconfig/include/mconfig_blufi.h

Functions

mdf_err_t **mconfig_blufi_init**(**const** *mconfig_blufi_config_t* **config*)
initialize Bluetooth network configuratxion

Attention The BLE stack must be enabled first through menuconfig configuration.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

mdf_err_t **mconfig_blufi_deinit**(void)
de-initialize Bluetooth network configuration

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_blufi_send**(uint8_t **data*, size_t *size*)

This function is called to custom data, send a custom request to the APP to verify the device.

Return

- MDF_OK
- MDF_FAIL
- MDF_ERR_INVALID_ARG___MCONFIG_BLUFI_H___

Parameters

- `data`: Custom data value
- `size`: The length of custom data

Structures

struct mconfig_blufi_config_t

Bluetooth configuration network related configuration.

Public Members

char **name**[MCONFIG_BLUFI_NAME_SIZE]

Local device & peripheral name, If the length of name is greater than 10 bytes, it will overwrite custom_data, and custom_data will not be available.

uint16_t **company_id**

Company Identifiers (https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers)

uint16_t **tid**

Type of device

uint8_t **custom_size**

Custom data size

uint8_t **custom_data**[MCONFIG_BLUFI_CUSTOM_SIZE]

Placed in a Bluetooth broadcast package

bool **only_beacon**

Send only beacon does not support connection

struct mconfig_blufi_data_t

Mconfig_blufi event callback parameters.

Public Members

uint8_t *data
: Custom data value

size_t size
: The length of custom data

Macros

MCONFIG_BLUFI_NAME_SIZE
: Contains the ending character

MCONFIG_BLUFI_CUSTOM_SIZE
: BLE broadcast data packets have a valid length of up to 31 bytes

CONFIG_BLUFI_BROADCAST_OUI
: Used to filter other Bluetooth broadcast packets, 3 bytes CONFIG_BLUFI_FILTER_OUI

MDF_EVENT_MCONFIG_BLUFI_STARTED

MDF_EVENT_MCONFIG_BLUFI_STOPED

MDF_EVENT_MCONFIG_BLUFI_CONNECTED

MDF_EVENT_MCONFIG_BLUFI_DISCONNECTED

MDF_EVENT_MCONFIG_BLUFI_STA_CONNECTED

MDF_EVENT_MCONFIG_BLUFI_STA_DISCONNECTED

MDF_EVENT_MCONFIG_BLUFI_FINISH

MDF_EVENT_MCONFIG_BLUFI_RECV

2.2.3 Mconfig Chain

Header File

- mconfig/include/mconfig_chain.h

Functions

mdf_err_t **mconfig_chain_slave_init**(void)
: Chain configuration network slave initialization for obtaining network configuration information.

  Attention The received network configuration information is sent to mconfig_queue

  Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_chain_slave_channel_switch_disable**(void)

Disable slave to switch wifi channel.

Attention Chain configuration network slaves, will constantly switch channels to find the master, if you need to make a wifi connection, You must first disable the switch of the slave wifi channel.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_chain_slave_channel_switch_enable**(void)

Enable slave to switch wifi channel.

Attention The slave will continuously switch the wifi channel for scanning, which is enabled by default.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_chain_slave_deinit**(void)

Free all resource allocated in mconfig_chain_slave_init and stop chain_slave task.

Return

- MDF_OK
- MDF_ERR_NOT_SUPPORTED

mdf_err_t **mconfig_chain_master**(**const** *mconfig_data_t* **config*, TickType_t *duration_ticks*)

Chain configuration network host initialization, sending network configuration information to the slave.

Return

- MDF_OK
- MDF_ERR_NOT_SUPPORTED

Parameters

- `config`: This configuration information will be sent to the slave device.
- `duration_ticks`: Stop chain_master after the set time

mdf_err_t `mconfig_chain_filter_rssi`(int8_t *rssi*)

Chain Devices with weak rssi are not allowed to join the network.

Note If you use it, you must call after `mconfig_chain_master`

Return

- MDF_OK
- MDF_FAIL___MCONFIG_CHAIN_H___

Parameters

- `rssi`: When the device signal strength is less than this value, it will not join the network.

Macros

`MDF_EVENT_MCONFIG_CHAIN_SLAVE_STARTED`

`MDF_EVENT_MCONFIG_CHAIN_SLAVE_STOPED`

`MDF_EVENT_MCONFIG_CHAIN_FINISH`

`MDF_EVENT_MCONFIG_CHAIN_MASTER_STARTED`

`MDF_EVENT_MCONFIG_CHAIN_MASTER_STOPED`

`MDF_EVENT_MCONFIG_CHAIN_FOUND_ROUTER`

2.2.4 Mconfig Queue

Header File

- mconfig/include/mconfig_queue.h

Functions

mdf_err_t `mconfig_queue_write`(`const` *mconfig_data_t* **mconfig_data*, TickType_t *wait_ticks*)

Write data to the queue of mconfig.

Return

- MDF_OK
- MDF_ERR_TIMEOUT

Parameters

- `mconfig_data`: information that points to the configuration of the network
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mconfig_queue_read**(*mconfig_data_t* ***mconfig_data*, TickType_t *wait_ticks*)

READ data to the queue of mconfig.

Return

- MDF_OK
- MDF_ERR_TIMEOUT___MCONFIG_QUEUE_H___

Parameters

- **mconfig_data**: mconfig_data is a secondary pointer and must be called after MDF_FREE is used
- **wait_ticks**: wait time if a packet isn't immediately available

Structures

struct mconfig_whitelist_t

List of configured networks for each device.

Public Members

uint8_t **addr**[**MWIFI_ADDR_LEN**]

the address of the device

struct mconfig_data_t

Network configuration information.

Public Members

mwifi_config_t **config**

Mwifi AP configuration

mwifi_init_config_t **init_config**

Mwifi initialization configuration, Used only during debugging

uint8_t **custom**[32 + **CONFIG_MCONFIG_CUSTOM_EXTERN_LEN**]

Custom data for specific applications, such as: uuid, token, username, etc

uint16_t **whitelist_size**

The size of the device's whitelist

mconfig_whitelist_t **whitelist_data**[0]

Whitelist of devices

2.2.5 Mconfig Security

Header File

- mconfig/include/mconfig_security.h

Functions

mdf_err_t **mconfig_random**(void **rng_state*, uint8_t **output*, size_t *len*)

Generate an array of random numbers.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG

Parameters

- `rng_state`: The seed of the random number, not used temporarily, you can pass NULL
- `output`: Pointer to an array of random numbers
- `len`: The length of the array of random numbers

mdf_err_t **mconfig_dhm_gen_key**(uint8_t **param*, ssize_t *param_size*, uint8_t **privkey*, uint8_t **pubkey*)

Generate a public and private key using the DHM algorithm.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG

Parameters

- `param`: DHM configuration parameters
- `param_size`: The length of the DHM configuration parameter
- `privkey`: DHM' s public key
- `pubkey`: DHM' s private key

mdf_err_t **mconfig_rsa_gen_key**(char **privkey_pem*, char **pubkey_pem*)

Generate a public and private key using the RSA algorithm.

Return

- ESP_OK

- ESP_FAIL

Parameters

- `privkey_pem`: RSA public key in pem format
- `pubkey_pem`: RSA private key in pem format

mdf_err_t **mconfig_rsa_decrypt**(**const** uint8_t **ciphertext*, **const** char **privkey_pem*, void **plaintext*, size_t *plaintext_size*)

Use RSA' s public key to encrypt the data.

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_INVALID_ARG

Parameters

- `ciphertext`: Encrypted data
- `privkey_pem`: RSA' s private key in pem format
- `plaintext`: Unencrypted data
- `plaintext_size`: The length of unencrypted data

mdf_err_t **mconfig_rsa_encrypt**(**const** void **plaintext*, size_t *plaintext_size*, **const** char **pubkey_pem*, uint8_t **ciphertext*)

use RSA' s public key to encrypt the data

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_INVALID_ARG___MCONFIG_SECURITY_H___

Parameters

- `plaintext`: Unencrypted data
- `plaintext_size`: The length of unencrypted data
- `pubkey_pem`: RSA' s public key in pem format
- `ciphertext`: Encrypted data

Macros

MCONFIG_RSA_PRIVKEY_PEM_SIZE

MCONFIG_RSA_PUBKEY_PEM_SIZE

MCONFIG_RSA_KEY_BITS

MCONFIG_RSA_CIPHERTEXT_SIZE

MCONFIG_RSA_PLAINTEXT_MAX_SIZE

MCONFIG_RSA_EXPONENT

MCONFIG_RSA_PUBKEY_PEM_DATA_SIZE

PEM_BEGIN_PUBLIC_KEY

PEM_END_PUBLIC_KEY

PEM_BEGIN_PRIVATE_KEY

PEM_END_PRIVATE_KEY

MCONFIG_DH_PRIVKEY_LEN

MCONFIG_DH_PUBKEY_LEN

MCONFIG_AES_KEY_LEN

2.3 Mespnow API

Mespnow (Mesh ESP-NOW) is the encapsulation of ESP-NOW APIs, and it adds to ESP-NOW the retransmission filter, Cyclic Redundancy Check (CRC), and data fragmentation features.

2.3.1 Features

1. **Retransmission filter**: Mespnow adds a 16-bit ID to each fragment, and the redundant fragments with the same ID will be discarded.
2. **Fragmented transmission**: When the data packet exceeds the limit of the maximum packet size, Mespnow splits it into fragments before they are transmitted to the target device for reassembly.
3. **Cyclic Redundancy Check**: Mespnow implements CRC when it receives the data packet to ensure the packet is transmitted correctly.

2.3.2 Writing Applications

1. Prior to the use of Mwifi, Wi-Fi must be initialized;

2. Max number of the devices allowed in the network: No more than 20 devices, among which 6 encrypted devices at most, are allowed to be network configured;
3. Security: CCMP is used for encrpytion with a 16-byte key.

2.3.3 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mespnow, which includes:

- A simple application that demonstrates how to use Mespnow for the communication between two devices.

2.3.4 API Reference

Header File

- mespnow/include/mespnow.h

Functions

mdf_err_t **mespnow_add_peer**(wifi_interface_t *ifx*, **const** uint8_t **addr*, **const** uint8_t **lmk*)

add a peer to espnow peer list based on esp_now_add_peer(···). It is convenient to use simplified MACRO follows.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `ifx`: Wi-Fi interface that peer uses to send/receive ESPNOW data
- `addr`: peer mac address
- `lmk`: local master key. If lmk is NULL, ESPNOW data that this peer sends/receives isn't encrypted

mdf_err_t **mespnow_del_peer**(**const** uint8_t **addr*)

delete a peer from espnow peer list.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `addr`: peer mac address

mdf_err_t **mespnow_read**(*mespnow_trans_pipe_e pipe*, uint8_t **src_addr*, void **data*, size_t **size*, TickType_t *wait_ticks*)

read data from espnow

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pipe`: mespnow packet type
- `src_addr`: source address
- `data`: point to received data buffer
- `*size`: A non-zero pointer to the variable holding the length of out_value. In case out_value is not zero, will be set to the actual length of the value written.
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mespnow_write**(*mespnow_trans_pipe_e pipe*, **const** uint8_t **dest_addr*, **const** void **data*, size_t *size*, TickType_t *wait_ticks*)

write date package to espnow.

1. It is necessary to add device to espnow_peer befor send data to dest_addr.
2. When data_len to write is too long, it may fail duration some package and and the return value is the data len that actually sended.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pipe`: Pipe of data from espnnow
- `dest_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mespnow_deinit**(void)

deinit mespnow

Return

- ESP_OK
- ESP_FAIL

mdf_err_t **mespnow_init**(void)

init mespnow

Return

- ESP_OK
- ESP_FAIL_cplusplus **MESPNOW_H**

Macros

`MESPNOW_PAYLOAD_LEN`

< _cplusplus

`MDF_EVENT_MESPNOW_RECV`

`MDF_EVENT_MESPNOW_SEND`

Enumerations

`enum mespnow_trans_pipe_e`

Divide espnnow data into multiple pipes.

Values:

`MESPNOW_TRANS_PIPE_DEBUG`

debug packet: log, coredump, espnow_debug config, ack

`MESPNOW_TRANS_PIPE_CONTROL`

control packet

`MESPNOW_TRANS_PIPE_MCONFIG`

network configuration packet

`MESPNOW_TRANS_PIPE_RESERVED`

reserved for other functiond

`MESPNOW_TRANS_PIPE_MAX`

2.4 Mlink API

`Mlink` (MESH LAN communication protocol) is a solution for controlling ESP-WIFI-MESH network devices through APP, including: device discovery, control, upgrade, etc.

2.4.1 Application Examples

For Mlink examples, please refer to the directory development_kit, which includes:

- Smart light: Mesh App can control the color of the lamp, distribution network, upgrade, etc.

2.4.2 Mlink

Header File

- mlink/include/mlink.h

Macros

`MDF_EVENT_MLINK_SYSTEM_RESET`
< _cplusplus

`MDF_EVENT_MLINK_SYSTEM_REBOOT`

`MDF_EVENT_MLINK_SET_STATUS`

`MDF_EVENT_MLINK_GET_STATUS`

`MDF_EVENT_MLINK_SET_TRIGGER`

`MDF_EVENT_MLINK_BUFFER_FULL`

Enumerations

`enum mlink_protocol`
Type of protocol.

Values:

`MLINK_PROTO_HTTPD` = 0
Http protocol communication

`MLINK_PROTO_NOTICE` = 1
UDP protocol communication

2.4.3 Mlink Handle

Header File

- mlink/include/mlink_handle.h

Functions

mdf_err_t **mlink_add_device**(uint32_t *tid*, **const** char ***name*, **const** char ***version*)

Configuring basic information about the device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- **tid**: Unique identifier for the device type
- **name**: The name of device
- **version**: The version of device

mdf_err_t **mlink_device_set_name**(**const** char ***name*)

Set device name.

Return

- ESP_OK
- ESP_FAIL

Parameters

- **name**: The name of device

mdf_err_t **mlink_device_set_position**(**const** char ***position*)

Set device version.

Return

- ESP_OK
- ESP_FAIL

Parameters

- **position**: The position of device

const char ***mlink_device_get_name**()

Get device name.

Return Name of the device

const char ***mlink_device_get_position**()

Get device version.

Return Version of the device

int **mlink_device_get_tid()**

Get device tid.

Return Unique identifier for the device type

mdf_err_t **mlink_add_characteristic**(uint16_t *cid*, **const** char **name*, *characteristic_format_t format*, *characteristic_perms_t perms*, int *min*, int *max*, uint16_t *step*)

Add device characteristic information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cid`: Unique identifier for the characteristic
- `name`: The name of the characteristic
- `format`: The format of the characteristic
- `perms`: The permissions of the characteristic
- `min`: The min of the characteristic
- `max`: The max of the characteristic
- `step`: The step of the characteristic

mdf_err_t **mlink_add_characteristic_handle**(*mlink_characteristic_func_t get_value_func*, *mlink_characteristic_func_t set_value_func*)

Increase the device' s characteristic handler.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `get_value_func`: [description]
- `set_value_func`: [description]

mdf_err_t **mlink_handle**(**const** uint8_t **src_addr*, **const** *mlink_httpd_type_t* **type*, **const** void **data*, size_t *size*)

Handling requests from the APP.

Note This function is deprecated. Use ‘mlink_handle_request’ function

Return

- ESP_OK
- ESP_FAIL

Parameters

- `src_addr`: Source address of the device
- `type`: Type of data
- `data`: Requested message
- `size`: The length of the requested data

mdf_err_t **mlink_set_handle**(**const** char ***name*, **const** *mlink_handle_func_t* *func*)

Add or modify a request handler.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `name`: The name of the handler
- `func`: The pointer of the handler

mdf_err_t **mlink_handle_request**(*mlink_handle_data_t* **handle_data*)

Call the handler in the request list.

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_NOT_SUPPORTED_cplusplus **MLINK_HANDLE_H**

Parameters

- `handle_data`: The data type of the parameter of the handler

Structures

struct mlink_handle_data_t

The data type of the parameter of the handler.

Public Members

const char ***req_data**

Received request data

ssize_t **req_size**

The length of the received request data

mlink_httpd_format_t **req_fromat**

The format of the received request data

char ***resp_data**

Response data to be sent

ssize_t **resp_size**

The length of response data to be sent

mlink_httpd_format_t **resp_fromat**

The format of response data to be sent

Type Definitions

typedef *mdf_err_t* (***mlink_handle_func_t**)(*mlink_handle_data_t* *data)

Type of request handler.

typedef *mdf_err_t* (***mlink_characteristic_func_t**)(uint16_t cid, void *value)

Get the type of callback function that sets the characteristic value.

Enumerations

enum characteristic_perms_t

Permissions for the characteristics.

< _cplusplus

Values:

CHARACTERISTIC_PERMS_READ = 1 << 0

The characteristic of the device are readable

CHARACTERISTIC_PERMS_WRITE = 1 << 1

The characteristic of the device are writable

CHARACTERISTIC_PERMS_TRIGGER = 1 << 2

The characteristic of the device can be triggered

CHARACTERISTIC_PERMS_RW = CHARACTERISTIC_PERMS_READ | CHARACTERISTIC_PERMS_WRITE

The characteristic of the device are readable & writable

`CHARACTERISTIC_PERMS_RT` = CHARACTERISTIC_PERMS_READ | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are readable & triggered

`CHARACTERISTIC_PERMS_WT` = CHARACTERISTIC_PERMS_WRITE | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are writable & triggered

`CHARACTERISTIC_PERMS_RWT` = CHARACTERISTIC_PERMS_RW | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are readable & writable & triggered

`enum characteristic_format_t`
Format for the characteristic.

Values:

`CHARACTERISTIC_FORMAT_NONE`
Invalid format

`CHARACTERISTIC_FORMAT_INT`
characteristic is a number format

`CHARACTERISTIC_FORMAT_DOUBLE`
characteristic is a double format

`CHARACTERISTIC_FORMAT_STRING`
characteristic is a string format

2.4.4 Mlink Httpd

Header File

- mlink/include/mlink_httpd.h

Functions

mdf_err_t `mlink_httpd_start`(void)
Start http server.

Return

- ESP_OK
- ESP_FAIL

mdf_err_t `mlink_httpd_stop`(void)
Stop http server.

Return

- ESP_OK

- ESP_FAIL

mdf_err_t **mlink_httpd_read**(*mlink_httpd_t* ***request*, TickType_t *wait_ticks*)

Receive data from http server.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `request`: Request data
- `wait_ticks`: Waiting timeout

mdf_err_t **mlink_httpd_write**(**const** *mlink_httpd_t* **response*, TickType_t *wait_ticks*)

Send data to http server.

Return

- ESP_OK
- ESP_FAIL_cplusplus **MLINK_HTTPD_H**

Parameters

- `response`: Response data
- `wait_ticks`: Waiting timeout

Structures

struct mlink_httpd_type_t

Http server data type.

Public Members

uint16_t **sockfd**

File descriptor for tcp

uint8_t **format**

Http body data format

uint8_t **from**

Data request source

bool **resp**

Whether to respond to request data

uint16_t **received**

Received

struct mlink_httpd_t

Http server data.

Public Members

mlink_httpd_type_t **type**

Http server data type

bool **group**

Send a package as a group

size_t **addrs_num**

Number of addresses

uint8_t ***addrs_list**

List of addresses

size_t **size**

Length of data

char ***data**

Pointer of Data

Enumerations

enum mlink_httpd_from_t

Data request source.

< _cplusplus

Values:

MLINK_HTTPD_FROM_DEVICE = 1

Request from within the ESP-WIFI-MESH network

MLINK_HTTPD_FROM_SERVER = 2

Request from server

enum mlink_httpd_format_t

Http body data format.

Values:

MLINK_HTTPD_FORMAT_NONE

Invalid format

`MLINK_HTTPD_FORMAT_HEX`
data is a hex format

`MLINK_HTTPD_FORMAT_JSON`
data is a json format

`MLINK_HTTPD_FORMAT_HTML`
data is a html format

2.4.5 Mlink Notice

Header File

- mlink/include/mlink_notice.h

Functions

mdf_err_t **mlink_notice_init()**
Initialize mlink mdns and udp.

< _cplusplus

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mlink_notice_deinit()**
Deinitialize mlink mdns and udp.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mlink_notice_write(const** char *_t *message*, size_t *size*, **const** uint8_t *_t *addr*)
Inform Mesh-App to initiate a request.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL_cplusplus **MLINK_NOTICE_H**

Parameters

- `message`: Type of message requested

- `size`: Length of the message
- `addr`: Address of the device to be requested

2.4.6 Mlink Utils

Header File

- mlink/include/mlink_utils.h

Functions

uint8_t *`mlink_mac_str2hex(const` char *_mac_str_, uint8_t *_mac_hex_)

Convert mac from string format to hex.

< _cplusplus

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `mac_str`: String format mac
- `mac_hex`: Hex format mac

char *`mlink_mac_hex2str(const` uint8_t *_mac_hex_, char *_mac_str_)

Convert mac from hex format to string.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `mac_hex`: Hex format mac
- `mac_str`: String format mac

uint8_t *`mlink_mac_ap2sta(const` uint8_t *_ap_mac_, uint8_t *_sta_mac_)

Convert mac address from ap to sta.

Each ESP32 chip has MAC addresses for Station (STA), Access Point (AP), Bluetooth low energy (BT) and Local Area Network (LAN). Their address values are incremented by one, i.e. LAN Mac = BT Mac + 1 = AP Mac + 2 = STA Mac + 3.

For example:

- MAC for STA: `xx:xx:xx:xx:xx:00`
- MAC for AP : `xx:xx:xx:xx:xx:01`
- MAC for BT : `xx:xx:xx:xx:xx:02`
- MAC for LAN: `xx:xx:xx:xx:xx:03`

The device' s STA address is used For ESP-WIFI-MESH communication.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `ap_mac`: Access Point address in hexadecimal format
- `sta_mac`: Station address in hexadecimal format

uint8_t *`mlink_mac_bt2sta(const` uint8_t *_bt_mac_, uint8_t *_sta_mac_)

Convert mac address from bt to sta.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL_cplusplus **MLINK_UTILS_H**

Parameters

- `bt_mac`: Access Point address in hexadecimal format
- `sta_mac`: Station address in hexadecimal format

2.4.7 Mlink Json

Header File

- mlink/include/mlink_json.h

Functions

esp_err_t **__mlink_json_parse**(**const** char *_json_str_, **const** char *_key_, void *_value_, int _value_type_)

esp_err_t mlink_json_parse(const char *json_str, const char *key, void *value) Parse the json formatted string

Note does not support the analysis of array types

Return

- ESP_OK
- ESP_FAIL

Parameters

- `json_str`: The string pointer to be parsed
- `key`: Build value pairs
- `value`: You must ensure that the incoming type is consistent with the post-resolution type
- `value_type`: Type of parameter

esp_err_t **__mlink_json_pack**(char **_json_str_, **const** char *_key_, int _value_, int _value_type_)

mlink_json_pack(char *json_str, const char *key, int/double/char value); Create a json string

Note if the value is double or float type only retains the integer part, requires complete data calling mlink_json_pack_double()

Return

- lenght: generates the length of the json string
- ESP_FAIL

Parameters

- `json_str`: Save the generated json string
- `key`: Build value pairs
- `value`: This is a generic, support long / int / float / char / char* / char []
- `value_type`: Type of parameter

ssize_t **mlink_json_pack_double**(char **_json_ptr_, **const** char *_key_, double _value_)

Create a double type json string, Make up for the lack of mdf_json_pack()

Return

- lenght: generates the length of the json string
- ESP_FAIL_cplusplus **MLINK_JOSN_H**

Parameters

- `json_ptr`: Save the generated json string
- `key`: Build value pairs
- `value`: The value to be stored

Macros

`mlink_json_parse`(json_str, key, value)

`mlink_json_pack`(json_str, key, value)

Enumerations

`enum mlink_json_type`

The type of the parameter.

< _cplusplus

Values:

`MLINK_JSON_TYPE_NONE` = 0

Invalid parameter type

`MLINK_JSON_TYPE_INT8` = 1

The type of the parameter is int8_t / uint8_t

`MLINK_JSON_TYPE_INT16` = 10

The type of the parameter is int16_t / uint16_t

`MLINK_JSON_TYPE_INT32` = 100

The type of the parameter is int32_t / uint32_t

`MLINK_JSON_TYPE_FLOAT` = 1000

The type of the parameter is float

`MLINK_JSON_TYPE_DOUBLE` = 10000

The type of the parameter is double

`MLINK_JSON_TYPE_STRING` = 100000

The type of the parameter is string

`MLINK_JSON_TYPE_POINTER` = 1000000

The type of the parameter is point

2.5 Mupgrade API

`Mupgrade` (Mesh Upgrade) is a solution for simultaneous over-the-air (OTA) upgrading of multiple ESP-WIFI-MESH devices on the same wireless network by efficient routing of data flows. It features automatic retransmission of failed fragments, data compression, reverting to an earlier version, firmware check, etc.

2.5.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mupgrade, which includes:

- Send a request for data to server via AP, in order to complete the upgrading of the overall Mesh network.

2.5.2 API Reference

Header File

- mupgrade/include/mupgrade.h

Functions

mdf_err_t **mupgrade_firmware_init**(**const** char *_name_, size_t _size_)

Initialize the upgrade status and erase the upgrade partition.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_ERR_MUPGRADE_FIRMWARE_PARTITION

Parameters

- `name`: Unique identifier of the firmware
- `size`: Total length of firmware, If image size is not yet known, pass OTA_SIZE_UNKNOWN and call `mupgrade_firmware_download_finished()` after the firmware download is complete.

mdf_err_t **mupgrade_firmware_download**(**const** void *_data_, size_t _size_)

Write firmware to flash.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT
- MDF_ERR_MUPGRADE_FIRMWARE_FINISH

Parameters

- `data`: Pointer to the firmware
- `size`: The length of the data

mdf_err_t **mupgrade_firmware_download_finished**(size_t *image_size*)

Finish OTA update and validate newly written app image.

Attention Only called `mupgrade_firmware_init()` firmware size unknown at root node

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT

Parameters

- `image_size`: of new OTA app image

mdf_err_t **mupgrade_firmware_check**(**const** esp_partition_t **partition*)

Check if the firmware is generated by this project.

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_INVALID

Parameters

- `partition`: The partition where the firmware to be checked is located

mdf_err_t **mupgrade_firmware_send**(**const** uint8_t **dest_addrs*, size_t *dest_addrs_num*, *mupgrade_result_t* **result*)

Root sends firmware to other nodes.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT
- MDF_ERR_MUPGRADE_DEVICE_NO_EXIST

Parameters

- `dest_addrs`: Destination nodes of mac
- `dest_addrs_num`: Number of destination nodes
- `result`: Must call mupgrade_result_free to free memory

mdf_err_t **mupgrade_firmware_stop()**

Stop Root to send firmware to other nodes.

- MDF_ERR_NOT_SUPPORTED

Return

- MDF_OK

mdf_err_t **mupgrade_result_free**(*mupgrade_result_t* **result*)

Free memory in the results list.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG

Parameters

- `result`: Pointer to device upgrade status

mdf_err_t **mupgrade_handle**(**const** uint8_t **addr*, **const** void **data*, size_t *size*)

Handling upgrade data sent by ROOT.

Attention Only called at the non-root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD
- MDF_ERR_MUPGRADE_NOT_INIT

Parameters

- `addr`: root address
- `data`: Upgrade instructions or firmware
- `size`: The length of the data

mdf_err_t **mupgrade_root_handle**(**const** uint8_t **addr*, **const** void **data*, size_t *size*)

Handling the status of non-root responses.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_TIMEOUT

Parameters

- `addr`: Non-root address
- `data`: State of non-root response
- `size`: The length of the data

mdf_err_t **mupgrade_get_status**(*mupgrade_status_t* **status*)

Get the status of the upgrade.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_ERR_NOT_SUPPORTED

Parameters

- `status`: The status of the upgrade

mdf_err_t **mupgrade_version_fallback**()

Upgrade error back to previous version.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mupgrade_stop**()

Stop upgrading.

Return

- MDF_OK
- MDF_FAIL_cplusplus **MUPGRADE_H**

Structures

struct mupgrade_packet_t

Firmware packet.

Public Members

uint8_t **type**
: Type of packet, MUPGRADE_TYPE_DATA

uint16_t **seq**
: Sequence

uint16_t **size**
: Size

uint8_t **data**[**MUPGRADE_PACKET_MAX_SIZE**]
: Firmware

struct mupgrade_status_t
: Status packet.

Public Members

uint8_t **type**
: Type of packet, MUPGRADE_TYPE_STATUS

char **name**[32]
: Unique identifier of the firmware

mdf_err_t **error_code**
: Upgrade status

size_t **total_size**
: Total length of firmware

size_t **written_size**
: The length of the flash has been written

uint8_t **progress_array**[0]
: Identify if each packet of data has been written

struct mupgrade_config_t
: Mupgrade config.

Public Members

QueueHandle_t **queue**
: Queue handle

esp_ota_handle_t **handle**
: OTA handle

const esp_partition_t ***partition**

Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get.

uint32_t **start_time**

Start time of the upgrade

mupgrade_status_t **status**

Upgrade status

struct mupgrade_result_t

List of devices' status during the upgrade process.

Public Members

size_t **unfinished_num**

The number of devices to be upgraded

uint8_t ***unfinished_addr**

MAC address of devices to be upgraded

size_t **successed_num**

The number of devices that succeeded to upgrade

uint8_t ***successed_addr**

MAC address of devices that succeeded to upgrade

size_t **requested_num**

The number of devices to be upgraded

uint8_t ***requested_addr**

This address is used to buffer the result of the request during the upgrade process

Macros

MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT

mupgrade error code definition

< _cplusplus Uninitialized firmware configuration

MDF_ERR_MUPGRADE_FIRMWARE_PARTITION

Partition table error

MDF_ERR_MUPGRADE_FIRMWARE_INVALID

Non-project generated firmware

MDF_ERR_MUPGRADE_FIRMWARE_INCOMPLETE

The firmware received by the device is incomplete

MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD
: Firmware write flash error

MDF_ERR_MUPGRADE_FIRMWARE_FINISH
: The firmware has been written to completion

MDF_ERR_MUPGRADE_DEVICE_NO_EXIST
: The device that needs to be upgraded does not exist

MDF_ERR_MUPGRADE_SEND_PACKET_LOSS
: Request device upgrade status failed

MDF_ERR_MUPGRADE_NOT_INIT
: Upgrade configuration is not initialized

MDF_ERR_MUPGRADE_STOP
: Upgrade configuration is not initialized

MDF_EVENT_MUPGRADE_STARTED
: enumerated list OF MDF event id

  The device starts to upgrade

MDF_EVENT_MUPGRADE_STATUS
: Proactively report progress

MDF_EVENT_MUPGRADE_FINISH
: The upgrade is complete and the new firmware will run after the reboot

MDF_EVENT_MUPGRADE_STOPED
: Stop upgrading

MDF_EVENT_MUPGRADE_FIRMWARE_DOWNLOAD
: Start firmware write flash

MDF_EVENT_MUPGRADE_SEND_FINISH
: Send the firmware to other devices to complete

MUPGRADE_PACKET_MAX_SIZE
: Firmware subcontract upgrade.

  Maximum length of a single packet transmitted

MUPGRADE_PACKET_MAX_NUM
: The maximum number of packets

MUPGRADE_GET_BITS(data, bits)
: Bit operations to get and modify a bit in an array.

MUPGRADE_SET_BITS(data, bits)

MUPGRADE_TYPE_DATA
: Type of packet.

`MUPGRADE_TYPE_STATUS`

2.6 Mwifi API

`Mwifi` (Wi-Fi Mesh) is the encapsulation of ESP-WIFI-MESH APIs, and it adds to ESP-WIFI-MESH the retransmission filter, data compression, fragmented transmission, and P2P multicast features.

2.6.1 Features

1. **Retransmission filter**: As ESP-WIFI-MESH won' t perform data flow control when transmitting data downstream, there will be redundant fragments in case of unstable network or wireless interference. For this, Mwifi adds a 16-bit ID to each fragment, and the redundant fragments with the same ID will be discarded.
2. **Fragmented transmission**: When the data packet exceeds the limit of the maximum packet size, Mwifi splits it into fragments before they are transmitted to the target device for reassembly.
3. **Data compression**: When the packet of data is in Json and other similar formats, this feature can help reduce the packet size and therefore increase the packet transmitting speed.
4. **P2P multicast**: As the multicasting in ESP-WIFI-MESH may cause packet loss, Mwifi uses a P2P (peer-to-peer) multicasting method to ensure a much more reliable delivery of data packets.

2.6.2 Writing Applications

Mwifi supports using ESP-WIFI-MESH under self-organized mode, where only the root node that serves as a gateway of Mesh network can require the LwIP stack to transmit/receive data to/from an external IP network.

Mwifi also supports the way to fix the root node. In this case, the nodes in the entire ESP-WIFI-MESH network cannot communicate with the external IP network through wifi. If you need to communicate with the external IP network, you need to use other methods.

If you want to call Wi-Fi API to connect/disconnect/scan, please pause Mwifi first. Self organized networking must be disabled before the application calls any Wi-Fi driver APIs.

Initialization

Wi-Fi Initialization

Prior to the use of Mwifi, Wi-Fi must be initialized. `esp_wifi_set_ps` (indicates whether Mesh network enters into sleep mode) and `esp_mesh_set_6m_rate` (indicates the minimum packet transmittig speed) must be set before `esp_wifi_start`.

The following code snippet demonstrates how to initialize Wi-Fi.

```
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();

MDF_ERROR_ASSERT(nvs_flash_init());

tcpip_adapter_init();
MDF_ERROR_ASSERT(esp_event_loop_init(NULL, NULL));
MDF_ERROR_ASSERT(esp_wifi_init(&cfg));
MDF_ERROR_ASSERT(esp_wifi_set_storage(WIFI_STORAGE_RAM));
MDF_ERROR_ASSERT(esp_wifi_set_mode(WIFI_MODE_STA));
MDF_ERROR_ASSERT(esp_wifi_set_ps(WIFI_PS_NONE));
MDF_ERROR_ASSERT(esp_mesh_set_6m_rate(false));
MDF_ERROR_ASSERT(esp_wifi_start());
```

Mwifi Initialization

In general, you don' t need to modify Mwifi and just use its default configuration. But if you want to modify it, you may use `make menuconfig`. See the detailed configuration below:

1. ROOT

Parameters	Description
vote_percentage	Vote percentage threshold above which the node becoms a root
vote_max_count	Max multiple voting each device can have for the self-healing of a MESH network
back-off_rssi	RSSI threshold below which connections to the root node are not allowed
scan_min_count	The minimum number of times a device should scan the beacon frames from other devices before it becomes a root node
root_conflicts_enable	Allow more than one root in one network
root_healing_ms	Time lag between the moment a root node is disconnected from the network and the moment the devices start electing another root node

2. **Capacity**

Parameters	Description
capacity_num	Network capacity, defining max number of devices allowed in the MESH network
max_layer	Max number of allowed layers
max_connection	Max number of MESH softAP connections

3. **Stability**

Parameters	Description
assoc_expire_ms	Period of time after which a MESH softAP breaks its association with inactive leaf nodes
beacon_interval_ms	Mesh softAP beacon interval
passive_scan_ms	Mesh station passive scan duration
monitor_duration_ms	Period (ms) for monitoring the parent' s RSSI. If the signal stays weak throughout the period, the node will find another parent offering more stable connection
cnx_rssi	RSSI threshold above which the connection with a parent is considered strong
select_rssi	RSSI threshold for parent selection. Its value should be greater than SWITCH_RSSI
switch_rssi	RSSI threshold below which a node selects a parent with better RSSI
attempt_count	Parent selection fail times, if the scan times reach this value, device will disconnect with associated children and join self-healing
monitor_ie_count	Allowed number of changes a parent node can introduce into its information element (IE), before the leaf nodes must update their own IEs accordingly

4. **Transmission**

Parameters	Description
xon_qsize	Number of MESH buffer queues
retransmit_enable	Enable retransmission on mesh stack
data_drop_enable	If a root is changed, enable the new root to drop the previous packet

The following code snippet demonstrates how to initialize Mwifi.

```
/*  Mwifi initialization */
wifi_init_config_t cfg = MWIFI_INIT_CONFIG_DEFAULT();
MDF_ERROR_ASSERT(mwifi_init(&cfg));
```

Configuration

The modified Mwifi configuration will only be effective after Mwifi is rebooted.

You may configure Mwifi by using the struct mwifi_config_t. See the configuration parameters below:

Parameters	Description
router_ssid	SSID of the router
router_password	Router password
router_bssid	BSSID is equal to the router' s MAC address. This field must be configured if more than one router shares the same SSID. You can avoid using BSSIDs by setting up a unique SSID for each router. This field must also be configured if the router is hidden
mesh_id	Mesh network ID. Nodes sharing the same MESH ID can communicate with one another
mesh_password	Password for secure communication between devices in a MESH network
mesh_type	Only MESH_IDLE, MESH_ROOT, and MESH_NODE device types are supported. MESH_ROOT and MESH_NODE are only used for routerless solutions
channel	Mesh network and the router will be on the same channel
channel_switch_disable	If this value is not set, when "attempt" (mwifi_init_config_t) times is reached, it will change to a full channel scan for a network that could join.
router_switch_disable	If the BSSID is specified and this value is not also set, when the router of this specified BSSID fails to be found after "attempt" (mwifi_init_config_t) times, the whole network is allowed to switch to another router with the same SSID.

The following code snippet demonstrates how to configure Mwifi.

```
mwifi_config_t config = {
    .router_ssid     = CONFIG_ROUTER_SSID,
    .router_password = CONFIG_ROUTER_PASSWORD,
    .mesh_id         = CONFIG_MESH_ID,
    .mesh_password   = CONFIG_MESH_PASSWORD,
    .mesh_type       = CONFIG_MESH_TYPE,
    .channel         = CONFIG_ROUTER_CHANNEL,
};

MDF_ERROR_ASSERT(mwifi_set_config(&config));
```

Start Mwifi

After starting Mwifi, the application should check for Mwifi events to determine when it has connected to the network.

1. MDF_EVENT_MWIFI_ROOT_GOT_IP: the root node gets the IP address and can communicate with the external network;
2. MDF_EVENT_MWIFI_PARENT_CONNECTED: connects with the parent node for data communication between devices.

```
MDF_ERROR_ASSERT(mdf_event_loop_init(event_loop_cb));
MDF_ERROR_ASSERT(mwifi_start());
```

2.6.3 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mwifi, which includes:

- Connect to the external network: This can be achieved by the root node via TCP.
- Routerless: It involves a fixed root node which communicates with the external devices via UART.
- MQTT example: Implement each node to communicate with the MQTT broker by the root node via MQTT.
- Channel switching: When the router channel changes, Mesh switches its channel accordingly.(In development)
- Low power consumption: The leaf nodes stop functioning as softAP and enter into sleep mode.(In development)

2.6.4 API Reference

Header File

- mwifi/include/mwifi.h

Functions

esp_err_t **esp_wifi_vnd_mesh_get**(mesh_assoc_t **mesh_assoc*, mesh_chain_layer_t **mesh_chain*)

Get mesh networking IE.

Return

- ESP_OK

- ESP_FAIL

Parameters

- `mesh_assoc`: pointer mesh networking IE.
- `mesh_chain`: pointer mesh chain layer infomation.

mdf_err_t **mwifi_init**(**const** *mwifi_init_config_t* **config*)

Mwifi initialization.

Attention 1. This API should be called after WiFi is started.

Attention 2. This API must be called before all other Mwifi API can be called.

Attention 3. Always use WIFI_INIT_CONFIG_DEFAULT macro to init the config to default values, this can guarantee all the fields got correct value when more fields are added into wifi_init_config_t in future release. If you want to set your owner initial values, overwrite the default values which are set by WIFI_INIT_CONFIG_DEFAULT.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT
- MDF_ERR_MWIFI_INITED

Parameters

- `config`: Mwifi initialization configuration.

mdf_err_t **mwifi_deinit**(void)

Deinit mwifi Free all resource allocated in mwifi_init and stop Mwifi task.

Attention This API should be called if you want to remove Mwifi from the system.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_set_init_config**(**const** *mwifi_init_config_t* **init_config*)

Set Mwifi configuration.

Attention This API shall be called between esp_mesh_init() and esp_mesh_start().

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `init_config`: Use MWIFI_INIT_CONFIG_DEFAULT() to initialize the default values.

mdf_err_t **mwifi_get_init_config**(*mwifi_init_config_t* **init_config*)

Get Mwifi init configuration.

Return [description]

Parameters

- `init_config`: pointer to Mwifi init configuration.

mdf_err_t **mwifi_set_config**(const *mwifi_config_t* **config*)

Set the configuration of the AP.

Attention This API shall be called between esp_mesh_init() and esp_mesh_start().

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `config`: pointer to AP configuration

mdf_err_t **mwifi_get_config**(*mwifi_config_t* **config*)

Get the configuration of the AP.

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `config`: pointer to AP configuration

mdf_err_t **mwifi_start**(void)

Start Mwifi according to current configuration.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_stop**(void)

Stop mwifi.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_restart()**

restart mwifi

Attention This API should be called when the Mwifi configuration is modified.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

bool **mwifi_is_started**(void)

whether wifi_mesh is running

Return

- true
- flase

bool **mwifi_is_connected**(void)

Whether the node is already connected to the parent node.

Return

- true
- flase

void **mwifi_print_config()**

Print mesh configuration information.

mdf_err_t **mwifi_write**(**const** uint8_t **dest_addrs*, **const** *mwifi_data_type_t* **data_type*, **const** void **data*, size_t *size*, bool *block*)

Send a packet to any node in the mesh network.

Attention 1. If data encryption is enabled, you must ensure that the task that calls this function is greater than 4KB.

1. When sending data to the root node, if the destination address is NULL, the root node receives it using mwifi_root_read(). If the destination address is the mac address of the root node or MWIFI_ADDR_ROOT, the root node receives it using mwifi_read()

Return

- MDF_OK

- MDF_ERR_MWIFI_NOT_START

Parameters

- `dest_addrs`: The address of the final destination of the packet If the packet is to the root and "dest_addr" parameter is NULL
- `data_type`: The type of the data If the default configuration is used, this parameter is NULL
- `data`: Pointer to a sending wifi mesh packet
- `size`: The length of the data
- `block`: Whether to block waiting for data transmission results

mdf_err_t **__mwifi_read**(uint8_t **src_addr*, *mwifi_data_type_t* **data_type*, void **data*, size_t **size*, TickType_t *wait_ticks*, uint8_t *type*)

Receive a packet targeted to self over the mesh network.

Attention Judging the allocation of buffers by the type of the parameter 'data' The memory of `data` is externally allocated, and the memory is larger than the total length expected to be received. The memory of `data` is allocated internally by `mwifi_read`, which needs to be released after the call.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `src_addr`: The address of the original source of the packet
- `data_type`: The type of the data
- `data`: Pointer to the received mesh packet To apply for buffer space externally, set the type of the data parameter to be (char *) or (uint8_t *) To apply for buffer space internally, set the type of the data parameter to be (char **) or (uint8_t **)
- `size`: A non-zero pointer to the variable holding the length of out_value. In case out_value is not zero, will be set to the actual length of the value written.
- `wait_ticks`: Wait time if a packet isn' t immediately available
- `type`: Buffer space when reading data

mdf_err_t **mwifi_root_write**(**const** uint8_t **dest_addrs*, size_t *dest_addrs_num*, **const** *mwifi_data_type_t* **data_type*, **const** void **data*, size_t *size*, bool *block*)

The root sends a packet to the device in the mesh.

Attention 1. If data encryption is enabled, you must ensure that the task that calls this function is greater than 8KB

1. This API is only used at the root node
2. Can send data to multiple devices at the same time
3. Packets are only transmitted down

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_DISCONNECTED
- ESP_ERR_MESH_OPT_UNKNOWN
- ESP_ERR_MESH_EXCEED_MTU
- ESP_ERR_MESH_NO_MEMORY
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_QUEUE_FULL
- ESP_ERR_MESH_NO_ROUTE_FOUND
- ESP_ERR_MESH_DISCARD

Parameters

- `dest_addrs`: The address of the final destination of the packet
- `dest_addrs_num`: Number of destination addresses
- `data_type`: The type of the data
- `data`: Pointer to a sending wifi mesh packet
- `size`: The length of the data
- `block`: Whether to block waiting for data transmission results

mdf_err_t **__mwifi_root_read**(uint8_t **src_addr*, *mwifi_data_type_t* **data_type*, void **data*, size_t **size*, TickType_t *wait_ticks*, uint8_t *type*)

receive a packet targeted to external IP network root uses this API to receive packets destined to

external IP network root forwards the received packets to the final destination via socket. This API is only used at the root node

Attention 1. This API is only used at the root node

1. Judging the allocation of buffers by the type of the parameter ‘data’ The memory of `data` is externally allocated, and the memory is larger than the total length expected to be received. The memory of `data` is allocated internally by `mwifi_read`, which needs to be released after the call.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `src_addr`: the address of the original source of the packet
- `data_type`: the type of the data
- `data`: Pointer to the received mesh packet To apply for buffer space externally, set the type of the data parameter to be (char *) or (uint8_t *) To apply for buffer space internally, set the type of the data parameter to be (char **) or (uint8_t **)
- `size`: The length of the data
- `wait_ticks`: wait time if a packet isn’ t immediately available(0:no wait, portMAX_DELAY:wait forever)
- `type`: Buffer space when reading data

mdf_err_t **`mwifi_post_root_status`**(bool *status*)

Post the toDS state to the mesh stack, Usually used to notify the child node, whether the root is successfully connected to the server.

Attention This API is only for the root.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `status`: this state represents whether the root is able to access external IP network

bool **mwifi_get_root_status()**

Get the toDS state from the mesh stack, Whether the root is successfully connected to the server.

Return

- true
- flase

int8_t **mwifi_get_parent_rssi()**

Get the RSSI of the parent node.

Return RSSI of the parent_cplusplus **MWIFI_H**

Structures

struct mwifi_init_config_t

Mwifi initialization configuration.

Public Members

uint8_t **vote_percentage**

Mesh root configuration.

Vote percentage threshold above which the node becoms a root

uint8_t **vote_max_count**

Max multiple voting each device can have for the self-healing of a MESH network

int8_t **backoff_rssi**

RSSI threshold below which connections to the root node are not allowed

uint8_t **scan_min_count**

Minimum scan times before being a root

bool **root_conflicts_enable**

Allow more than one root in one network

uint16_t **root_healing_ms**

Time lag between the moment a root node is disconnected from the network and the moment the devices start electing another root node

uint16_t **capacity_num**

Mesh network capacity configuration.

Network capacity, defining max number of devices allowed in the MESH network

uint8_t **max_layer_deprecated**

Deprecated, shouldn't use it

uint8_t **max_connection**

Max number of MESH softAP connections

uint16_t **assoc_expire_ms**

Mesh network stability configuration.

Period of time after which a MESH softAP breaks its association with inactive leaf nodes

uint16_t **beacon_interval_ms**

Mesh softAP beacon interval

uint16_t **passive_scan_ms**

Mesh station passive scan duration

uint16_t **monitor_duration_ms**

Period (ms) for monitoring the parent's RSSI. If the signal stays weak throughout the period, the node will find another parent offering more stable connection

int8_t **cnx_rssi**

RSSI threshold above which the connection with a parent is considered strong

int8_t **select_rssi**

RSSI threshold for parent selection. Its value should be greater than switch_rssi

int8_t **switch_rssi**

RSSI threshold below which a node selects a parent with better RSSI

uint8_t **attempt_count**

Parent selection fail times, if the scan times reach this value, device will disconnect with associated children and join self-healing

uint8_t **monitor_ie_count**

Allowed number of changes a parent node can introduce into its information element (IE), before the leaf nodes must update their own IEs accordingly

uint8_t **xon_qsize**

Mesh network data transmission configuration.

Number of MESH buffer queues

bool **retransmit_enable**

Enable a source node to retransmit data to the node from which it failed to receive ACK

bool **data_drop_enable**

If a root is changed, enable the new root to drop the previous packet

uint16_t **max_layer**

Max number of allowed layers

uint8_t **topology**

Topology of mesh network, can be tree or chain

struct mwifi_config_t

Mwifi AP configuration.

Public Members

char **router_ssid**[32]

Information about the router connected to the root. This item does not need to be set in the routerless scheme.

SSID of the router

char **router_password**[64]

Password of router

uint8_t **router_bssid**[6]

BSSID, if this value is specified, users should also specify "router_switch_disable"

uint8_t **mesh_id**[6]

Information about the mesh internal connected.

Mesh network ID. Nodes sharing the same MESH ID can communicate with one another

char **mesh_password**[64]

Password for secure communication between devices in a MESH network

mwifi_node_type_t **mesh_type**

Only MWIFI_MESH_IDLE, MWIFI_MESH_ROOT, MWIFI_MESH_NODE and MWIFI_MESH_LEAF device types are supported. Routerless solutions must be configured as MWIFI_MESH_ROOT or MWIFI_MESH_NODE MWIFI_MESH_IDLE: The node type is selected by MESH according to the network conditions. MWIFI_MESH_ROOT: The Device is the root node MWIFI_MESH_NODE: The device is a non-root node and contains intermediate nodes and leaf nodes. MWIFI_MESH_LEAF: The device is a leaf node, closes the softap, and is used in a low-power solutions.

uint8_t **channel**

Channel, mesh and router will be on the same channel

uint8_t **channel_switch_disable**

If this value is not set, when "attempt" (*mwifi_init_config_t*) times is reached, device will change to a full channel scan for a network that could join.

uint8_t **router_switch_disable**

If the BSSID is specified and this value is not also set, when the router of this specified BSSID fails to be found after "attempt" (*mwifi_init_config_t*) times, the whole network is allowed to switch to another router with the same SSID. The new router might also be on a different channel.

There is a risk that if the password is different between the new switched router and the previous one, the mesh network could be established but the root will never connect to the new switched router.

struct mwifi_data_type_t

Mwifi packet type.

Public Members

bool **compression**

Enable data compression. If the data sent is json format or string, use data compression to increase the data transmission rate.

bool **upgrade**

Upgrade packet flag

uint8_t **communicate**

Mesh data communication method, There are three types: MWIFI_COMMUNICATE_UNICAST, MWIFI_COMMUNICATE_MULTICAST, MWIFI_COMMUNICATE_BROADCAST

bool **group**

Send a package as a group

uint8_t **reserved**

reserved

uint8_t **protocol**

Type of transmitted application protocol

uint32_t **custom**

Type of transmitted application data

Macros

MWIFI_PAYLOAD_LEN

< _cplusplus Max payload size(in bytes)

MWIFI_ADDR_LEN

Length of MAC address

MWIFI_ADDR_NONE

MWIFI_ADDR_ROOT

MWIFI_ADDR_ANY

All node in the mesh network

MWIFI_ADDR_BROADCAST
: Other node except the root

MWIFI_ADDR_IS_EMPTY(addr)

MWIFI_ADDR_IS_ANY(addr)

MWIFI_ADDR_IS_BROADCAST(addr)

MDF_ERR_MWIFI_NOT_INIT
: Mwifi error code definition.

  Mwifi isn’ t initialized

MDF_ERR_MWIFI_INITED
: Mwifi has been initialized

MDF_ERR_MWIFI_NOT_START
: Mwifi isn’ t started

MDF_ERR_MWIFI_ARGUMENT
: Illegal argument

MDF_ERR_MWIFI_EXCEED_PAYLOAD
: Packet size exceeds payload

MDF_ERR_MWIFI_TIMEOUT
: Timeout

MDF_ERR_MWIFI_DISCONNECTED
: Disconnected with parent on station interface

MDF_ERR_MWIFI_NO_CONFIG
: Router not configured

MDF_ERR_MWIFI_NO_FOUND
: Routes or devices not found

MDF_ERR_MWIFI_NO_ROOT
: Routes or devices not found

MDF_EVENT_MWIFI_STARTED
: enumerated list of Mwifi event id

  Mwifi is started

MDF_EVENT_MWIFI_STOPPED
: Mwifi is stopped

MDF_EVENT_MWIFI_CHANNEL_SWITCH
: Channel switch

`MDF_EVENT_MWIFI_CHILD_CONNECTED`
: A child is connected on softAP interface

`MDF_EVENT_MWIFI_CHILD_DISCONNECTED`
: A child is disconnected on softAP interface

`MDF_EVENT_MWIFI_ROUTING_TABLE_ADD`
: Routing table is changed by adding newly joined children

`MDF_EVENT_MWIFI_ROUTING_TABLE_REMOVE`
: Routing table is changed by removing leave children

`MDF_EVENT_MWIFI_PARENT_CONNECTED`
: Parent is connected on station interface

`MDF_EVENT_MWIFI_PARENT_DISCONNECTED`
: Parent is disconnected on station interface

`MDF_EVENT_MWIFI_NO_PARENT_FOUND`
: No parent found

`MDF_EVENT_MWIFI_LAYER_CHANGE`
: Layer changes over the Mwifi network

`MDF_EVENT_MWIFI_TODS_STATE`
: State represents if root is able to access external IP network

`MDF_EVENT_MWIFI_VOTE_STARTED`
: The process of voting a new root is started either by children or by root

`MDF_EVENT_MWIFI_VOTE_STOPPED`
: The process of voting a new root is stopped

`MDF_EVENT_MWIFI_ROOT_ADDRESS`
: The root address is obtained. It is posted by Mwifi stack automatically.

`MDF_EVENT_MWIFI_ROOT_SWITCH_REQ`
: Root switch request sent from a new voted root candidate

`MDF_EVENT_MWIFI_ROOT_SWITCH_ACK`
: Root switch acknowledgment responds the above request sent from current root

`MDF_EVENT_MWIFI_ROOT_ASKED_YIELD`
: Root is asked yield by a more powerful existing root.

`MDF_EVENT_MWIFI_SCAN_DONE`
: if self-organized networking is disabled

`MDF_EVENT_MWIFI_NETWORK_STATE`
: network state, such as whether current mesh network has a root.

MDF_EVENT_MWIFI_STOP_RECONNECTION
: the root stops reconnecting to the router and non-root devices stop reconnecting to their parents.

MDF_EVENT_MWIFI_FIND_NETWORK
: when the channel field in mesh configuration is set to zero, mesh stack will perform a full channel scan to find a mesh network that can join, and return the channel value after finding it.

MDF_EVENT_MWIFI_ROUTER_SWITCH
: if users specify BSSID of the router in mesh configuration, when the root connects to another router with the same SSID, this event will be posted and the new router information is attached.

MDF_EVENT_MWIFI_CHANNEL_NO_FOUND
: The router' s channel is not set.

MDF_EVENT_MWIFI_ROOT_GOT_IP

MDF_EVENT_MWIFI_ROOT_LOST_IP

MDF_EVENT_MWIFI_EXCEPTION
: Some abnormal situations happen, eg. disconnected too many times

CONFIG_MWIFI_ROOT_CONFLICTS_ENABLE
: CONFIG_MWIFI_ROOT_CONFLICTS_ENABLE

CONFIG_MWIFI_RETRANSMIT_ENABLE
: CONFIG_MWIFI_RETRANSMIT_ENABLE

MWIFI_INIT_CONFIG_DEFAULT()

MWIFI_RSSI_THRESHOUD_DEFAULT()

mwifi_read(src_addr, data_type, data, size, wait_ticks)

mwifi_root_read(src_addr, data_type, data, size, wait_ticks)

Type Definitions

typedef uint8_t **mwifi_node_type_t**

Enumerations

enum node_type_t
: Device type.

Values:

MWIFI_MESH_IDLE
: hasn' t joined the mesh network yet

MWIFI_MESH_ROOT
: the only sink of the mesh network. Has the ability to access external IP network

`MWIFI_MESH_NODE`
intermediate device. Has the ability to forward packets over the mesh network

`MWIFI_MESH_LEAF`
has no forwarding ability

`MWIFI_MESH_STA`
connect to router with a standlone Wi-Fi station mode, no network expansion capability

enum mwifi_communication_method
Mesh network internal data transmission method.

Values:

`MWIFI_COMMUNICATE_UNICAST`
Send data by unicast.

`MWIFI_COMMUNICATE_MULTICAST`
Send data by multicast.

`MWIFI_COMMUNICATE_BROADCAST`
Send data by broadcast.

enum mwifi_data_memory_t
Buffer space when reading data.

Values:

`MWIFI_DATA_MEMORY_MALLOC_INTERNAL` = 1
Buffer space is requested by internal when reading data

`MWIFI_DATA_MEMORY_MALLOC_EXTERNAL` = 2
Buffer space is requested by external when reading data

2.7 Mdebug API

`Mdebug` (Mesh Network debug) is an important debugging solution used in ESP-MDF. It is designed to efficiently obtain ESP-MDF device logs through wireless espnow protocol, TCP protocol, serial port, etc., so that it can be read more conveniently and quickly. The device log information can then be analyzed according to the extracted logs.

2.7.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/Mdebug, which includes:

- The log information of the child node is transmitted to the root node through the *ESP-Mesh* network, and then outputted through the serial port of the root node or transmitted to the server through http;

- Output the desired log information through the console.

1. UART enable

The serial port is enabled and the log information will be printed out via `vprintf`. Enable the following program:

```
if (config->log_uart_enable) { /**< Set log uart enable */
    mdebug_log_init();
} else {
    mdebug_log_deinit();
}
```

2. Flash enable

Write flash enable to store log information in flash. Enable the following program:

```
if (config->log_flash_enable) { /**< Set log flash enable */
      mdebug_flash_init();
} else {
      mdebug_flash_deinit();
}
```

2.1 Log information access

1. The log is initialized, creating two text flash memory spaces, using the main functions as `esp_vfs_spiffs_register`, `esp_spiffs_info`, `sprintf`, `fopen`;

```
/**< Create spiffs file and number of files */
esp_vfs_spiffs_conf_t conf = {
    .base_path       = "/spiffs",
    .partition_label = NULL,
    .max_files       = MDEBUG_FLASH_FILE_MAX_NUM,
    .format_if_mount_failed = true
};
```

2. Write the log data, and write down the address of the write pointer. In order to address the next log write flash, use the main function as `fwrite`. The following procedure:

```
/**< Record the address of each write pointer */
fseek(g_log_info[g_log_index].fp, g_log_info[g_log_index].size,␣
↪SEEK_SET);
```

(下页继续)

(续上页)

```
ret = fwrite(data, 1, size, g_log_info[g_log_index].fp);
```

3. Read the log data, and write down the address of the read pointer. For the next time to read the log from the flash, address the address, use the main function as `fread`. The following procedure:

```
/**< Record the pointer address for each read */
flash_log_info_t *log_info = g_log_info + ((g_log_index + 1 +␣
↪i) % MDEBUG_FLASH_FILE_MAX_NUM);
fseek(log_info->fp, log_info->offset, SEEK_SET);

ret = fread(data, 1, MIN(*size - read_size, log_info->size -␣
↪log_info->offset), log_info->fp);
```

4. The data is erased. When the data is full, the data pointer will be cleared, and the main function `rewind` will be restarted from the beginning or the end of the file header address. The following procedure:

```
/**< Erase file address pointer */
for (size_t i = 0; i < MDEBUG_FLASH_FILE_MAX_NUM; i++) {
    g_log_index          = 0;
    g_log_info[i].size   = 0;
    g_log_info[i].offset = 0;
    rewind(g_log_info[i].fp);
    mdf_info_save(MDEBUG_FLASH_STORE_KEY, g_log_info,␣
↪sizeof(flash_log_info_t) * MDEBUG_FLASH_FILE_MAX_NUM);
}
```

注解:

1. The header of the log data is an added timestamp. It is only used as an experiment, and there is no real-time calibration. Users can modify it according to their own needs. The following procedure:

```
/**< Get the current timestamp */
time(&now);
localtime_r(&now, &timeinfo);
strftime(strtime_buf, 32, "\n[%Y-%m-%d %H:%M:%S] ", &timeinfo);
```

2. Extract and select valid string data information. Because the log information in the MDF has unnecessary data at the beginning and the end, it needs to be removed. The following

procedure removes unwanted data from the head and tail:

```
/**< Remove the header and tail that appear in the string in the log */
if (log_data->data[0] == 0x1b) {
    data = log_data->data + 7;
    size = log_data->size - 12;
}
```

---

3. ESPNOW enable

Espnow can be enabled to send the log data of the child node to the root node through espnow, so that the log information of the child node is read from the root node. Enable the following program:

```
/**< config espnow enable */
if (config->log_espnow_enable) {
    mdebug_espnow_init();
} else {
    mdebug_espnow_deinit();
}
```

Write the log to espnow. The procedure is as follows:

```
/**< write log information by espnow */
if (!g_log_config->log_espnow_enable && !MDEBUG_ADDR_IS_EMPTY(g_log_config->dest_addr)) {
    log_data->type |= MDEBUG_LOG_TYPE_ESPNOW;
}

if (log_data->type & MDEBUG_LOG_TYPE_ESPNOW) {
    mdebug_espnow_write(g_log_config->dest_addr, log_data->data,
                        log_data->size, MDEBUG_ESPNOW_LOG, pdMS_TO_TICKS(MDEBUG_LOG_
↪TIMEOUT_MS));
}
```

2.7.2 Mdebug Console

Header File

- mdebug/include/mdebug_console.h

Functions

mdf_err_t **mdebug_console_init**(void)

Initialize console module.

< _cplusplus

- Initialize the console
- Register help commands
- Initialize filesystem
- Create console handle task

Attention Baudrate should not greater than 2030400 if console is enable

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_console_deinit**(void)

De-initialize console module Call this once when done using console module functions.

Return

- MDF_OK
- MDF_FAIL

void **mdebug_cmd_register_common**(void)

Register frequently used system commands.

- version: Get version of chip and SDK
- heap: Get the current size of free heap memory
- restart: Software reset of the chip
- reset: Clear device configuration information
- log: Set log level for given tag
- coredump: Get core dump information_cplusplus **MDF_CONSOLE_DEBUG_H**

2.7.3 Mdebug Espnow

Header File

- mdebug/include/mdebug_espnow.h

Functions

mdf_err_t **mdebug_espnow_init**(void)

Initialize the wireless debug receiver.

- register espnow log redirect function
- Create mdebug espnow send and mdebug log send task
- Create log send queue

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_espnow_deinit**(void)

De-initialize the wireless debug receiver.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_espnow_write**(**const** uint8_t *_dest_addr_, **const** void *_data_, size_t _size_, *mdebug_espnow_t* _type_, TickType_t _wait_ticks_)

Send debug data with ESP-NOW.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `dest_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `type`: Type of data
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mdebug_espnow_read**(uint8_t *_src_addr_, void *_data_, size_t *_size_, *mdebug_espnow_t* *_type_, TickType_t _wait_ticks_)

receive debug data with ESP-NOW

Return

- ESP_OK
- ESP_FAIL_cplusplus **MDF_ESPNOW_DEBUG_H**

Parameters

- `src_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `type`: Type of data
- `wait_ticks`: wait time if a packet isn't immediately available

Structures

`struct mdebug_coredump_packet_t`

Core dump data structure.

Public Members

uint8_t **type**

Type of packet

int16_t **size**

Size of data

int16_t **seq**

Sequence

uint8_t **data**[230]

data

Enumerations

`enum mdebug_espnow_t`

Type of data sent during wireless debugging.

< _cplusplus

Values:

`MDEBUG_ESPNOW_COREDUMP` = 1

Core dump information

`MDEBUG_ESPNOW_CONSOLE`

Remotely call local terminal commands

`MDEBUG_ESPNOW_LOG`
Log information

`enum [anonymous]`
Type of core dump data.

Values:

`MDEBUG_COREDUMP_BEGIN` = 1
Start transferring core dump data

`MDEBUG_COREDUMP_DATA`
Send core dump data

`MDEBUG_COREDUMP_END`
End core dump data transfer

2.7.4 Mdebug Flash

Header File

- mdebug/include/mdebug_flash.h

Functions

mdf_err_t **mdebug_flash_init**()
Init mdebug_flash Create Several files under the spiffs folder,open the file.Open the file for the storage of the next step data.paramters MDEBUG_FLASH_FILE_MAX_NUM if files sizes change.

< _cplusplus

Return

- MDF-OK

mdf_err_t **mdebug_flash_deinit**()
Deinit medbug_flash If you open the file, close the file accordingly.

Return

- MDF-OK

mdf_err_t **mdebug_flash_write**(**const** char **data*, size_t *size*)
Write memory data in flash.

Note Don’ t include timestamp in interface input data

Return

- MDF_OK

Parameters

- `data`: Data from the flash’s spiffs files in the log
- `size`: Size from the flash’s spiffs files in the log

mdf_err_t **mdebug_flash_read**(char **data*, size_t **size*)

Read memory data in flash.

Return

- MDF_OK
- read_size

Parameters

- `data`: Data from the flash’s spiffs files in the log
- `size`: Size from the flash’s spiffs files in the log

mdf_err_t **mdebug_flash_erase**()

Erase when the data and pointers is full.

Return

- MDF_OK

size_t **mdebug_flash_size**()

Create files size,For the data to be stored in the file for subsequent calls.paramters MDEBUG_FLASH_FILE_MAX_NUM if files sizes change.

Return

- size_cplusplus **MDF_DEBUG_FLASH_H**

2.7.5 Mdebug Log

Header File

- mdebug/include/mdebug_log.h

Functions

mdf_err_t **mdebug_log_get_config**(*mdebug_log_config_t* **config*)

Get the configuration of the log during wireless debugging.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `config`: The configuration of the log

mdf_err_t **mdebug_log_set_config**(**const** *mdebug_log_config_t* **config*)

Set the configuration of the log during wireless debugging.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `config`: The configuration of the log

mdf_err_t **mdebug_log_init**(void)

Init log mdebug.

- Set log mdebug configuration
- Create log mdebug task

Return

- MDF_OK

mdf_err_t **mdebug_log_deinit**(void)

De-initialize log mdebug Call this once when done using log mdebug functions.

Return

- MDF_OK_cplusplus **MDF_DEBUG_LOG_H**

Structures

struct mdebug_log_config_t

Log sending configuration.

< _cplusplus

Public Members

bool **log_uart_enable**
Serial port enable

bool **log_flash_enable**
Write the log to flash enable

bool **log_espnow_enable**
enable log transferred via ESP-NOW

uint8_t **dest_addr**[6]
Turn off log sending if all is zero

struct mdebug_log_queue_t
Type of log storage queue.

Public Members

uint16_t **size**
The length of the log data

mdebug_log_type_t **type**
Ways to send logs

char **data**[0]
Log data

Enumerations

enum mdebug_log_type_t
Set the send type.

Values:

MDEBUG_LOG_TYPE_ESPNOW = 1 << 1

MDEBUG_LOG_TYPE_FLASH = 1 << 2

2.7.6 Mdebug Partially Referenced Header File

Header File

- mdebug/include/mdebug.h

Macros

`MDEBUG_PRINTF`(fmt, ...)

Configuration mdebug print enable, whether the output information according to the client needs to know. please assign CONFIG_MDEBUG_PRINTF_ENABLE a value.

< _cplusplus

Note CONFIG_MDEBUG_PRINTF_ENABLE = 1 enable CONFIG_MDEBUG_PRINTF_ENABLE = 0 disable

`MDEBUG_ADDR_IS_EMPTY`(addr)

`MDF_EVENT_MDEBUG_FLASH_FULL`

_cplusplus **MDF_DEBUG_H**

2.8 Third Party API

The third-party items:

- Driver: drivers for different devices, such as frequently used buttons and LEDs
- Miniz: lossless, high performance data compression library

2.8.1 Driver

We can regard IoT solution project as a platform that contains different device drivers and features.

2.8.2 Miniz

Miniz is a lossless, high performance data compression library in a single source file that implements the zlib (RFC 1950) and Deflate (RFC 1951) compressed data format specification standards. It supports the most commonly used functions exported by the zlib library, but is a completely independent implementation so zlib' s licensing requirements do not apply. Miniz also contains simple to use functions for writing .PNG format image files and reading/writing/appending .ZIP format archives. Miniz' s compression speed has been tuned to be comparable to zlib' s, and it also has a specialized real-time compressor function designed to compare well against fastlz/minilzo.

2.9 Configuration Options

2.9.1 Introduction

ESP-MDF uses Kconfig system to provide a compile-time configuration mechanism. Kconfig is based around options of several types: integer, string, boolean. Kconfig files specify dependencies between options, default values of the options, the way the options are grouped together, etc.

Applications developers can use `make menuconfig` build target to edit components' configuration. This configuration is saved inside `sdkconfig` file in the project root directory. Based on `sdkconfig`, application build targets will generate `sdkconfig.h` file in the build directory, and will make sdkconfig options available to component makefiles.

2.9.2 Using sdkconfig.defaults

When updating ESP-MDF version, it is not uncommon to find that new Kconfig options are introduced. When this happens, application build targets will offer an interactive prompt to select values for the new options. New values are then written into `sdkconfig` file. To supress interactive prompts, applications can either define `BATCH_BUILD` environment variable, which will cause all prompts to be suppressed. This is the same effect as that of `V` or `VERBOSE` variables. Alternatively, `defconfig` build target can be used to update configuration for all new variables to the default values.

In some cases, such as when `sdkconfig` file is under revision control, the fact that `sdkconfig` file gets changed by the build system may be inconvenient. The build system offers a way to avoid this, in the form of `sdkconfig.defaults` file. This file is never touched by the build system, and must be created manually. It can contain all the options which matter for the given application. The format is the same as that of the `sdkconfig` file. Once `sdkconfig.defaults` is created, `sdkconfig` can be deleted and added to the ignore list of the revision control system (e.g. `.gitignore` file for git). Project build targets will automatically create `sdkconfig` file, populated with the settings from `sdkconfig.defaults` file, and the rest of the settings will be set to their default values. Note that when `make defconfig` is used, settings in sdkconfig will be overriden by the ones in `sdkconfig.defaults`.

2.9.3 Configuration Options Reference

Subsequent sections contain the list of available ESP-MDF options, automatically generated from Kconfig files. Note that depending on the options selected, some options listed here may not be visible by default in the interface of menuconfig.

By convention, all option names are upper case with underscores. When Kconfig generates sdkconfig and sdkconfig.h files, option names are prefixed with `CONFIG_`. So if an option `ENABLE_FOO` is defined in a Kconfig file and selected in menuconfig, then sdkconfig and sdkconfig.h files will have `CONFIG_ENABLE_FOO` defined. In this reference, option names are also prefixed with `CONFIG_`, same as in the source code.

2.9.4 Customisations

when the Kconfig tool generates Makefiles (the `auto.conf` file) its behaviour has been customised. In normal Kconfig, a variable which is set to "no" is undefined. In MDF's version of Kconfig, this variable is defined in the Makefile but has an empty value.

(Note that `ifdef` and `ifndef` can still be used in Makefiles, because they test if a variable is defined *and has a non-empty value*.)

When generating header files for C & C++, the behaviour is not customised - so `#ifdef` can be used to test if a boolean config item is set or not.

2.10 Error Codes Reference

This section lists various error code constants defined in ESP-MDF.

For general information about error codes in ESP-MDF, see *Error Handling*.

MDF_FAIL (-1): Generic mdf_err_t code indicating failure

MDF_OK (0): mdf_err_t value indicating success (no error)

MDF_ERR_NO_MEM **(0x100001)**: Out of memory

MDF_ERR_INVALID_ARG **(0x100002)**: Invalid argument

MDF_ERR_INVALID_STATE **(0x100003)**: Invalid state

MDF_ERR_INVALID_SIZE **(0x100004)**: Invalid size

MDF_ERR_NOT_FOUND **(0x100005)**: Requested resource not found

MDF_ERR_NOT_SUPPORTED **(0x100006)**: Operation or feature not supported

MDF_ERR_TIMEOUT **(0x100007)**: Operation timed out

MDF_ERR_INVALID_RESPONSE **(0x100008)**: Received response was invalid

MDF_ERR_INVALID_CRC **(0x100009)**: CRC or checksum was invalid

MDF_ERR_INVALID_VERSION **(0x10000a)**: Version was invalid

MDF_ERR_INVALID_MAC **(0x10000b)**: MAC address was invalid

MDF_ERR_NOT_INIT **(0x10000c)**: MAC address was invalid

MDF_ERR_BUF **(0x10000d)**: The buffer is too small

MDF_ERR_MWIFI_BASE **(0x200000)**: Starting number of MWIFI error codes

MDF_ERR_MWIFI_NOT_INIT **(0x200001)**: Mwifi isn't initialized

MDF_ERR_MWIFI_INITED **(0x200002)**: Mwifi has been initialized

MDF_ERR_MWIFI_NOT_START (**0x200003**): Mwifi isn’ t started

MDF_ERR_MWIFI_ARGUMENT (**0x200004**): Illegal argument

MDF_ERR_MWIFI_EXCEED_PAYLOAD (**0x200005**): Packet size exceeds payload

MDF_ERR_MWIFI_TIMEOUT (**0x200006**): Timeout

MDF_ERR_MWIFI_DISCONNECTED (**0x200007**): Disconnected with parent on station interface

MDF_ERR_MWIFI_NO_CONFIG (**0x200008**): Router not configured

MDF_ERR_MWIFI_NO_FOUND (**0x200009**): Routes or devices not found

MDF_ERR_MWIFI_NO_ROOT (**0x20000a**): Routes or devices not found

MDF_ERR_MESPNOW_BASE (**0x300000**): Starting number of MESPNOW error codes

MDF_ERR_MCONFIG_BASE (**0x400000**): Starting number of MCONFIG error codes

MDF_ERR_MUPGRADE_BASE (**0x500000**): Starting number of MUPGRADE error codes

MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT (**0x500001**): Uninitialized firmware configuration

MDF_ERR_MUPGRADE_FIRMWARE_PARTITION (**0x500002**): Partition table error

MDF_ERR_MUPGRADE_FIRMWARE_INVALID (**0x500003**): Non-project generated firmware

MDF_ERR_MUPGRADE_FIRMWARE_INCOMPLETE (**0x500004**): The firmware received by the device is incomplete

MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD (**0x500005**): Firmware write flash error

MDF_ERR_MUPGRADE_FIRMWARE_FINISH (**0x500006**): The firmware has been written to completion

MDF_ERR_MUPGRADE_DEVICE_NO_EXIST (**0x500007**): The device that needs to be upgraded does not exist

MDF_ERR_MUPGRADE_SEND_PACKET_LOSS (**0x500008**): Request device upgrade status failed

MDF_ERR_MUPGRADE_NOT_INIT (**0x500009**): Upgrade configuration is not initialized

MDF_ERR_MUPGRADE_STOP (**0x50000a**): Upgrade configuration is not initialized

MDF_ERR_MDEBUG_BASE (**0x600000**): Starting number of MDEBUG error codes

MDF_ERR_MLINK_BASE (**0x700000**): Starting number of MLINK error codes

MDF_ERR_CUSTOM_BASE (**0x800000**): Starting number of COUSTOM error codes

CHAPTER 3

ESP32 Hardware Reference

CHAPTER 4

API 指南

4.1 ESP-WIFI-MESH 基本概念困惑解答

本文用于回答初次接触 ESP-WIFI-MESH 时产生的疑惑。

4.1.1 准备工作

ESP-WIFI-MESH 是特为 ESP32 芯片研发的自组网技术。ESP32 使用基于双核的修定版本的 FreeRTOS，并使用 ESP-IDF 作为其官方开发框架。

1. 了解 ESP-WIFI-MESH 网络协议
2. **了解 ESP32 芯片：**
    - ESP32 技术参考手册
    - ESP32 技术规格书
3. 熟悉 ESP-IDF 开发指南
4. 了解 FreeRTOS

4.1.2 常见问题

ESP-WIFI-MESH 根节点相关

1. 根节点是什么意思？如何生成？

根节点是一个 ESP-WIFI-MESH 网络内唯一与外界（路由器或服务器）连接的节点，其产生方式可以有多种：

- **通过底层协议自行选举产生**，具体流程请参考 ESP-WIFI-MESH 组网流程说明。此方式下，需要手机或服务器告诉设备路由器的信息，作为选举的依据；
- **通过应用层设计固定某一个或某一类设备作为根节点**，此时周围的设备可以自动连上该类设备，此种情况即不需要路由或手机参与。但需注意，若多个设备都固定为根节点，则相互之间无法组网

2. 根节点挂了，网络是不是就崩了？

   分为两种情况：

   - **若网络为非固定根节点方案**，则网络内设备会在确认根节点无法链接的情况下，重新选出新的根节点，具体流程请参考 Root Node Failure。
   - **若网络为固定根节点方案**，在用户重新指派出新的根节点之前，网络无法正常工作。

ESP-WIFI-MESH 组网相关

1. 组网是否可以不需要路由器？

   ESP-WIFI-MESH 组网不依赖于路由器，只要确认了根节点即可。

2. 设备组网需要多久？

   设备组网需要的时间与设备的数量、相互之间的距离（相互之间的 Wi-Fi 信号强弱）、设备分布方式等因素有关系，数量越少、Wi-Fi 信号越强、分布越均匀，设备组网越快。

3. 一个路由器下，是否只能有一个 MESH 网络？

   分以下几种情形讨论：

   - **路由范围内的设备相互之间能通讯（直接或间接），则：**
     - 若 MESH ID 不同，则以 MESH ID 为标识组成不同的 MESH 网络。
     - **若 MESH ID 相同，则判断是否允许根节点冲突：**
       * **若允许根节点冲突**，但当网络不稳定导致生成其他根节点时，则不会进行排除操作，最终允许多个相同 ID 的 MESH 网络，每个网络各有一个根节点；
       * **若不允许根节点冲突**，但当网络不稳定导致生成其他根节点时，则会进行排除操作，最终只允许一个相同 ID 的 MESH 网络，该网络只有一个根节点；
   - 路由范围内的设备相互之间无法通讯（直接或间接），则：
   - 由于设备不知道其他网络的存在，则会单独形成各自的网络。

4. 一个 MESH 网络内，是否只能有一个根节点？

   是的，一个 MESH 网络内只能有一个根节点。

5. 假设有两个相同 MESH ID 的 MESH 网络，则新加入一个设备，会加到哪个网络?

   若不指定连接的父节点，则根据信号强度跟信号更强的节点产生关联。

ESP-WIFI-MESH 性能相关

1. 网络环境会不会影响 ESP-WIFI-MESH 性能?

   ESP-WIFI-MESH 是基于 Wi-Fi 通讯协议的方案，因此任何会影响 Wi-Fi 信号强弱的因素都会影响 ESP-WIFI-MESH 的性能。使用时建议选择较为空闲的信道，使用性能更好的天线等来提高 Wi-Fi 信号强度。

2. 设备支持多远距离的通讯?

   通讯距离取决于 Wi-Fi 信号强度和数据吞吐量需求，两个设备间信号越好，吞吐量要求越低，则通信距离越大。

其余

1. ESP-MDF 是什么，和 ESP-WIFI-MESH 有什么关系或区别?

   MDF，Mesh Development Framework，是基于 ESP-WIFI-MESH 的开发框架。ESP-WIFI-MESH 是 ESP-IDF 中的 MESH 网络通信协议模块。在 ESP-WIFI-MESH 组网、数据传输的基本功能上，添加了配网模块、OTA 模块、连云功能、本地控制、设备驱动等模块和应用示例，方便用户进行开发。用户可以根据需求选择基于 IDF 或 MDF 进行开发。

2. ESP32 芯片能读出来多个 Mac 地址，请问 ESP-WIFI-MESH 通讯是以哪个 Mac 地址

   ESP32 芯片有 STA、AP、BLE、Ethernet 4 个 Mac 地址，其依次 +1，即 Ethernet Mac= BLE Mac+1 = AP Mac+2 = STA Mac+3，例如假设：*xx:xx:xx:xx:xx:00* 为 STA Mac，则 *xx:xx:xx:xx:xx:01* 为 AP *xx:xx:xx:xx:xx:02* 为 BLE *xx:xx:xx:xx:xx:03* 为 Ethernet。

   在 ESP-WIFI-MESH 通讯时，以设备的 STA 地址为目标通讯。

3. 没有路由器，ESP-WIFI-MESH 能不能正常工作?

   ESP-WIFI-MESH 网络形成后，网络内部的通讯不依赖路由器，可以正常工作。

4.1.3 问题反馈

1. **可以通过以下途径进行问题反馈：**

   - 乐鑫 ESP-MDF 论坛页面
   - Github ESP-MDF 问题反馈页面

2. **进行问题反馈时，请提供以下信息：**

   - **硬件**：使用的开发板型号

- **错误描述**：问题复现的步骤、条件和出现的概率
- **ESP-MDF 版本信息**：使用 `git commit` 获取 ESP-MDF 的版本信息
- **日志**：设备完整的日志文件及 `build` 文件夹下的 elf 文件

4.2 错误处理

4.2.1 概述

本指南介绍了 ESP-MDF 错误调试方式，并提供了一些常见的错误处理方式。

4.2.2 错误调试

错误代码

大多数特定于 ESP-MDF 的函数使用*mdf_err_t* 类型来返回错误代码。*mdf_err_t* 是有符号整数类型。`MDF_OK` 代码表示成功（无错误），代码定义为零。各种 ESP-MDF 头文件定义可能的错误代码。通常这些定义以 "MDF_ERR_" 前缀开头。通用故障的常见错误代码（内存不足，超时，无效参数等）在 `mdf_err.h` 文件中定义。ESP-MDF 中的各种组件可以为特定情况定义附加的错误代码。使用 `MDF_ERROR_` 前缀开头的宏定义去检查返回值，出错时会直接返回文件名和行号，并通过 `esp_err_to_name()` 将值转换为错误代码的名称，以便更容易理解发生了哪个错误。

有关错误代码的完整列表，请参阅*错误代码*。

常用配置

建议仅在调试时启用此模式，而不是在生产中，请使用如下配置以便定位问题，参见: examples/sdkconfig.defaults:

```
#
# FreeRTOS
#
CONFIG_FREERTOS_UNICORE=y               # Debug portMUX Recursion
CONFIG_TIMER_TASK_STACK_DEPTH=3072      # FreeRTOS timer task stack size


#
# ESP32-specific
#
CONFIG_TASK_WDT_PANIC=y                 # Invoke panic handler on Task Watchdog timeout
CONFIG_TASK_WDT_TIMEOUT_S=10            # Task Watchdog timeout period (seconds)
CONFIG_ESP32_ENABLE_COREDUMP=y          # Core dump destination
```

(下页继续)

(续上页)

```
CONFIG_TIMER_TASK_STACK_SIZE=4096      # High-resolution timer task stack size
CONFIG_ESP32_ENABLE_COREDUMP_TO_FLASH=y # Select flash to store core dump
```

致命错误

CPU 异常或其他致命错误发生时，panic 处理程序会输出记录一些 CPU 寄存器和 Backtrace。Backtrace 包含 PC:SP 对。其中 PC 是程序计数器，SP 是堆栈指针。如果在中断内发生致命错误，则 Backtrace 可能包括来自被中断的任务和中断的 PC:SP 对。有关诊断不可恢复错误的说明，请参阅 致命错误。

使用 IDF Monitor

如果使用 IDF Monitor，程序计数器值将转换为代码位置（函数名称，文件名和行号）：

```
Guru Meditation Error: Core  0 panic'ed (StoreProhibited). Exception was unhandled.
Core 0 register dump:
PC      : 0x400d33ed  PS      : 0x00060f30  A0      : 0x800d1055  A1      : 0x3ffba950
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_started.
↪c:200

A2      : 0x00000000  A3      : 0x00000000  A4      : 0x00000000  A5      : 0x3ffbaaf4
A6      : 0x00000001  A7      : 0x00000000  A8      : 0x800d2306  A9      : 0x3ffbaa10
A10     : 0x3ffb0834  A11     : 0x3ffb3730  A12     : 0x40082784  A13     : 0x06ff1ff8
0x40082784: _calloc_r at ~/project/esp-mdf-master/esp-idf/components/newlib/syscalls.c:51

A14     : 0x3ffafff4  A15     : 0x00060023  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400d33ed:0x3ffba950 0x400d1052:0x3ffbaa60
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_started.
↪c:200

0x400d1052: main_task at ~/project/esp-mdf-master/esp-idf/components/esp32/cpu_start.
↪c:476
```

要查找发生致命错误的位置，请查看 “Backtrace” 行后面的行。致命错误位置是紧跟 “Backtrace” 行的后一行，后续行则显示调用堆栈。

未使用 IDF Monitor

如果未使用 IDF Monitor，您将只能获取到 Backtrace，您需要使用 `xtensa-esp32-elf-addr2line` 命令转换为代码位置:

```
Guru Meditation Error: Core  0 panic'ed (StoreProhibited). Exception was unhandled.
Core 0 register dump:
PC      : 0x400d33ed  PS      : 0x00060f30  A0      : 0x800d1055  A1      : 0x3ffba950
A2      : 0x00000000  A3      : 0x00000000  A4      : 0x00000000  A5      : 0x3ffbaaf4
A6      : 0x00000001  A7      : 0x00000000  A8      : 0x800d2306  A9      : 0x3ffbaa10
A10     : 0x3ffb0834  A11     : 0x3ffb3730  A12     : 0x40082784  A13     : 0x06ff1ff8
A14     : 0x3ffafff4  A15     : 0x00060023  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400d33ed:0x3ffba950 0x400d1052:0x3ffbaa60
```

跳转到您的工程下，并在终端输入如下命令:

```
xtensa-esp32-elf-addr2line -pfia -e build/*.elf Backtrace: 0x400d33ed:0x3ffba950␣
↪0x400d1052:0x3ffbaa60
```

转换结果如下:

```
0x00000bac: ?? ??:0
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_started.
↪c:200
0x400d1052: main_task at ~/project/esp-mdf-master/esp-idf/components/esp32/cpu_start.
↪c:476
```

内存调试

ESP-IDF 集成了用于请求堆信息，检测堆损坏和跟踪内存泄漏的工具，详见：Heap Memory Debugging，如果您使用的是 `mdf_mem.h` 中相关的 APIs，也能使用。*`mdf_mem_print_record()`* 可以打印所有未释放的内存，快速定位内存泄露的问题:

```
I (1448) [mdf_mem, 95]: Memory record, num: 4
I (1448) [mdf_mem, 100]: <mwifi : 181> ptr: 0x3ffc5f2c, size: 28
I (1458) [mdf_mem, 100]: <mwifi : 401> ptr: 0x3ffc8fd4, size: 174
I (1468) [mdf_mem, 100]: <get_started : 96> ptr: 0x3ffd3cd8, size: 1456
I (1468) [mdf_mem, 100]: <get_started : 66> ptr: 0x3ffd5400, size: 1456
```

注解:

1. 配置：通过 make menuconfig 开启 CONFIG_MDF_MEM_DEBUG；
2. 仅记录使用 MDF_*ALLOC 和 MDF_FREE 申请和释放的内存空间

任务调度

使用*mdf_mem_print_task()* 可以获取所有任务的运行状态、优先级和栈的剩余空间:

```
Task Name       Status  Prio    HWM     Task
main            R       1       1800    3
IDLE            R       0       1232    4
node_write_task B       6       2572    16
node_read_task  B       6       2484    17
Tmr Svc         B       1       1648    5
tiT             B       18      1576    7
MEVT            B       20      2080    10
eventTask       B       20      2032    8
MTXBLK          B       7       2068    11
MTX             B       10      1856    12
MTXON           B       11      2012    13
MRX             B       13      2600    14
MNWK            B       15      2700    15
mdf_event_loop  B       10      2552    6
esp_timer       B       22      3492    1
wifi            B       23      1476    9
ipc0            B       24      636     2

Current task, Name: main, HWM: 1800
Free heap, current: 170884, minimum: 169876
```

注解:

1. 调用*mdf_mem_print_task()* 会挂起所有任务，这一过程可能持续较长时间，因此建议本函数仅在调试时使用；
2. 配置：通过 make menuconfig 开启 CONFIG_FREERTOS_USE_TRACE_FACILITY 和 CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS；
3. 状态：R（Ready）代表准备态，B（blocked）代表阻塞态；
4. 剩余空间：HWM（High Water Mark）应不小于 512 byte，防止栈溢出。

4.2.3 常见错误

编译错误

1. MDF_PATH **未设置**:

未设置 MDF_PATH 环境变量编译时找不到 esp-mdf:

```
Makefile:8: /project.mk: No such file or directory
make: *** No rule to make target '/project.mk'.  Stop.
```

- **解决方法：** 输入如下命令进行配置:

```
$ export MDF_PATH=~/project/esp-mdf
```

输入如下命令进行验证:

```
$ echo $MDF_PATH
~/project/esp-mdf
```

2. **获取工程不完整**

通过 *git clone* 获取工程时，没有带有 --recursive 标志，以至于 esp-mdf 的子工程没有被获取:

```
~/project/esp-mdf/project.mk:9: ~/project/esp-mdf/esp-idf/make/project.mk: No such file␣
↪or directory
make: *** No rule to make target '~/project/esp-mdf/esp-idf/make/project.mk'.  Stop.
```

- **解决方法：** 运行如下命令重新获取子工程

```
`shell cd $MDF_PAHT git submodule update --init `
```

烧录错误

1. **串口权限不足**

在 linux 系统下，TTY 设备隶属于 dialout 用户组，普通用户没有权限访问:

```
serial.serialutil.SerialException: [Errno 13] could not open port /dev/ttyUSB0: [Errno␣
↪13] Permission denied: '/dev/ttyUSB0'
```

- **解决方法：**

1. 直接修改串口权限:

```
sudo chmod 0666 /dev/ttyUSB0
```

2. 将用户添加至 dialout 用户组，该用户就会具备访问 TTY 等串口设备的权限:

```
sudo gpasswd --add <user> dialout
```

2. `make flash` 错误

python 与 pyserial 版本不兼容:

```
AttributeError: 'Serial' object has no attribute 'dtr'
AttributeError: 'module' object has no attribute 'serial_for_url'
```

- **解决方法：** 运行如下命令，如果问题还未解决，你可以到 esptool issues 查找是否有相同的问题:

```
sudo apt-get update
sudo apt-get upgrade
sudo pip install esptool
sudo pip install --ignore-installed pyserial
```

ESP-WIFI-MESH 错误

1. **设备无法连接路由器**

路由器名称与密码配置正确，但设备无法连接路由器，日志如下:

```
I (2917) mesh: [SCAN][ch:1]AP:1, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (2917) mesh: [FAIL][1]root:0, fail:1, normal:0, <pre>backoff:0

I (3227) mesh: [SCAN][ch:1]AP:1, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3227) mesh: [FAIL][2]root:0, fail:2, normal:0, <pre>backoff:0

I (3527) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3527) mesh: [FAIL][3]root:0, fail:3, normal:0, <pre>backoff:0

I (3837) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3837) mesh: [FAIL][4]root:0, fail:4, normal:0, <pre>backoff:0

I (4137) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (4137) mesh: [FAIL][5]root:0, fail:5, normal:0, <pre>backoff:0
```

- **原因：**
    1. 未配置信道：ESP-WIFI-MESH 为了更快速的进行组网，因此只在固定的一个信道上进行扫描，因此必须配置 ESP-WIFI-MESH 的工作信道；
    2. 连接隐藏路由器：当 ESP-WIFI-MESH 连接隐藏路由器时，必须配置路由器的 BSSID；
    3. 路由器信道通常是非固定的，路由器会根据网络情况进行信道迁移。
- **解决方式：**
    1. 固定路由器的信道并配置路由器的信道和 BSSID；
    2. 通过 `mwifi_scan()` 让设备自动去获取路由器信息，但这会增加组网时间。

4.3 Mconfig

[English]

Mconfig (Mesh Network Configuration) 是 ESP-WIFI-MESH 配网的一种方案，目的是将配置信息便捷、高效地传递给 ESP-WIFI-MESH 设备。

Mconfig 使用的 App 为 ESP-Mesh App，配网过程中使用 RSA 算法进行密钥协商，使用 128-AES 算法进行数据加密，使用 CRC 算法进行校验和验证。

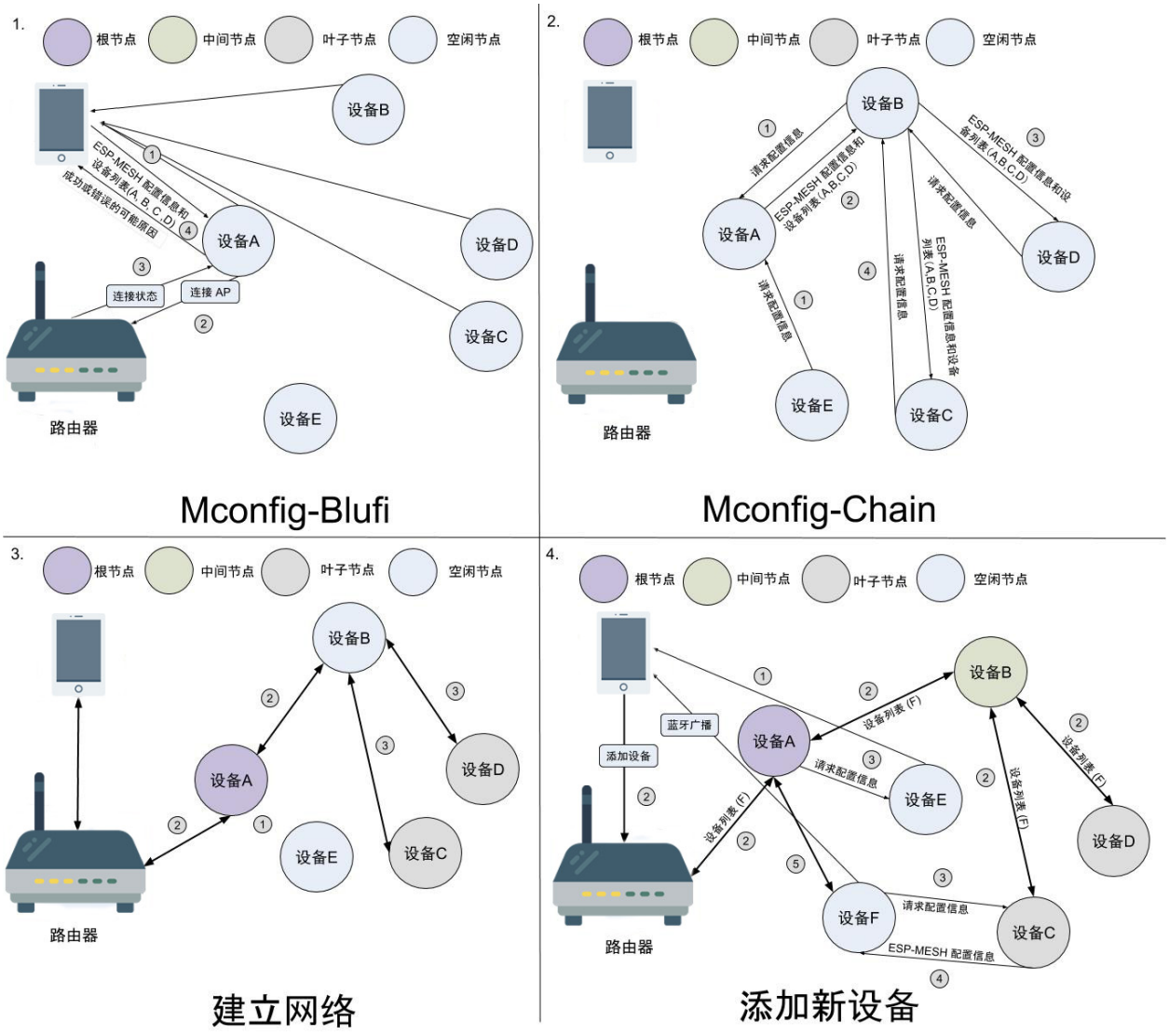
4.3.1 概念

术语	描述
设备	任何属于或可以属于 ESP-WIFI-MESH 网络的设备
白名单	包含设备的 MAC 地址列表和设备公钥的 MD5 值。可通过蓝牙扫描生成或扫描设备二维码导入，用于链式配网时设备校验
配置信息	路由器和 ESP-WIFI-MESH 相关配置
AES	高级加密标准（英语：Advanced Encryption Standard，缩写：AES），在密码学中又称 Rijndael 加密法，是美国联邦政府采用的一种区块加密标准
RSA	RSA 加密算法是一种非对称加密算法。在公开密钥加密和电子商业中 RSA 被广泛使用
CRC	循环冗余校验 (Cyclic Redundancy Check, CRC) 是一种根据网络数据包或电脑文件等数据产生简短固定位数校验码的一种散列函数，主要用来检测或校验数据传输或者保存后可能出现的错误

4.3.2 整体流程

请见下方流程图详细说明。

1. ***Mconfig-BluFi*：App 通过蓝牙给单个设备配网。**


1.
根节点
中间节点
叶子节点
空闲节点
设备B
设备A
设备D
设备C
设备E
路由器
连接状态
连接 AP
Mconfig-Blufi
2.
请求配置信息
ESP-MESH 配置信息和设备列表(A,B,C,D)
Mconfig-Chain
3.
建立网络
4.
蓝牙广播
添加设备
设备F
请求配置信息
ESP-MESH 配置信息
添加新设备

1. App 扫描设备的蓝牙广播包，生成白名单，即设备列表（A、B、C、D），并连接信号最好的设备（A），将配置信息与设备列表传输给设备（A）；
2. 设备（A）使用配置信息，尝试连接路由器；
3. 设备（A）根据路由器返回的状态，校验配置信息是否正确：
4. **设备（A）将配网状态返回给 App，同时：**
    - 若信息正确：通过 Wi-Fi 的 beacon，通知其余设备来请求配网信息；
    - 若信息错误：将失败的可能原因返回给 App，并返回步骤 i 时的状态。

2. ***Mconfig-Chain*：已配网设备给未配网设备传递配网信息，通过 Wi-Fi 传输。**
    1. 设备（B，E）收到设备（A）通知后，向设备（A）发送配网请求；
    2. **设备（A）对请求的设备进行校验：**
        - 设备（E）不在设备列表中，因此忽略设备（E）的请求；
        - 设备（B）在设备列表中，设备（A）通过 Wi-Fi 将配置信息发送给设备（B）
    3. 设备（B）获取配置信息将重复步骤 i, 给设备（C，D）配网。

3. **建立网络**
    1. 根节点选择：设备（A，B，C，D）根据其与路由器之间的信号强度动态选择，协商生成根节点设备（A）
    2. 第二层形成：一旦根节点连接到路由器，根节点范围内的空闲节点设备（B）将开始与设备（A）连接，从而形成网络的第二层
    3. 剩余层的形成：设备（C，D）与中间父节点设备（B）连接，形成网络的叶子节点

4. **添加新设备**
    1. App 扫描设备的蓝牙广播包，并弹出是否添加设备（E，F）的提示框，当选择添加设备（F）时，App 会发送添加新设备（F）的指令；
    2. 根节点设备（A）收到指令后，转发给 Mesh 网络内的设备，设备（A，B，C，D）收到添加新设备（F）的指令将会开启 Mconfig-Chain；
    3. 设备（F）向与其信息最好的设备（C）发送配网请求;
    4. 设备（C）通过 Wi-Fi 将配置信息发送给设备（F）
    5. 设备（F）获取到配置信息后连入 MESH 网络

4.3.3 安全机制

- **数据安全**
    - 使用非对称加密算法 RSA 生成出一个共享随机密钥；

  - 使用密钥通对称加密算法 AES 对所有的配置信息进行加密，以保证数据传输过程的安全；

- **设备安全**
  - Mconfig-BluFi：通过 App 端对设备的信号强度进行筛选，可以在一定程度上防止伪造设备攻击；
  - Mconfig-Chain：通过设置配网窗口期，仅在其窗口期内接收配网请求，来防止伪造设备攻击。
- **身份安全（即数字签名，可选）**
  - 在为每一个设备烧录时由烧录工具为每一个设备随机提供一个单独的 RSA 公私钥对，并将公钥上传云端以备验证；
  - 对存储 RSA 公私钥对的 flash 分区进行加密。

4.3.4 注意事项

若要自定义配网方式，请注意：

- 密码校验：ESP-WIFI-MESH 非根节点不检查路由器的信息，只检查 ESP-WIFI-MESH 网络内部的配置是否正确；如果 ESP-WIFI-MESH 网络内部的配置正确，但是路由器密码错误，当非根节点成为根节点时就连不上路由器。因此，在对非根节点进行配网时需要校验路由器的密码；

4.3.5 Mconfig-BluFi

Mconfig-BluFi 是基于 BluFi （一种由 Espressif 定义的蓝牙配网协议）的配网协议，并在 BluFi 的基础上增加了广播包的定义、RSA 加密和身份认证。配网中涉及的硬件有：手机、设备和路由器，分为设备发现、密钥协商、数据通信，校验配置四个过程。

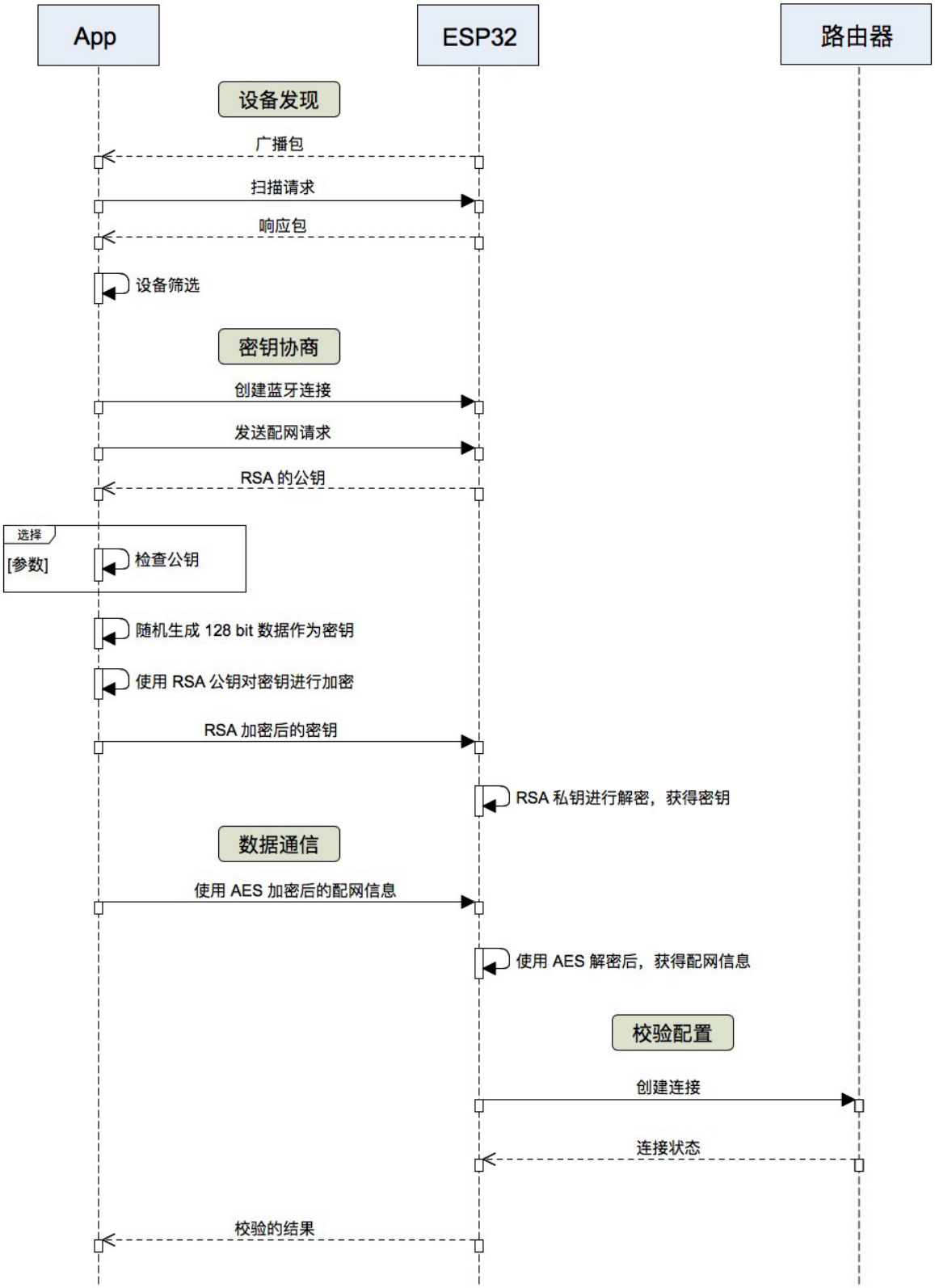
注解： 使用 Mconfig-BluFi 必须使能蓝牙的协议栈，使能蓝牙的协议栈注意如下事项：

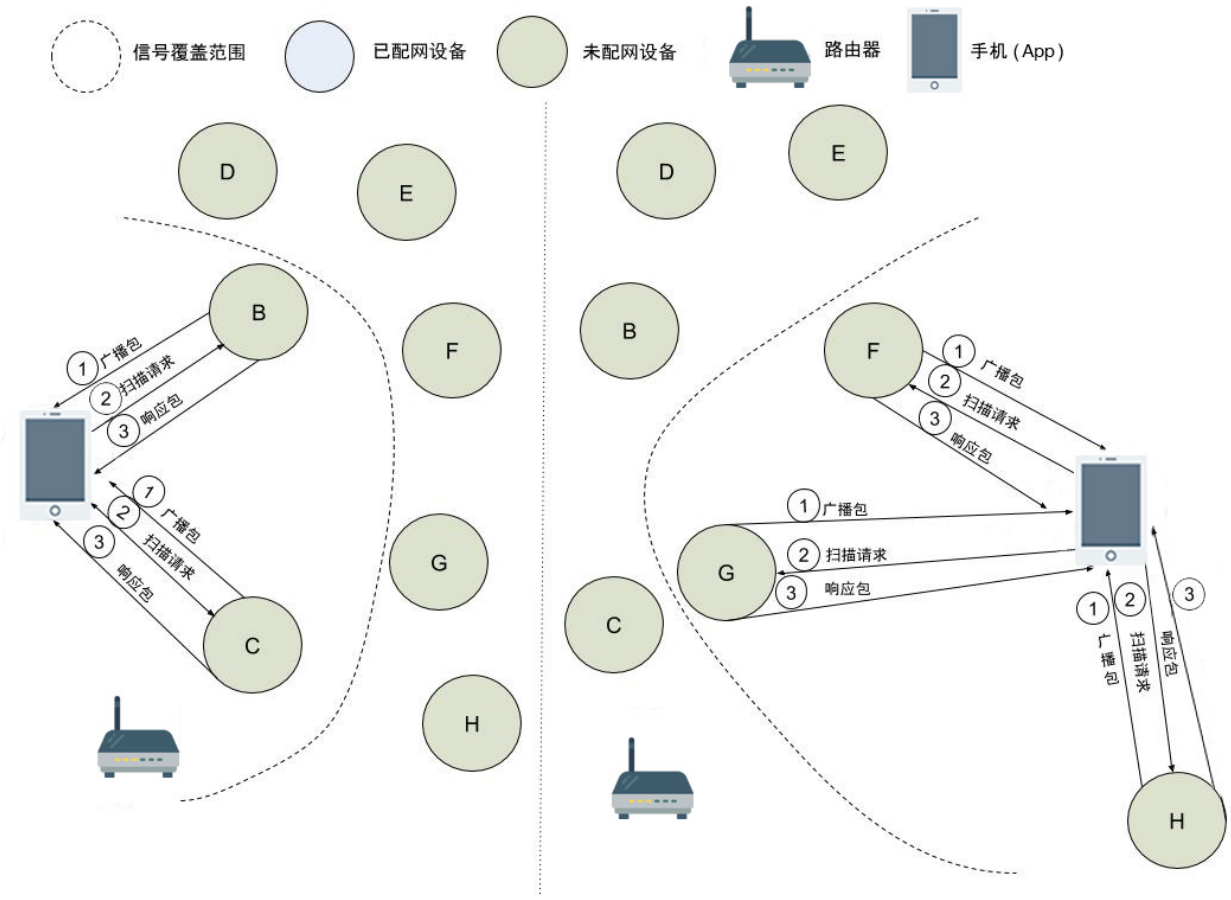
1. 固件大小：固件的大小将增大 500 KB 左右，因此需调整 flash 分区表，保证存放固件的分区大于 1 MB；
2. 内存使用：内存将多占用 30 KB，若释放此内存，需要重启才能再次使用蓝牙。

设备发现

设备通过 BLE 发送特定的蓝牙广播包，App 搜索到此特定的广播，根据信号强度进行筛选，生成白名单，避免将附近不属于自己的设备添加到自己的网络中。其过程如下图所示：

蓝牙广播包分为广播包 (Advertising Data) 和响应包 (Scan Response) 两种类型。广播包 (Advertising Data) 用于存放具体产品的自定义的数据，响应包（Scan Response）用于存放配网信息。


App
ESP32
路由器
设备发现
广播包
扫描请求
响应包
设备筛选
密钥协商
创建蓝牙连接
发送配网请求
RSA 的公钥
选择
[参数]
检查公钥
随机生成 128 bit 数据作为密钥
使用 RSA 公钥对密钥进行加密
RSA 加密后的密钥
RSA 私钥进行解密，获得密钥
数据通信
使用 AES 加密后的配网信息
使用 AES 解密后，获得配网信息
校验配置
创建连接
连接状态
校验的结果


信号覆盖范围
已配网设备
未配网设备
路由器
手机（App）
D
E
B
F
G
C
H
1 广播包
2 扫描请求
3 响应包

- **广播包（Advertising Data）**
    1. 最大长度为 31 bytes；
    2. 数据格式必须满足 蓝牙广播包的标准。
- **响应包（Scan Response）**
    1. 设备名称占用 10 bytes，
    2. 厂家信息占用 14 bytes，具体内容如下：

字段	长度	描述
company id	2 bytes	Bluetooth SIG 分配给 SIG 成员 公司的唯一标识符
OUI	2 bytes	Mconfig Blufi 的标识码用于广播包过滤，数据为：0x4d, 0x44, 0x46，即：” MDF”
version	2 bits	当前的版本号
whitelist	1 bit	是否使能白名单过滤
security	1 bit	是否验证白名单中设备的合法性
reserved	4 bits	保留以备后期扩展
sta mac	6 bytes	设备 sta 的 MAC 地址
tid	2 bytes	设备的类型

密钥协商

1. App 通过 BLE 连接信号最好的设备（A），并向其发送配网请求；
2. 设备收到配网请求后，返回 RSA 的公钥给 App；
3. App 校验 RSA 的公钥的合法性；
4. App 随机生成一个 128 bit 的密钥，并用 RSA 的公钥对进行加密，发送给设备；
5. 设备使用 RSA 的私钥对接收到的数据进行解密，获取密钥，之后 App 与设备之间数据均以此密钥进行 AES 加密。

数据通信

App 将配置信息与设备列表合成一个数据包，并以 BluFi 的自定义字段进行传输，数据包采用 TLV 的格式，数据包中数据的类型及描述如下：

类型	含义	长度 (bytes)	解释
	路由器配置		
1	BLUFI_DATA_ROUTER_SSID	32	SSID of the rou
2	BLUFI_DATA_ROUTER_PASSWORD	64	Router passwor
3	BLUFI_DATA_ROUTER_BSSID	6	BSSID is equal

类型	含义	长度 (bytes)	解释
4	BLUFI_DATA_MESH_ID	6	Mesh network I
5	BLUFI_DATA_MESH_PASSWORD	64	Password for se
6	BLUFI_DATA_MESH_TYPE	1	Only MESH_II
	MESH 网络配置		
16	BLUFI_DATA_VOTE_PERCENTAGE	1	Vote percentage
17	BLUFI_DATA_VOTE_MAX_COUNT	1	Max multiple vo
18	BLUFI_DATA_BACKOFF_RSSI	1	RSSI threshold
19	BLUFI_DATA_SCAN_MIN_COUNT	1	The minimum n
20	BLUFI_DATA_SCAN_FAIL_COUNT	1	Max fails (60 by
21	BLUFI_DATA_MONITOR_IE_COUNT	1	Allowed number
22	BLUFI_DATA_ROOT_HEALING_MS	2	Time lag betwee
23	BLUFI_DATA_ROOT_CONFLICTS_ENABLE	1	Allow more than
24	BLUFI_DATA_FIX_ROOT_ENABLE	1	Enable a device
25	BLUFI_DATA_CAPACITY_NUM	2	Network capacit
26	BLUFI_DATA_MAX_LAYER	1	Max number of
27	BLUFI_DATA_MAX_CONNECTION	1	Max number of
28	BLUFI_DATA_ASSOC_EXPIRE_MS	2	Period of time a
29	BLUFI_DATA_BEACON_INTERVAL_MS	2	Mesh softAP be
30	BLUFI_DATA_PASSIVE_SCAN_MS	2	Mesh station pa
31	BLUFI_DATA_MONITOR_DURATION_MS	2	Period (ms) for
32	BLUFI_DATA_CNX_RSSI	1	RSSI threshold
33	BLUFI_DATA_SELECT_RSSI	1	RSSI threshold
34	BLUFI_DATA_SWITCH_RSSI	1	RSSI threshold
35	BLUFI_DATA_XON_QSIZE	1	Number of MES
36	BLUFI_DATA_RETRANSMIT_ENABL	1	Enable a source
37	BLUFI_DATA_DROP_ENABLE	1	If a root is chan
	白名单配置		
64	BLUFI_DATA_WHITELIST	6 * N	Device address
		32 * N	Verify the validi

校验配置

设备端获取到 AP 的信息后，尝试连接路由器，以校验配置信息是否正确，并将连接路由器的状态和校验的结果返回给 App，其校验结果如下：

类型	含义	备注
0	ESP_BLUFI_STA_CONN_SUCCESS	连接路由器成功
1	ESP_BLUFI_STA_CONN_FAIL	连接路由器失败
16	BLUFI_STA_PASSWORD_ERR	密码配置错误
17	BLUFI_STA_AP_FOUND_ERR	路由器未找到
18	BLUFI_STA_TOOMANY_ERR	路由器已经达到最大的连接数
19	BLUFI_STA_CONFIG_ERR	参数配置错误

4.3.6 Mconfig-Chain

Mconfig-Chain 是基于 ESP-NOW （一种由 Espressif 定义的无连接 Wi-Fi 通信协议）的设备间配网协议。

当前 Wi-Fi 网络配置主要有三种方式：BLE 配网、智能配网（sniffer）和 softAP 配网，均是为单个设备配网设计的，其并不适合 ESP-WIFI-MESH 网络这种多设备同时配网的场景。Mconfig-Chain 是专为设计 ESP-WIFI-MESH 网络的配网方式，其配网过程是链式的、可传递的，所有已配网的设备均可以为其他设备配网，实现大范围高效配网。

Mconfig-Chain 将设备分为 Master（已配网的设备）和 Slave （等待配网的设备）两种类型，配网过程分为设备发现、密钥协商和数据通信。

设备发现

1. **Master 在 Wi-Fi beacon 中的 Vendor IE 加入链式配网的标识，等待 Slave 的配网请求；**

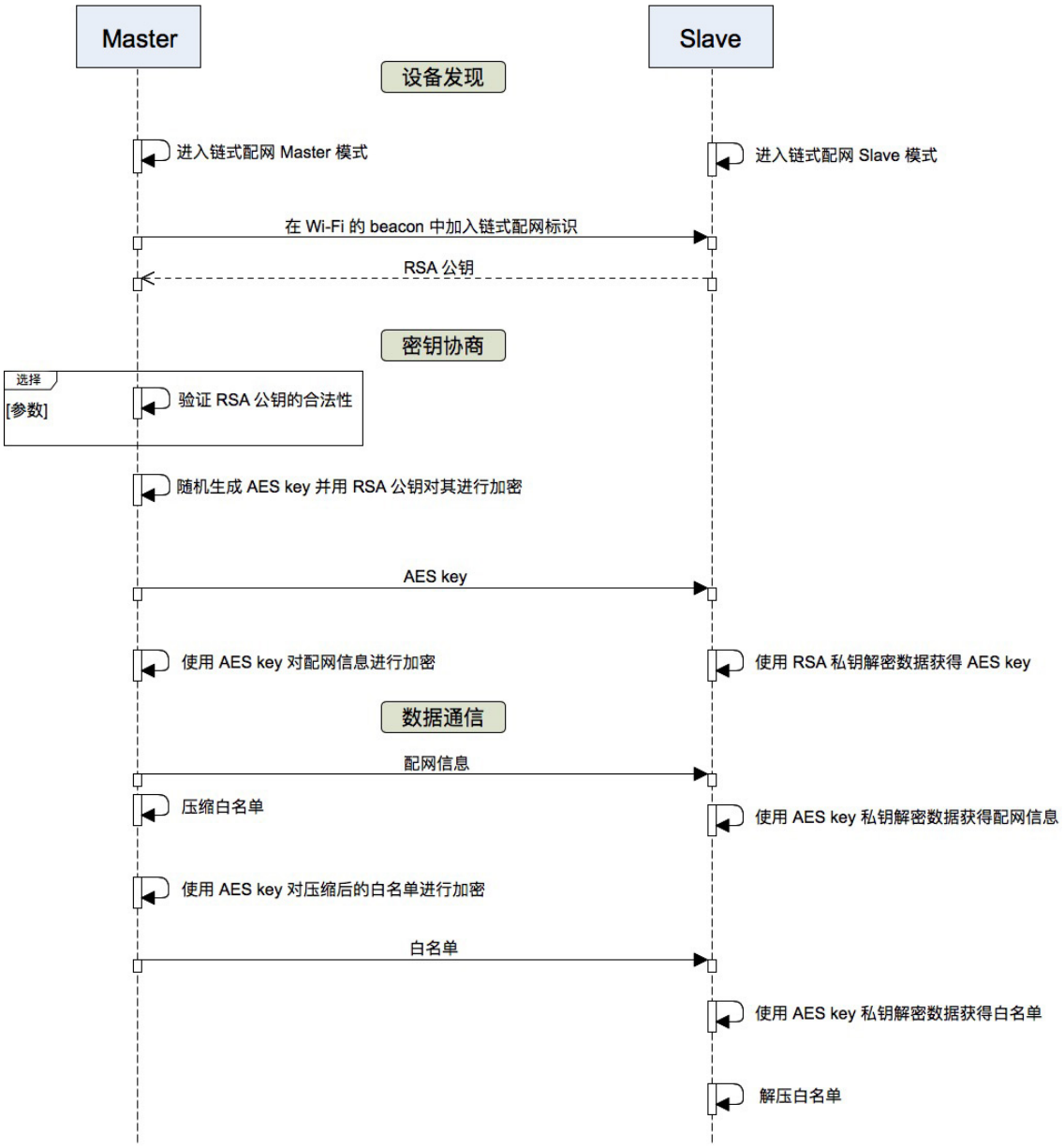
    - Vendor IE 标识的格式如下：

类型	数据
Element ID	0xDD
Length	0X04
OUI	0X18, 0XFE, 0X34
Type	0X0F

- Master 需要配置窗口期，仅在其窗口期内接收 Slave 的请求；
- 通过 Wi-Fi beacon 发送链式配网的标识，如果设备仅处于 STA 模式将无法启用 Master；

2. **Slave 开启 Wi-Fi 的 sniffe 功能，不断切换信道监听 Wi-Fi 广播包，查找链式配网的标识，如若发现 Master 则停**

    - Slave 工作时会切换信道，在使用之前应停用 ESP-WIFI-MESH 的自组网。


Master
Slave
设备发现
进入链式配网 Master 模式
进入链式配网 Slave 模式
在 Wi-Fi 的 beacon 中加入链式配网标识
RSA 公钥
密钥协商
选择
[参数]
验证 RSA 公钥的合法性
随机生成 AES key 并用 RSA 公钥对其进行加密
AES key
使用 AES key 对配网信息进行加密
使用 RSA 私钥解密数据获得 AES key
数据通信
配网信息
压缩白名单
使用 AES key 私钥解密数据获得配网信息
使用 AES key 对压缩后的白名单进行加密
白名单
使用 AES key 私钥解密数据获得白名单
解压白名单

密钥协商

1. Master 收到 Slave 配网请求，校验 Slave 是否在配网白名单中，若使能设备身份认证，则需将设备收到的 RSA 公钥进行 MD5 运算与配网白名单比较校验其合法性；
2. Master 删除 Wi-Fi beacon 中的 Vendor IE 链式配网的标识；
3. Master 随机生成一个 128 bit 的数据，作为与 Slave 通信的密钥，并用接收到的 RSA 公钥对密钥进行加密，通过 ESP-NOW 发送给 Slave；
4. Slave 收到 Master 的 Response 后，使用 RSA 私钥对其进行解密获得通信的密钥。

数据通信

1. Master 使用密钥通过 AES 算法对配网信息和白名单进行加密，通过 ESP-NOW 发送给 Slave；
2. Slave 使用密钥通过 AES 算法对收到的数据进行解密，完成配网，并从 Slave 模式切换到 Master 模式。

注解： ESP-NOW 会在数据链路层对数据进行加密，相互配网设备加密的密钥必须相同，其密钥在产品生产时写入 flash 中或直接存储在固件中。

4.4 Mlink

[English]

4.4.1 1. 准备工作

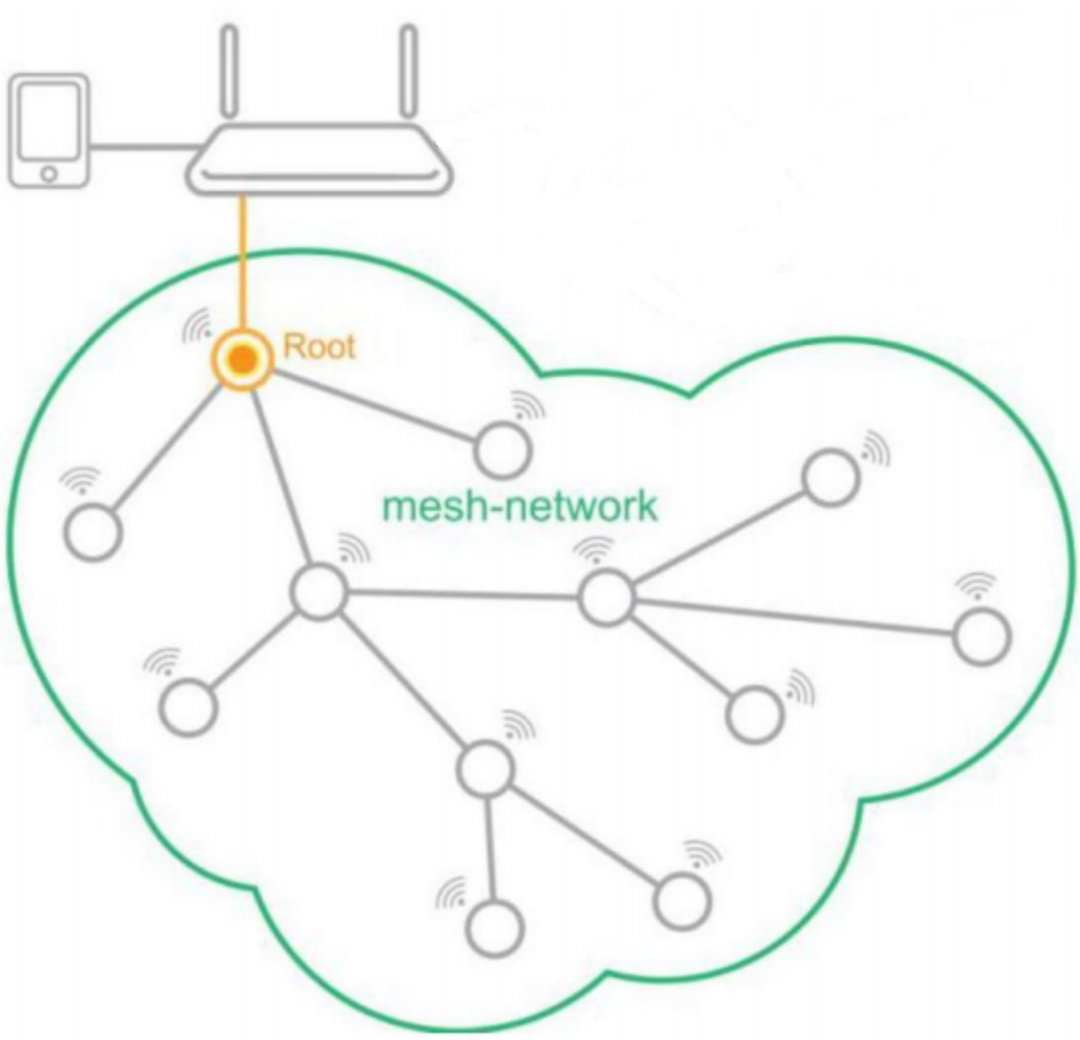
ESP-Mesh app 与 ESP-MDF 设备之间进行局域网通信的前提是：设备已经组网成功，且手机和 Mesh 网络处于同一个局域网中。

ESP-MDF 设备在配网成功后会自动进入 组网阶段, 在 ESP-Mesh app 的设备页面中能够扫描到所以已配网的设备即可表示组网已经成功。

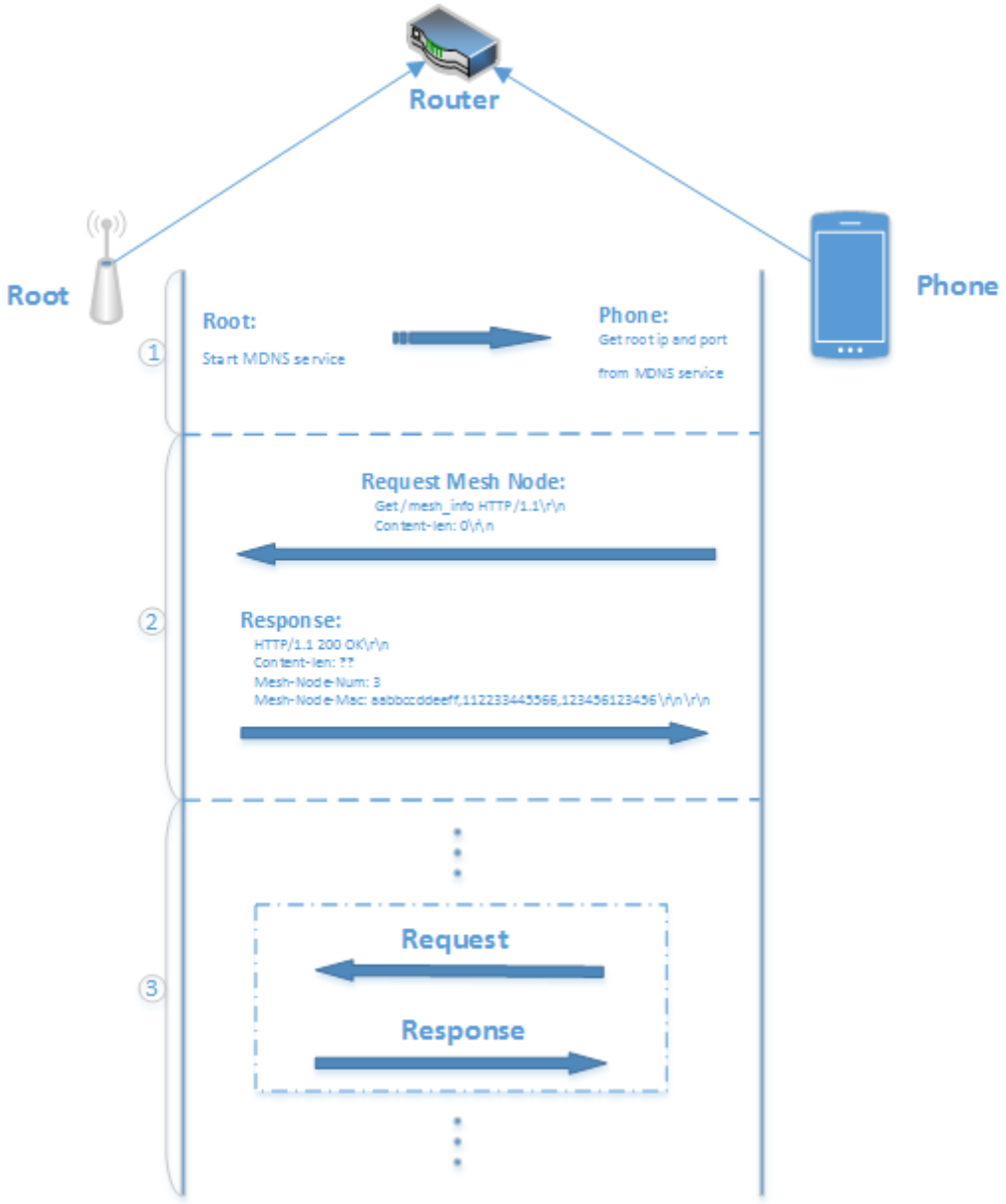
4.4.2 2. 通信流程

根节点是一个 Mesh 网络与外部通信的唯一出口，app 想要控制 Mesh 网络内设备，首先需要在该网络中找到根节点，之后通过根节点获取该 Mesh 网络内的设备列表，最后才能与该 Mesh 网络中任意设备进行通信。整个过程主要分为以下三步：

1. App 获取根节点的 IP 地址，局域网通信端口和 mac 地址
2. App 询问根节点获得 Mesh 网络设备列表


Root
mesh-network

3. App 与 Mesh 网络中任意设备进行通信



4.4.3 3. 通信协议

本章节从通信流程角度，分别介绍这三个步骤中所涉及的通信协议格式。其中，app 获取 Mesh 网络设备列表，app 与 Mesh 网络中任意设备的通信采用标准的 HTTP 或 HTTPS 通信协议。此外，协议还包括以下两部分：

1. 设备状态通知：为了方便用户通过 app 查看设备的实时状态，在现有协议中增加了设备通知功能，即在设备状态发生变化时主动发 UDP 广播通知 app，之后由 app 来获取设备详细状态的过程。
2. 本地触发控制：实现局域网内设备间的触发控制

3.1. App 获取根节点的 IP 地址，局域网通信端口和 MAC 地址

现阶段，根节点会同时启动 mDNS 服务和接收 UDP 广播两种方式用于设备发现。设备发现表示 app 获取根节点的 IP 地址，通信端口和 MAC 地址的过程

1. mDNS 设备发现服务

当 app 查询到相关的 mDNS 服务时，便获取了根节点的 *IP*，再通过该服务中 *port* 和 *txt* 字段可知根节点的 *局域网通信端口和 mac 地址*

mDNS Service Info:

```
hostname: "esp32_mesh"
instance_name: "mesh"
service_type: "_mesh-https"
proto: "_tcp"
port: 80
txt:
    key: "mac"
    value: "112233445566"
```

2. 接收 UDP 广播

在设备发现阶段，app 会主动发送 UDP 广播包，并根据根节点的回复获取根节点的信息.

Request:

```
"Are You Espressif IOT Smart Device?"
```

Response:

```
"ESP32 Mesh 112233445566 http 80"
```

注解:

- 112233445566 为根节点的 MAC 地址
- 80 是 http 服务器端口
- 此外，app 通过 UDP 回复包获取根节点的 IP 地址

---

3. 设备主动通知

当 ESP-MDF 设备状态（打开或关闭）、网络连接关系（连接或断开）、路由表（route table）信息等发生变化时，根节点会主动发送 UDP 广播，通知 app 来获取设备的最新状态

UDP Broadcast:

mac=112233445566 flag=1234 type=***

- `mac` 状态发生变化的设备的 mac 地址
- `flag` 为一个随机整数值，用于区分不同时刻的通知
- `type` 状态变化的类型，包含以下几种：
- `status` 设备属性状态发生变化
- `http` 设备拓扑结构发生变化
- `sniffer` 设备使用监听模式监听到了设备

3.2. App 获取设备列表

Request::

```
GET /mesh_info HTTP/1.1
Host: 192.168.1.1:80
```

Response::

```
HTTP/1.1 200 OK
Content-Length: ??
Mesh-Node-Mac: aabbccddeeff,112233445566
Host: 192.168.1.1:80
```

注解:

- `/mesh_info` app 获取设备列表的命令，通过 http 的 URL 字段实现
- `Mesh-Node-Mac` 节点的 Station Mac 地址，以逗号分隔
- `Host` HTTP/1.1 协议必须携带的字段，表示控制端 IP 地址和通信端口

3.3. App 与 ESP-MDF 设备通信格式说明

1. App 请求格式

根节点只解析 http 头部信息, 将 http 的消息体通过 ESP-WIFI-MESH 透传给目标设备

Request::

```
POST /device_request HTTP/1.1
Content-Length: ??
Content-Type: application/json
```

(下页继续)

(续上页)

```
Root-Response::??
Mesh-Node-Mac: aabbccddeeff,112233445566
Mesh-Node-Group: 01000000000,02000000000
Host: 192.168.1.1:80

**content_json**
```

1. `/device_request` app 对设备的控制操作，包括状态的设置和获取，通过 http 请求的 URL 字段实现
2. `Content-Length` http 请求的 body 数据长度
3. `Content-Type` http 请求的 body 数据类型，现阶段采用 `application/json` 格式
4. `Root-Response` 是否只需要根节点回复。如果只需要根节点回复, 则只由根节点回复命令是否收到. 通常用于控制设备时, 去除上报的数据包, 以达到更好的控制效果.
5. `Mesh-Node-Mac` 命令转发的目标设备的 MAC 地址。*ffffffffffff* 则表示控制所有设备
6. `Mesh-Node-Group` 命令转发的目标设备所在的组。
7. `**content_json**` http 请求的消息体，表示章节 `3.4. 消息体的数据`中的 `Response` 部分
2. 设备回复的格式

根节点 ESP-WIFI-MESH 收到设备回复信息后, 生成 http 的头部信息, 将回复信息放到 http 的消息体. 转发给 app

Response:

```
HTTP/1.1 200 OK
Content-Length: ??
Content-Type: application/json
Mesh-Node-Mac: 30aea4062ca0
Host: 192.168.1.1:80
\r\n
**content_json**
```

1. `Content-Length` 消息体数据长度
2. `Content-Type` 消息体数据类型
3. `Mesh-Node-Mac` 设备自身的 mac 地址
4. `**content_json**` http 响应的消息体，表示章节 `3.4. 消息体的数据`中的 `Response` 部分

3.4. 消息体的数据

1. Acquire device information: get_device_info

Request:

```
{
    "request": "get_device_info"
}
```

- `request` is field defining the operation on the device, followed by specific commands of operation.

Response:

```
{
    "tid": "1",
    "name": "light_c800",
    "mesh_id": "94d9b3808c81",
    "version": "1.1.1",
    "idf_version": "v3.2.2-46-g878d70d9e",
    "mdf_version": "e0fb50c-dirty",
    "mlink_version": 2,
    "mlink_trigger": 0,
    "rssi": -28,
    "layer": 1,
    "group": ["010000000000"],
    "characteristics": [{
        "cid": 0,
        "name": "on",
        "format": "int",
        "perms": 7,
        "value": 1,
        "min": 0,
        "max": 3,
        "step": 1
    }, {
        "cid": 1,
        "name": "hue",
        "format": "int",
        "perms": 7,
        "value": 360,
        "min": 0,
        "max": 360,
        "step": 1
    }, {
        "cid": 2,
```

(下页继续)

(续上页)


```
        "name": "saturation",
        "format": "int",
        "perms": 7,
        "value": 0,
        "min": 0,
        "max": 100,
        "step": 1
    }, {
        "cid": 3,
        "name": "value",
        "format": "int",
        "perms": 7,
        "value": 100,
        "min": 0,
        "max": 100,
        "step": 1
    }, {
        "cid": 4,
        "name": "color_temperature",
        "format": "int",
        "perms": 7,
        "value": 0,
        "min": 0,
        "max": 100,
        "step": 1
    }, {
        "cid": 5,
        "name": "brightness",
        "format": "int",
        "perms": 7,
        "value": 30,
        "min": 0,
        "max": 100,
        "step": 1
    }, {
        "cid": 6,
        "name": "mode",
        "format": "int",
        "perms": 3,
        "value": 2,
```


(下页继续)

(续上页)

```
        "min": 1,
        "max": 3,
        "step": 1
    }],
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

- `tid` 设备的 type ID，用于区分灯，插座，空调等不同类型的 ESP-MDF 设备. 1~9: 灯, 10~19: 按键, 20~29: 传感器
- `name` 设备名称
- `version` 设备固件版本
- `idf_version` esp-idf 的版本
- `mdf_version` esp-mdf 的版本
- `mlink_version` mlink 通信协议的版本
- `rssi` 与父节点之间的信号强度
- `layer` 设备所属的层级
- `group` 设备所在的组
- **`characteristics` 设备的属性信息**
  - `cid` 设备属性身份（characteristic ID），用于区分亮度，明暗，开关等不同的的设备属性
  - `name` 属性名称
  - `format` 数据类型，支持 `int`，`double`，`string`，`json` 四种数据类型
  - `value` 属性值
  - `min` 属性值的最小值或支持的字符串的最小长度
  - `max` 属性值的最大值或支持的字符串的最大长度
  - **`step` 属性值的最小步长**
    - 当 `format` 为 `int` 或 `double` 数据类型时，`min`，`max` 和 `step` 分别表示属性值的最小值，最大值和最小变化值
    - 当 `format` 为 `string` 或 `json` 数据类型时，`min` 和 `max` 分别表示支持的字符串的最小长度和最大长度，无关键字 `step`
- **`perms` 属性的操作权限，以二进制整数解析，第一位表示** 读权限，**第二位表示** 写权限，**第三位表示** 执行权限，**0 表示**
  - 若参数没有读权限，则无法获得对应的值

  - 若参数没有写权限，则无法修改对应的
  - 若参数没有执行权限，则无法执行设置对应的触发事件

- `status_code` 请求命令的错误码，`0` 表示正常，`-1` 表示错误
- `status_msg` 错误码对应的消息

2. 获取设备状态：get_status

Request:

```
{
    "request": "get_status",
    "cids": [
        0,
        1,
        2
    ]
}
```

- `cids` 设备属性字段，后接请求的 CID 值列表

Response:

```
{
    "characteristics": [
        {
            "cid": 0,
            "value": 0
        },
        {
            "cid": 1,
            "value": 0
        },
        {
            "cid": 2,
            "value": 100
        }
    ],
    "status_code": 0
}
```

3. 修改设备状态: set_status

Request:

```
{
    "request": "set_status",
    "characteristics": [
        {
            "cid": 0,
            "value": 0
        },
        {
            "cid": 1,
            "value": 0
        },
        {
            "cid": 2,
            "value": 100
        }
    ]
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

4. 进入配网模式: config_network

Request:

```
{
    "request": "config_network"
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

5. 重启设备: reboot

Request:

```
  {
      "request": "reboot",
      "delay": 50
  }

``delay`` 命令延时执行时间，该字段非必需，若不设，使用设备端默认延时 ``3s``
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

6. 恢复出厂设置: reset

Request:

```
{
    "request": "reset",
    "delay": 50
}
```

- `delay` 命令延时执行时间，该字段非必需，若不设，使用设备端默认延时 `3s`。

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

7. 添加新设备: add_device

Request:

```
{
    "request": "add_device",
    "whitelist": ["aabbccddeeff", "112233445566"],
    "timeout": 30000,
    "rssi": -65
}
```

- `whitelist` 新设备的 mac 地址列表

- `timeout` 允许设备添加的窗口时间
- `rssi` 允许设备添加的最小信号强度

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

8. 修改设备名称: rename_device

Request:

```
{
    "request": "rename_device",
    "name": "light_c800_11"
}
```

- `name` 修改后的设备名称, 总长度必需小于 32 个字节

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

9. 修改设备地址: set_position

Request:

```
{
    "request": "set_position",
    "position": "1F-A-001"
}
```

- `position` 修改后的设备地址, 总长度必需小于 32 个字节

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

10. 获取升级状态: get_ota_progress

Request:

```
{
    "request": "get_ota_progress",
}
```

Response:

```
{
    "firmware_name": "v1.2.0",
    "total_size": 1422768,
    "written_size": 1190912,
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

- `firmware_name` 固件的版本号
- `total_size` 固件的大小
- `written_size` 已经烧录的固件长度

11. 版本回退: set_ota_fallback

Request:

```
{
    "request": "set_ota_fallback",
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

12. 获取 ESP-WIFI-MESH 的配置: get_mesh_config

Request:

```
{
    "request": "get_mesh_config",
}
```

Response:

```
{
    "id": "94d9b3808c81",
    "max_layer": 16,
    "max_connections": 6,
    "layer": 2,
    "parent_mac": "30aea480659c",
    "type": 2,
    "prarent_rssi": -22,
    "router_rssi": -30,
    "beacon_interval": 1000,
    "assoc_expire": 30,
    "capacity_num": 512,
    "free_heap": 68324,
    "running_time": 1474945,
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

- `id` ESP-WIFI-MESH 的网络 id
- `max_layer` 网络最大层级
- `max_connections` 每一个设备的子节点连接数
- `layer` 当前所处的层级
- `parent_mac` 父节点的 mac 地址
- `type` 节点的类型, 1: 根节点, 2: 中间节点, 3: 叶子节点
- `prarent_rssi` 与父节点之间的信号强度
- `router_rssi` 与路由器之间的信号强度
- `beacon_interval` `beacon` 包的发送间隔
- `assoc_expire` 被动断开检测的超时时间
- `capacity_num` 网络的最大设备数
- `free_heap` 当前剩余的可用内存
- `running_time` 运行的时间

13. 修改 ESP-WIFI-MESH 的配置: set_mesh_config

Request:

```
{
    "request": "set_mesh_config",
    "beacon_interval": 1000
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

14. 添加组: set_group

Request:

```
{
    "request": "set_group",
    "group": ["cec0c9fabce4"]
}
```

- group 组的 id, 只能由 6 Byte 的十六进制数据组成

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

15. 获取组: get_group

Request:

```
{
    "request": "get_group"
}
```

- group 组的 id, 只能由 6 Byte 的十六进制数据组成

Response:

```
{
    "group": ["cec0c9fabce4"],
    "status_msg": "MDF_OK",
```

(下页继续)

(续上页)

```
    "status_code": 0
}
```

16. 删除组: remove_group

Request:

```
{
    "request": "remove_group",
    "group": ["cec0c9fabce4"]
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

17. 修改 sniffer 配置: set_sniffer_config

Request:

```
{
    "request": "set_sniffer_config",
    "type": 3,
    "notice_threshold": 50,
    "esp_module_filter": 1,
    "ble_scan_interval": 1000,
    "ble_scan_window": 50
}
```

- type 扫描到的无线数据包的类型, 0: 不监听, 1: Wi-Fi 广播包, 2: BLE 广播包, 3: Wi-Fi + BLE 广播包
- notice_threshold 当设备缓冲区达到此占比是会发送 UDP 通知
- esp_module_filter 是否过滤扫描到的乐鑫芯片
- ble_scan_interval BLE 扫描两次扫描之间的间隔时间
- ble_scan_window BLE 每一次扫描的时间

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

18. 获取 sniffer 配置: get_sniffer_config

Request:

```
{
    "request": "get_sniffer_config",
}
```

Response:

```
{
    "type": 3,
    "notice_threshold": 50,
    "esp_module_filter": 1,
    "ble_scan_interval": 1000,
    "ble_scan_window": 50,
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

19. 获取 sniffer 到的数据: get_sniffer_info

Request:

```
{
    "request": "get_sniffer_info",
}
```

Response:

```
31 02 07 02 75 78 f9 b9 38 a4 05 03 1c 00 00 00
02 01 b9 1e 06 4c 00 07 19 01 02 20 22 f7 0f 03
00 00 13 ca 1b 7b dc 5a 16 6b 7c 8d 9a 18 dd 36
20 39 15 01 07 02 b8 27 eb 0c c1 20 05 03 bf 01
00 00 02 01 c0 02 05 01 15 01 07 02 3c 71 bf 6a
12 fc 05 03 79 04 00 00 02 01 b1 02 05 01 1a 02
07 02 a4 5e 60 bd 42 64 05 03 10 00 00 00 02 01
b1 07 06 4c 00 10 02 0b 00 31 02 07 02 39 46 16
```

(下页继续)

(续上页)

```
d5 f3 be 05 03 1f 04 00 00 02 01 b6 1e 06 06 00
01 09 20 02 c3 fd ec b2 c7 9a 61 14 d3 e0 fc 84
41 86 b4 af 8a 46 12 90 04 b8 26 1d 02 07 02 7a
32 38 bf 29 dd 05 03 0f 04 00 00 02 01 bc 0a 06
4c 00 10 05 0b 1c d0 48 b1
```

- 数据格式为: [size (1 Byte) | type (1 Byte) | data (n Byte)] ···. [size | type | data]
- size: *size = sizeof(type) + sizeof(data)* 即: size = 1 + n
- type: 数据包的类型. 1: Wi-Fi 数据包, 2: BLE 数据包
- **data: < len (1 Byte) | data_type (1 Byte) | data (n Byte) > ···< len | data_type | data >**
  - len: len = sizeof(data_type) + sizeof(data) 即: len = 1 + n
  - type: 各个字段的类型: 1: 信号强度, 2: 设备的地址, 3: 多长时间之前被扫描到, 单位: ms, 4: 名称, 5: 信道, 6: 厂商 ID

20. 获取已有触发事件: get_event

Request:

```
{
    "request": "get_event"
}
```

Response:

```
{
    "events": {
        "name": "on",
        "trigger_content": {
            "request": "contorl"
        },
        "trigger_cid": 2,
        "trigger_compare": {
            ">": 1,
            "~": 10
        },
        "execute_mac": [
            "30aea4064060"
        ]
    },
```

(下页继续)

(续上页)

```
    "status_code": 0
}
```

- events 触发控制事件 * name 触发事件的名称，是触发事件的唯一标识 * trigger_content 触发事件的内容
  - **request 触发事件的类型**
    * sync 同步，同步两个设备，保持状态一致
    * linkage 关联，一般的关联控制，当触发条件满足时发送触发命令
    * delay 延时，表示在触发条件满足时，延迟给定的时间（单位秒）后发送触发命令。
  - trigger_cid 关联设备的属性值，是触发事件的检测对象
  - **trigger_compare 触发事件的判断条件，用于比较关联设备的实际值与设置的触发值**
    * > 大于
    * < 小于
    * == 等于
    * != 不等于
    * ~ 单位时间（秒）内的变化量
    * / 单位时间（秒）上升到某个值
    * \\ 单位时间（秒）下降到某个值
  - execute_mac 触发事件的目标地址，及执行指令的执行设备地址

21. 修改触发事件：set_event

触发控制的协议主要在于对触发条件的理解和转换，即如何将用户设置的条件转成计算机可理解的条件判断, 如果设备端无此事件则添加该事件，若存在则修改事件

Request:

```
{
    "request": "set_event",
    "events": {
        "name": "on",
        "trigger_cid": 0,
        "trigger_content": {
            "request": "linkage"
        },
        "trigger_compare": {
            "==": 0,
```

(下页继续)

(续上页)

```
            "~": 1
        },
        "execute_mac": [
            "30aea457e200",
            "30aea457dfe0"
        ],
        "execute_content": {
            "request": "set_status",
            "characteristics": [
                {
                    "cid": 0,
                    "value": 1
                }
            ]
        }
    }
}
```

- `set_event` 设置设备触发控制数据的命令
- `events` 触发控制事件 * `name` 触发事件的名称，是触发事件的唯一标识 * `trigger_cid` 关联设备的属性值，是触发事件的检测对象 * `trigger_content` 触发事件的内容，其中触发事件类型包括以下三种：*sync*、*linkage* 和 *delay* * `trigger_compare` 触发事件的判断条件，用于比较关联设备的实际值与设置的触发值，包括以下七种：>、<、==、!=、~、/ 和 \ * `execute_mac` 触发事件的目标地址，即执行指令的设备地址 * `execute_content` 执行指令内容，当触发条件满足时，发送该命令到 `execute_mac`

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

22. 删除触发事件：remove_event

Request:

```
{
    "request": "remove_event",
    "events": [
        {
            "name": "on"
```

(下页继续)

(续上页)

```
        },
        {
            "name": "off"
        }
    ]
}
```

Response:

```
{
    "status_msg": "MDF_OK",
    "status_code": 0
}
```

4.5 Mupgrade

[English]

Mupgrade (Mesh Upgrade) 是 ESP-WIFI-MESH OTA 升级的一种方案，目的是通过断点续传、数据压缩、版本回退和固件检查等机制实现 ESP-WIFI-MESH 设备高效的升级。

Mupgrade 先将升级的固件下载到根节点，再由根节点将固件分包后，同时给多个设备进行批量升级。

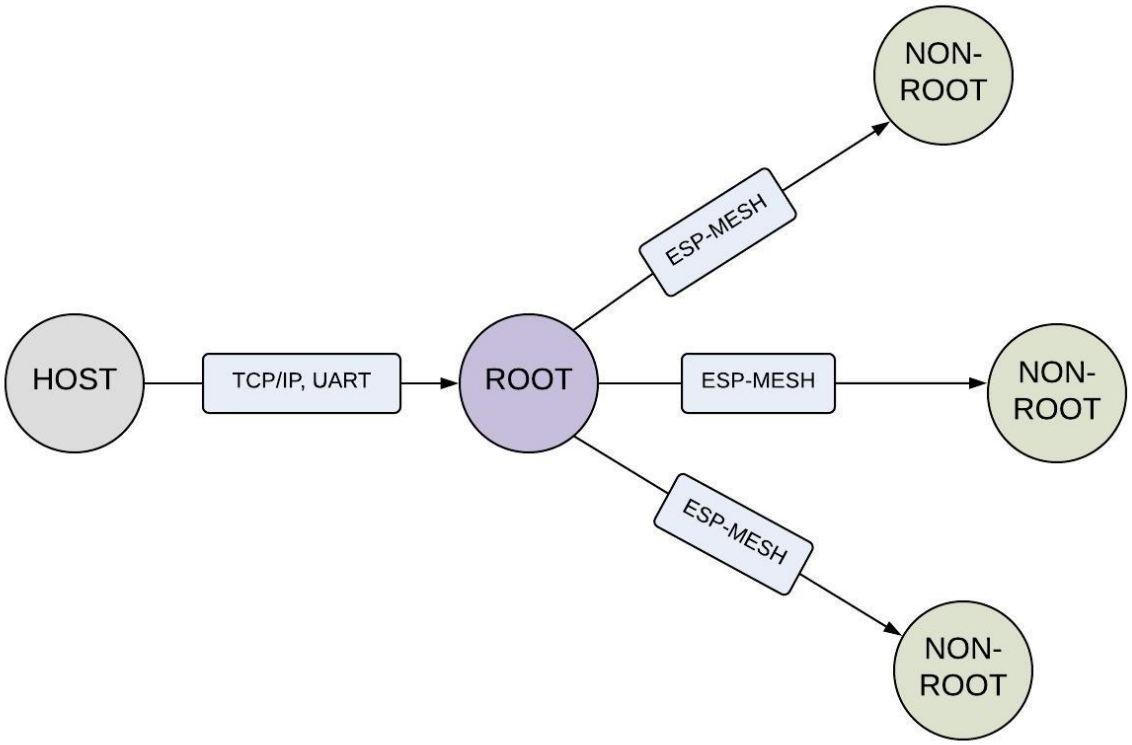
4.5.1 功能

- **断点续传**：根节点将固件以固定的大小进行分包，待升级设备会记录每包固件的升级情况并写入 flash 中，中断升级后，再次升级时只请求未接收到的数据；
- **数据压缩**：使用 miniz 对每片固件进行压缩处理，减少数据包的大小，提高传输速率；
- **组播发送**：多个设备同时进行批量升级时，设备在收到升级数据包后自行拷贝一份，再传给下一节点，减少数据传输；
- **固件检查**：生成的固件中包含 Mupgrde 标识和 CRC 校验码，避免升级不带此功能的版本、固件发送错误和固件不完整等问题；
- **版本回退**：设备可以通过一定方式（GPIO 触发或连续断电重启多次）回退到上一个版本。

4.5.2 流程

步骤一：固件下载

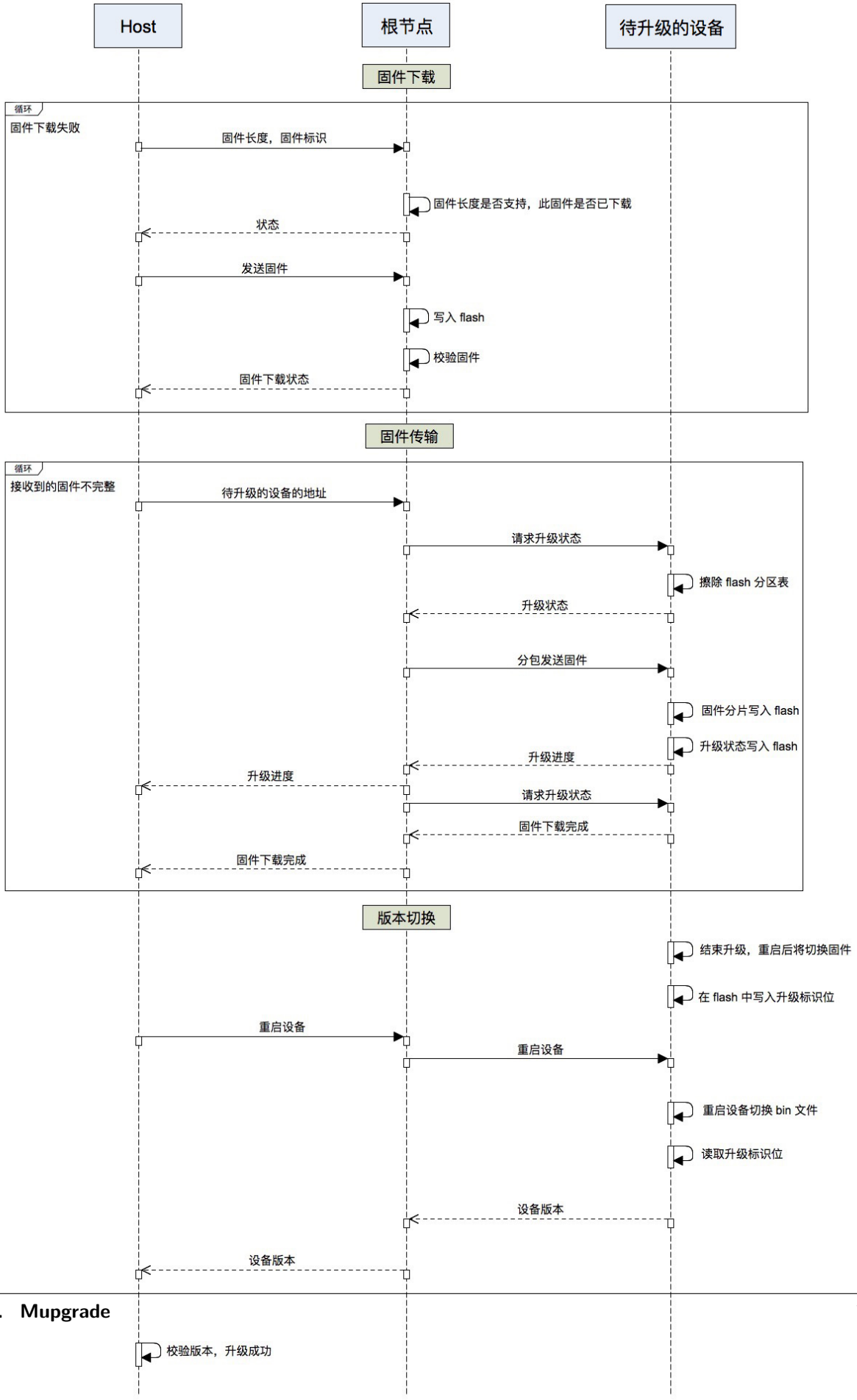
1. HOST 通过 UART 或 Wi-Fi 等方式与根节点建立通信；



2. HOST 将固件的长度和标识等信息传给根节点；
3. 根节点通过固件的长度和标识判断是否支持此固件升级，擦除 flash 分区并返回状态给 HOST；
4. **HOST 收到返回状态后，做出相应的处理；**
    - 已经下载：跳过固件发送的步骤 5，进入步骤 6；
    - 长度错误：检查固件是否发送错误，大小是否超过分区限制；
    - 检查成功：继续后续的操作。
5. HOST 发送固件给根节点，根节点收到后直接写入 flash，当接收到的固件长度与设置长度相等时，会主动校验固件，并返回固件烧录情况；
6. HOST 将待升级的设备列表发送给根节点。

注解:

1. 分区表中必须包含：ota_data, ota_0, ota_1；
2. 固件大小不能超过 ota_0 与 ota_1 分区的大小；
3. 根节点在收到数据包后会进行写 flash 操作，此时会挂起其他 task，若此时 HOST 给根节点发包将会造成数据包的丢失。因此若使用 UART 发送固件，请打开流控，或分包发送且等到根节点回 ACK 后再发下一包数据。


Host
根节点
待升级的设备
固件下载
循环
固件下载失败
固件长度，固件标识
固件长度是否支持，此固件是否已下载
状态
发送固件
写入 flash
校验固件
固件下载状态
固件传输
循环
接收到的固件不完整
待升级的设备的地址
请求升级状态
擦除 flash 分区表
升级状态
分包发送固件
固件分片写入 flash
升级状态写入 flash
升级进度
升级进度
请求升级状态
固件下载完成
固件下载完成
版本切换
结束升级，重启后将切换固件
在 flash 中写入升级标识位
重启设备
重启设备
重启设备切换 bin 文件
读取升级标识位
设备版本
设备版本
校验版本，升级成功

步骤二：固件传输

1. 根节点收到设备列表后，检查设备列表是否正确；
2. 根节点向列表中的设备发送升级状态请求；
3. 相关设备收到升级状态请求后，若无升级断点数据则擦除 flash 分区，返回丢包情况；
4. 根节点根据返回的丢包情况将固件分包发送给目标设备；
5. 相关设备按固件包的序号将固件分片写入 flash 中，每升级 10% 上报一次升级状态；
6. 根节点重复步骤 2，直至列表中所有设备均完成固件下载或超过重试次数。

步骤三：版本切换

1. 待升级的设备完成接收固件后会在 flash 的 ota_data 中标识下一次重启后运行的分区；
2. 根节点收到所有相关设备的完成信息后，给所有待升级设备发送重启命令；
3. 相关设备收到重启命令后，重启设备后主动上报当前设备的版本；
4. HOST 校验版本，完成升级。

4.5.3 分区表

单个 ESP32 的 flash 有多个应用程序分区和多种不同类型的数据。分区表可以用来定义 flash 布局。由于 esp-idf 内置分区表存放应用程序的分区仅为 1 MB，无法满足 ESP-WIFI-MESH 应用开发的需求，为了简化您对分区表的配置，提供了以下两种类型的分区给您参考。

1. 无 *factory* 分区:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
ota_0,    app,  ota_0,    0x10000,  1920k
ota_1,    app,  ota_1,    ,         1920k
coredump, data, coredump, ,         64K
reserved, data, 0xfe,     ,         128K
```

2. 有 *factory* 分区:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
factory,  app,  factory,  0x10000,  1280k
ota_0,    app,  ota_0,    ,         1280k
ota_1,    app,  ota_1,    ,         1280k
coredump, data, coredump, ,         64K
reserved, data, 0xfe,     ,         128K
```

注解:

1. 更新分区表前，需要先擦除整个 flash；
2. 应用程序分区（factory、ota_0、ota_1）必须处于与 0x10000（64K）对齐的偏移量；
3. 分区表无法通过 OTA 的方式进行修改；
4. 根节点会使用 ota_0 或 ota_1 作固件缓冲，如无 factory 分区版本回退，版本回退后的固件可能不是自己的版本。

4.5.4 注意事项

若要自定义升级方式，请注意：

1. **勿逐一升级**：若版本间不兼容，会破坏原有网络，容易造成孤立节点，增加升级困难；
2. **勿整包传输**：ESP-WIFI-MESH 属于多跳网络，只能保证点对点，无法保证端对端的可靠性。若整个固件一次性传输，当设备层级较高时，很可能出现数据的丢失，造成设备无法成功升级。

4.6 Mwifi

[English]

Mwifi (MESH Wi-Fi) 是用于 ESP-MDF 中最重要的一个组件，目的是实现 ESP-WIFI-MESH 网络组建、通信。为了更好的使用 ESP-WIFI-MESH，Mwifi 对原生的 API 进一步封装。

4.6.1 功能

- **网络配置**：通过简单的 API 实现对 ESP-WIFI-MESH 网络的配置：根节点配置、网络容量、ESP-WIFI-MESH 网络稳定性、数据传输、路由信息、ESP-WIFI-MESH 连接参数;
- **分片发送**：实现对应用层数据进行分片，将 ESP-WIFI-MESH 底层数据包一个个发送出去;

- **数据压缩**：使用 miniz 对每个数据包进行压缩处理，减少数据包的大小，提高传输速率;
- **单播发送**：当使用单播方式发送时，设备在收到数据包后，设备将向每个目的地址分别发送数据包，数据包在 ESP-WIFI-MESH 网络中经过多跳的方式传输到目的地址;
- **多播发送**：当使用多播方式发送时，设备在收到数据包后，将传给该节点下一层的所有节点，减少数据传输次数，该方式类似与 IP 多播;
- **广播发送**：当使用广播方式发送时，设备将向周围发送 ESP-WIFI-MESH 广播数据包，周围所有的节点都能收到该数据包，收到数据包的设备将继续向周围广播数据包;
- **设备分组**：将希望批量控制的相同类型设备纳入到同一个设备组当中，组的唯一标识为组地址;
- **重包过滤**：使用 Mwifi 提供的 API 接收 ESP-WIFI-MESH 数据包，将过滤收到的相同数据包;
- **线程安全**：在发送 ESP-WIFI-MESH 数据包之前都需要获取到互斥锁并在之后释放，这样就可以在 FreeRTOS 操作系统中使用。

网络配置

可调用 Mwifi 提供的 API 配置 ESP-WIFI-MESH 网络，例如：路由器 SSID 和密码，MESH ID 和密码，节点类型，MESH 信道等。

1. 无路由器场景

在无路由器的情况下，我们只能指定每个节点的类型（根/非根节点）；参考无路由器示例:*example/function_demo/mwifi/no_router*

2. 有路由器场景

在有路由器的情况下，我们可以选择指定每个节点的类型的这种方式，也可以选择根节点自动选择的方式;参考有路由器示例:*example/function_demo/mwifi/router*

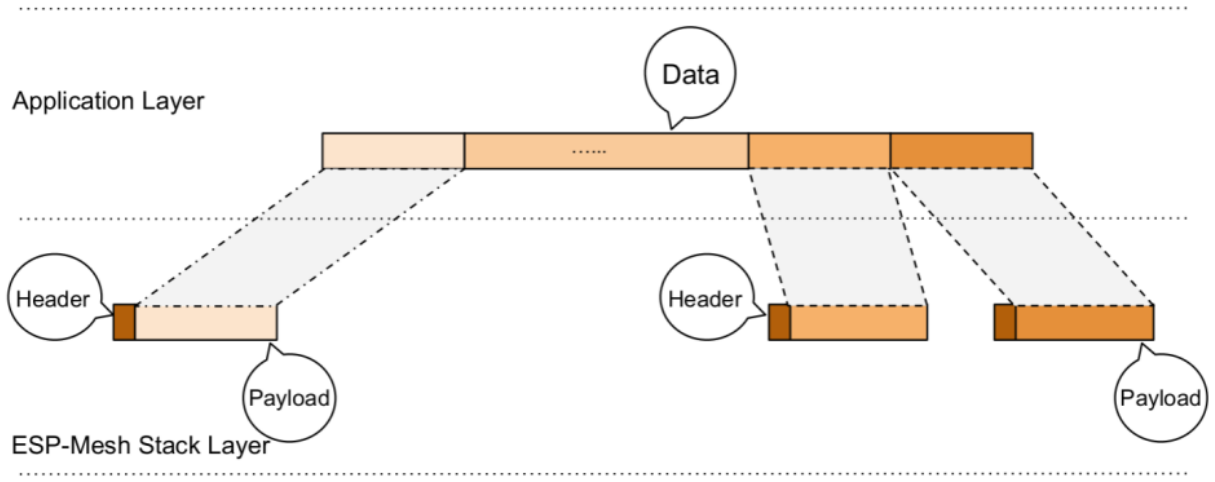
分片发送

在实际的应用中，应用层中每一包的数据可能比较多，经常超过底层每个数据包的最大长度。所以，在发送时经常需要做的就是分片发送。在 ESP-MDF 应用层中每一个数据包允许的最大数据长度为 8095 字节，而 ESP-WIFI-MESH 底层协议栈中每一个数据包允许的最大数据长度为 1472 字节。

数据压缩

考虑到应用层的数据量会比较大，在传输前对应用层数据进行压缩，而在接收到数据后进行解压缩。这都是通过 miniz 数据压缩库实现的。

- 时间：压缩 2 ~ 4 ms; 解压缩 1 ~ 2 ms
- 内存使用：任务的栈空间应不小于 8 KB，动态内存压缩:7 KB，解压缩 2 KB
- 压缩率：与数据内容关系很大，数据重复率越高压缩率越高，适用于 json 等明文格式数据传输



单播发送

如图所示，在单播发送方式中，设备向目的地址发送的数据将在 MEHS 网络中通过多跳的方式传输到目的设备。

注解： 单播向上发送 (子节点发向根节点) 有流控，因此是阻塞的，需要等待发包结束或者出错才能返回

多播发送

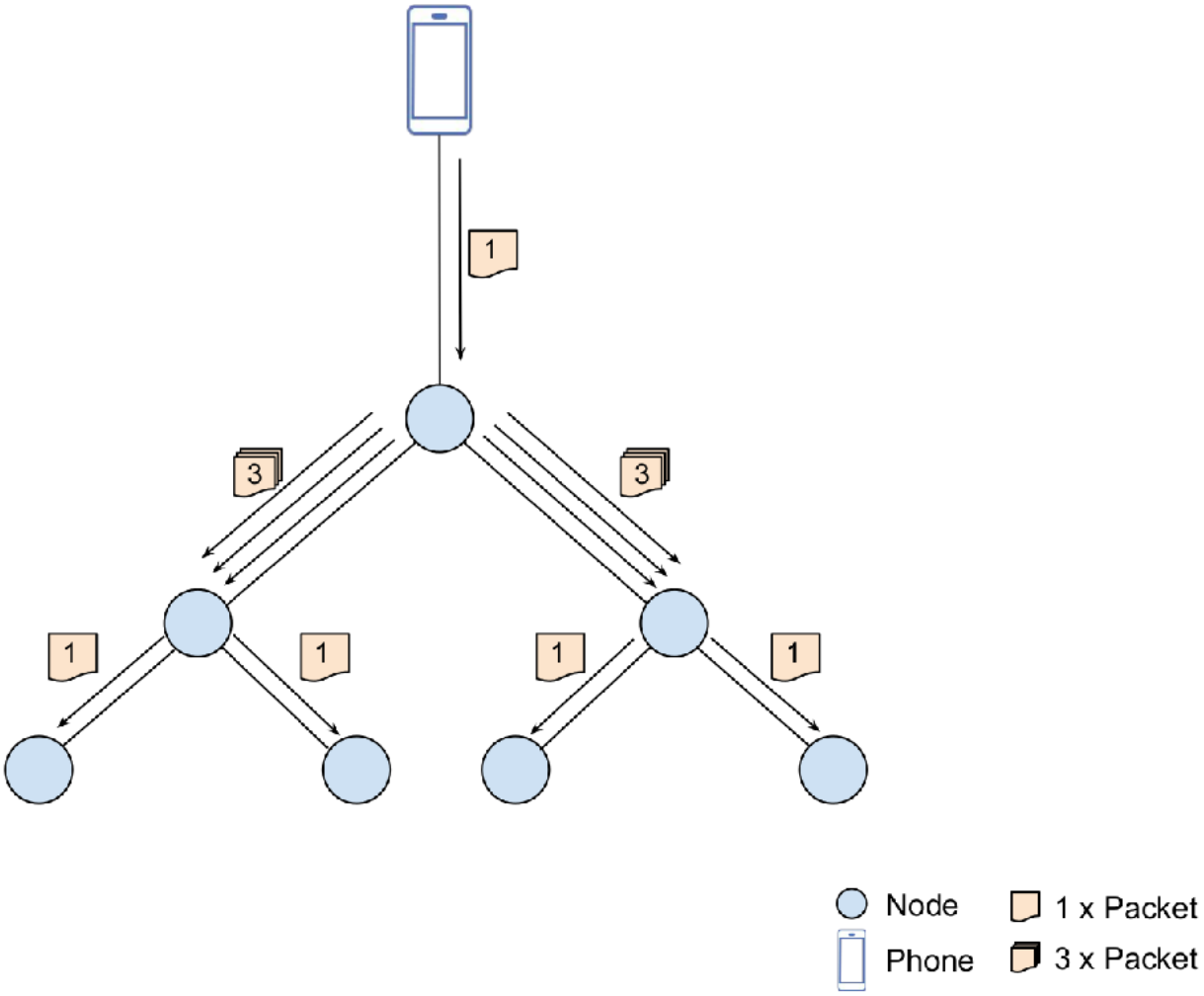
由于有许多的应用需要由一个源节点发送到许多目的节点，即一对多的通信。例如：控制家庭中所有的灯关闭。这时使用多播方式进行就可以明显地减轻网络中各种资源的消耗。

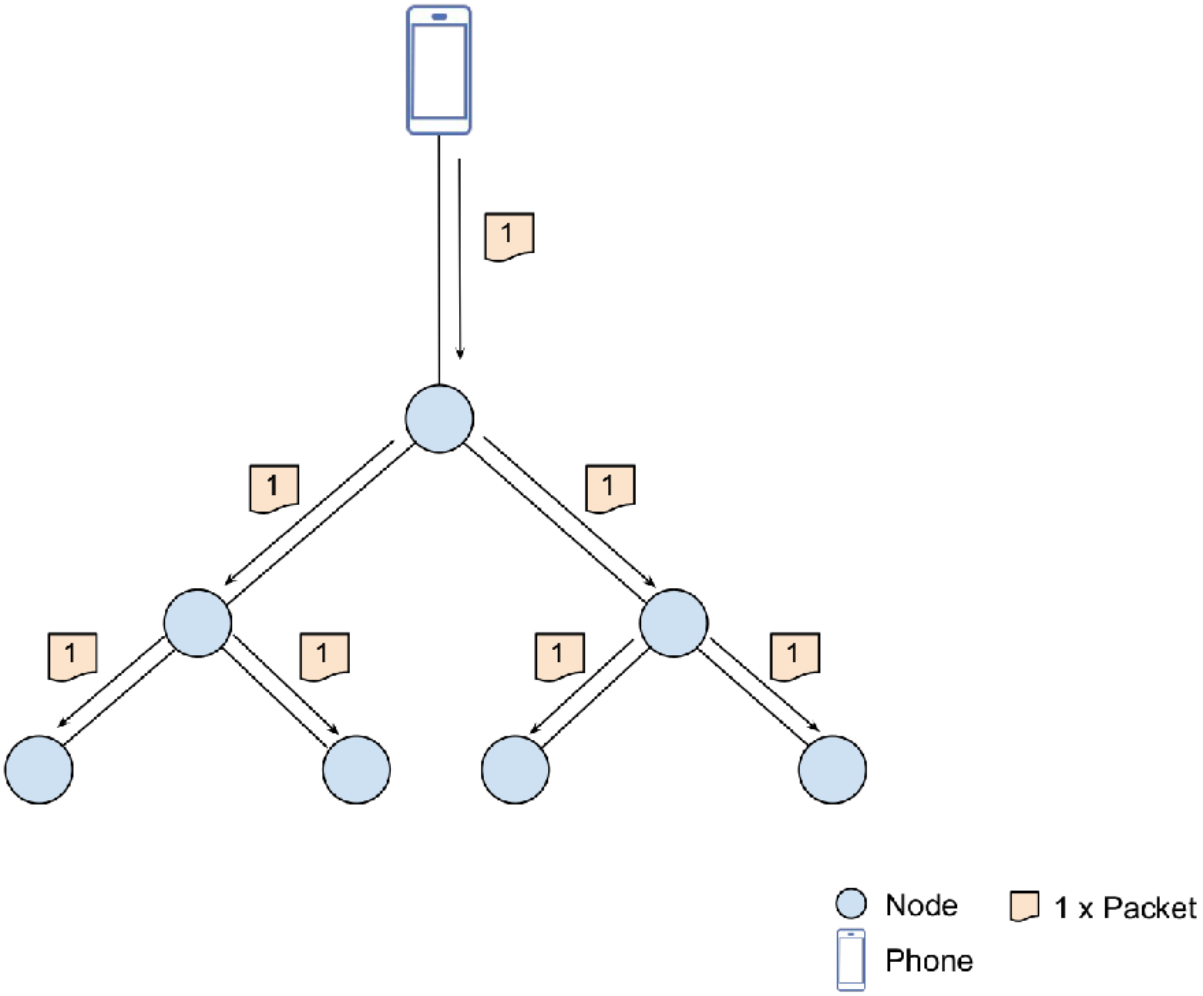
与单播方式相比，在一对多的通信中，多播可大大节约网络资源。从图中可以看出，使用单播方式，源节点需要向所有目的节点传输同样的数据，为此，需要发送 n 个单播，即同一个数据包要发送 n 个副本。但如果使用多播方式，源节点只需将数据发送一次到根节点，根节点在收到数据包时，将其转发到下一层的所有节点，下一层的所有节点在接收到数据包后，做同样的操作，经过这些操作后，该 ESP-WIFI-MESH 网络内所有目的节点都能收到来自源节点的数据包。

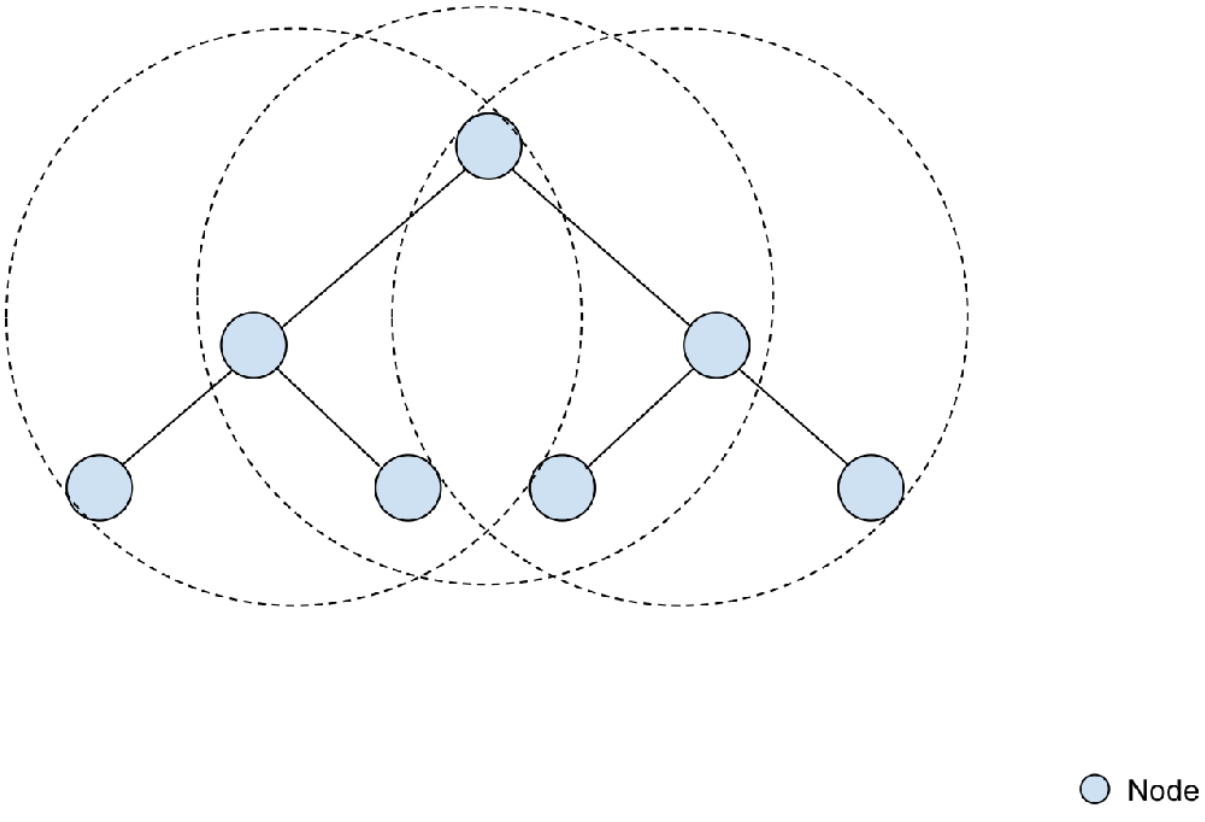
广播发送

在一对多的通信中，使用多播方式发送，可以明显地减轻网络中各种资源的消耗。但是，在有些情况，使用广播的方式也是一种不错的选择。例如：对时延和速率要求高，但可以容忍丢包的场景，如：音频传输。

当使用广播方式发送的时候，设备 A 向环境中广播数据包，收到广播数据包的设备 B 同样向环境中广播数据包。这时设备 A 同样会接收到设备 B 发送的广播数据包，设备 A 接收到这个数据包之后发现这个数据包和之前自己广播的数据包相同，故不再向环境中广播数据包。


1
3
3
1
1
1
1
Node
1 x Packet
Phone
3 x Packet


1
1
1
1
1
1
1
Node
1 x Packet
Phone



设备分组

在 ESP-WIFI-MESH 实际应用时，同一个 ESP-WIFI-MESH 网络中会存在很多设备，并且有很多设备功能类似，在对设备进行控制时，希望批量控制这些设备，这时，就可以将这些功能类似的设备添加到同一个设备组中。

根节点向所有子节点发送数据包，但只有目的组中的节点才能对数据包中的内容进行处理。

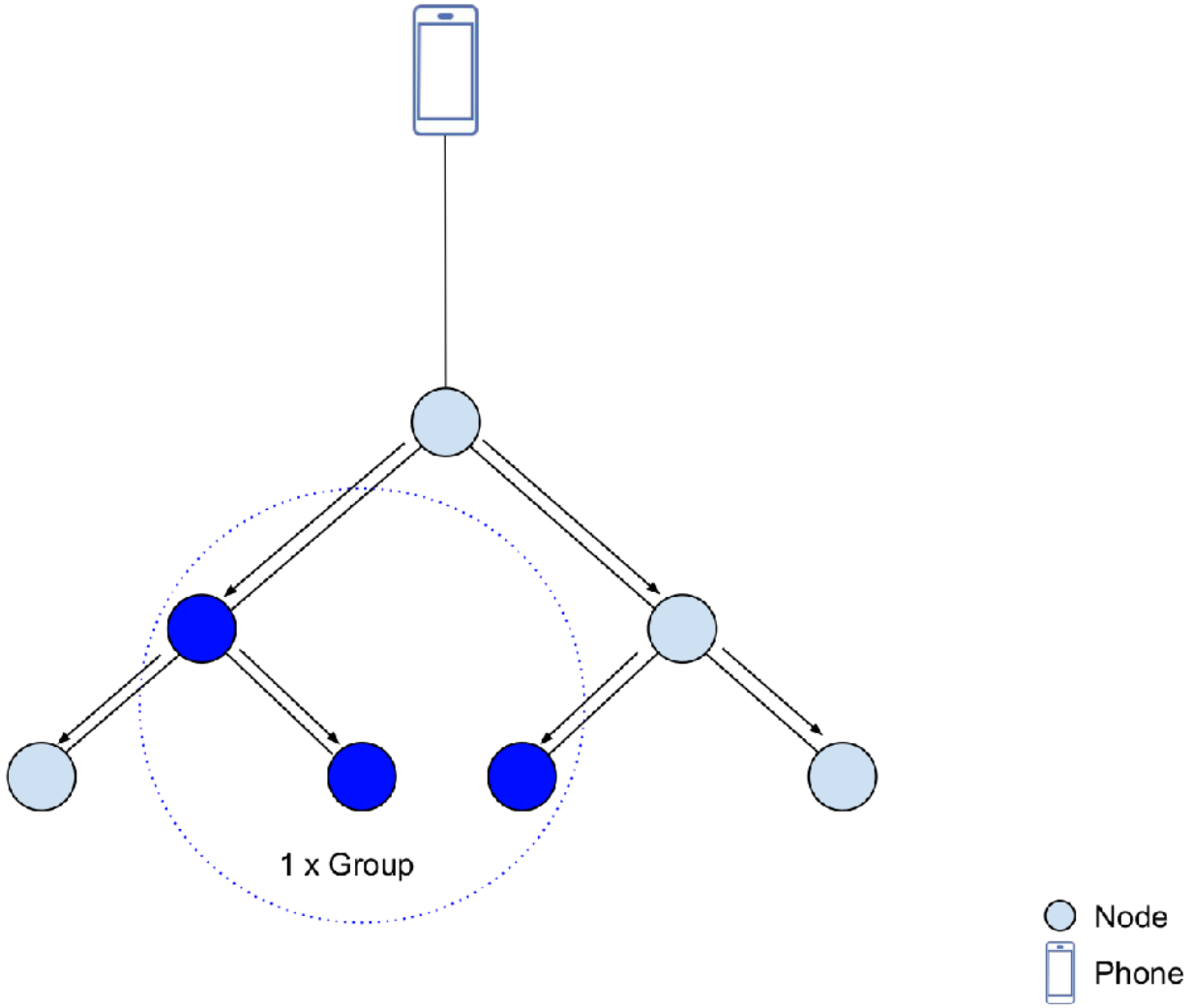
4.7 Mdebug

[English]

Mdebug (Mesh Network debug) 是用于 ESP-MDF 中重要的一种调试方案，目的是将通过无线 espnow 协议、TCP 协议、串口等方式实现 ESP-MDF 设备日志高效的获取，从而更加方便快捷读取设备日志信息，然后可以根据提取出来的日志进行对应的分析等。

Mdebug 指南主要分为以下几个形式：

1. 介绍
2. 功能
3. 调试方法

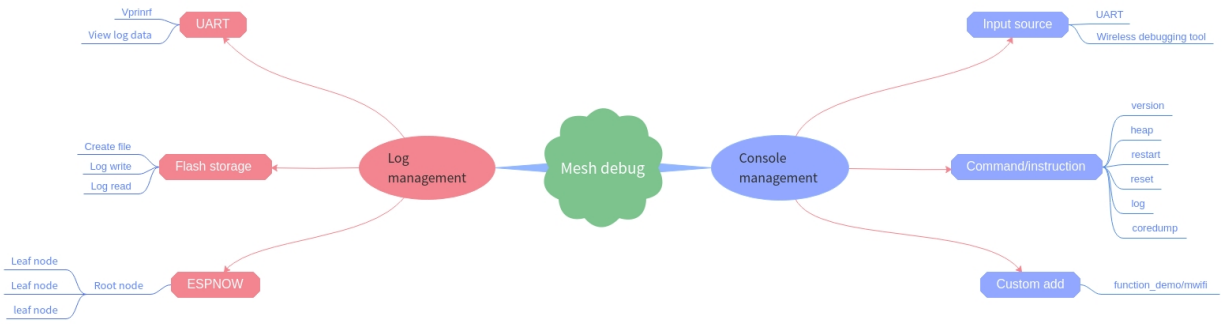

1 x Group
Node
Phone

4. 命令管理
5. 日志管理

4.7.1 介绍

传统的调试方法是通过连接串口读取所有的日志信息，其中这些日志信息有部分是不必要信息，浪费了用户大量筛选时间，而且需要通过 PC 端和设备同时在线连接才能采集到日志信息，导致了时间和资源的浪费。

Mdebug 调试方法与传统的调试方法的不同之处在于，在不影响正常设备的运行的同时，Mdebug 调试可以通过无线调试，日志存储，控制命令筛选日志，从而提高了日志的使用效率和方便性，为设备查找问题和读取需要的信息节约时间和资源。



Mdebug 主要分为命令管理和日志管理这两种方式

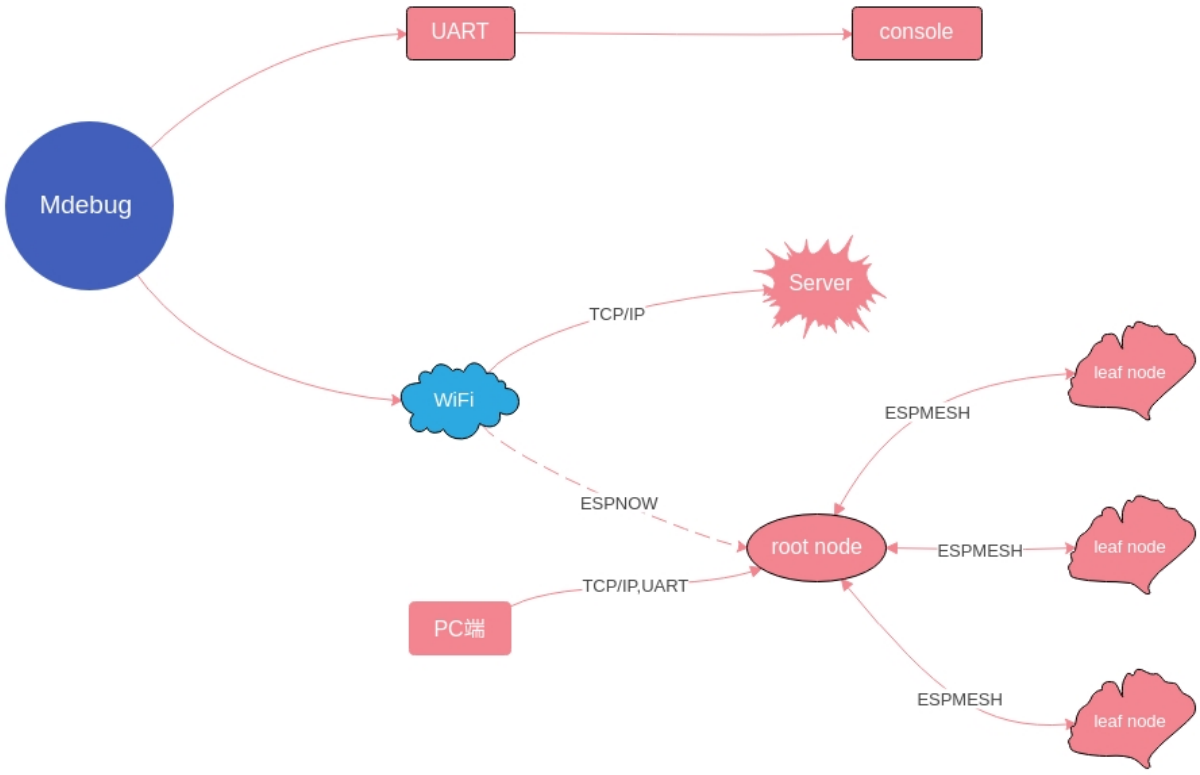
1. 命令管理可以大致分为 输入源、命令/指令、用户自定义添加;
2. 日志管理可以大致分为 *UART* 、*Flash storage* 、*ESPNOW*。

4.7.2 功能

- `mdebug espnow`：将设备中日志数据通过 ESPNOW 无线 WiFi 形式传输给其他设备或者服务器等, 从而提高了调试的方便和效率。
- `mdebug flash`：将设备中日志数据直接写入到 flash 内存中，该 flash 内存中已经创建好内存空间，掉电保护日志信息，可以随时读取该日志信息，提高了数据的可利用性以及客户的方便。
- `mdebug log`：将设备中日志数据通过串口监视器读取，同时设置 *UART*，*flash*，*espnow*'使能状态，其中日志信息将 'ESP_MDF 中 *MDF_LOGI*、*MDF_LOGD*、*MDF_LOGW*、*MDF_LOGE* 等日志信息读取出来。
- `mdebug console`：将设备中日志数据通过在终端中的输入命令行形式读取，提高了用户的可操作性和实用性。

4.7.3 调试方法

调试方法可以概括为两种方法，一种是 UART 串口调试，另一种是无线 WiFi 调试。



1. 串口调试在 PC 端使用串口调试工具（linux 下用 minicom 更好），直接获得输出的日志信息，还有可以通过控制命令行调试方法输出需要的日志信息。
2. 无线 WiFi 调试有两种通讯方式分别是 TCP/IP 协议和 ESPNOW 协议，一种是日志信息通过 TCP/IP 协议传输到 Sever 端，另一种是从子节点的日志信息通过 ESPNOW 协议传输到根节点，然后从根节点的串口或者 TCP/IP 形式提取日志信息。也可以通过控制命令行调试方法输出需要的日志信息。

4.7.4 命令管理

1. 输入源

- **串口** PC 端的监视器
- **ESPNOW** 无线调试工具 (ESP-WROVER-KIT-V2)

2. 命令/操作

2.1 一般指令

- **help** 打印已注册命令及其说明
- **version** 获取芯片和 SDK 版本

- **heap** 获取当前可用堆内存大小
- **restart** 软重启芯片
- **reset** 清除设备所有配置信息

2.2 日志命令

命令定义	log -or [<tag>] [<level>] [-s <addr(xx:xx:xx:xx:xx:xx)>] [-e <enable_type (‘uart’ or’ flash’ or’ espnow’)>] [-d <disable_type(‘uart’ or’ flash’ or’ espnow’)>]	
指令	log -o	将获取日志使能状态
	log -r	读取日志信息
	log -s	将日志发送到指定设备
参数	tag	使用 tag 过滤日志
	level	使用 level 过滤日志
	addr	监视设备 MAC 地址
	e ‘uart’ or ‘flash’ or ‘espnow’	使能 uart，flash，espnow
	d ‘uart’ or ‘flash’ or ‘espnow’	禁止串口，flash，espnow
示例	log mdebug_cmd INFO	设置 TAG 为 mwifi 的日志输出等级为 INFO
	log * NONE	设置所有的日志不输出

2.3 coredump 命令

命令定义	coredump [-loe] [-q] [-s <addr (xx:xx:xx:xx:xx:xx)>]	
指令	coredump -l	获取该设备上的 coredump 数据长度
	coredump -o	读取该设备上的 coredump 数据并打印到控制台
	coredump -e	擦除该设备上的 coredump 数据
	coredump -s	发送设备上的 coredump 数据到指定设备
参数	addr	监视设备 MAC 地址
	sequence	coredump 数据的序号
示例	coredump -s 30:ae:a4:00:4b:90	将 coredump 数据发送到 30:ae:a4:00:4b:90 设备
	coredump -q 110 -s 30:ae:a4:00:4b:90	将序号 110 开始的 coredump 数据发送到 30:ae:a4:80:16:3c 设备

3. 自定义添加

可以根据 ESP-MDF 中的 example:*function_demo/mwifi/console_test*，用户可以自定义添加。

4.7.5 日志管理

Mdebug 中根据日志写入方式不同，大致可以分为两种形式：

1. 设备的日志信息通过打印方式直接从 串口打印出来或者把 日志信息写入 `flash` 存储，然后再调用读取。日志信息的存储将设备日志先写入 flash 内存中（这里的 flash 内存中分配了一个分区为 *storage*，为了存储设备日志，但是这里的分配的内存是有限的，根据用户设置的文件大小而决定），将会以文件的形式暂存起来，然后通过串口或者无线方式将数据以数据包的形式发送给 PC 或者是服务器；
2. 设备将通过 `espnow` 形式发送日志信息。将子节点日志信息通过 ESP-MESH 网络将数据发送给根节点，再从根节点的设备读取日志信息。

Mdebug 中根据日志读取方式不同，日志中有三种使能状态，分别是 `uart`、`flash`、`espnow` 使能。

1. UART 使能

将串口使能，日志信息将会通过 `vprintf` 打印出来。

1. 读取日志的 I/O 口是 UART0，该串口的引脚为 TXD0 是 GPIO1, RXD0 是 GPIO3，同时也是下载串口。

UART0	
TXDO	RXD0
GPIO1	GPIO3

2. 读取日志信息，诊断设备出现的问题和查找问题；如果在设备正常运行时，可以关闭串口使能，如果不关闭串口打印，将会占用内存，同时串口打印信息过多，还会激活看门狗，使正常设备运行程序 `Backtrace`。

3. 串口默认状态为使能状态，用户可以根据自己需要进行关闭。

2. Flash 使能

写入 flash 使能，将日志信息存入 flash 中。

2.1 将 log 保存到 flash

分区表中选择出来一定的内存空间为 `storage`，这里分配的内存为了给 log 写入到 flash 中提供内存空间。文件名为 `spiffs`,

spiffs 分区:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
ota_0,    app,  ota_0,    0x10000,  1920k
ota_1,    app,  ota_1,    ,         1920k
coredump, data, coredump, ,         64K
storage,  data, spiffs,   ,         64K
reserved, data, 0xfe,     ,         64K
```

注解:

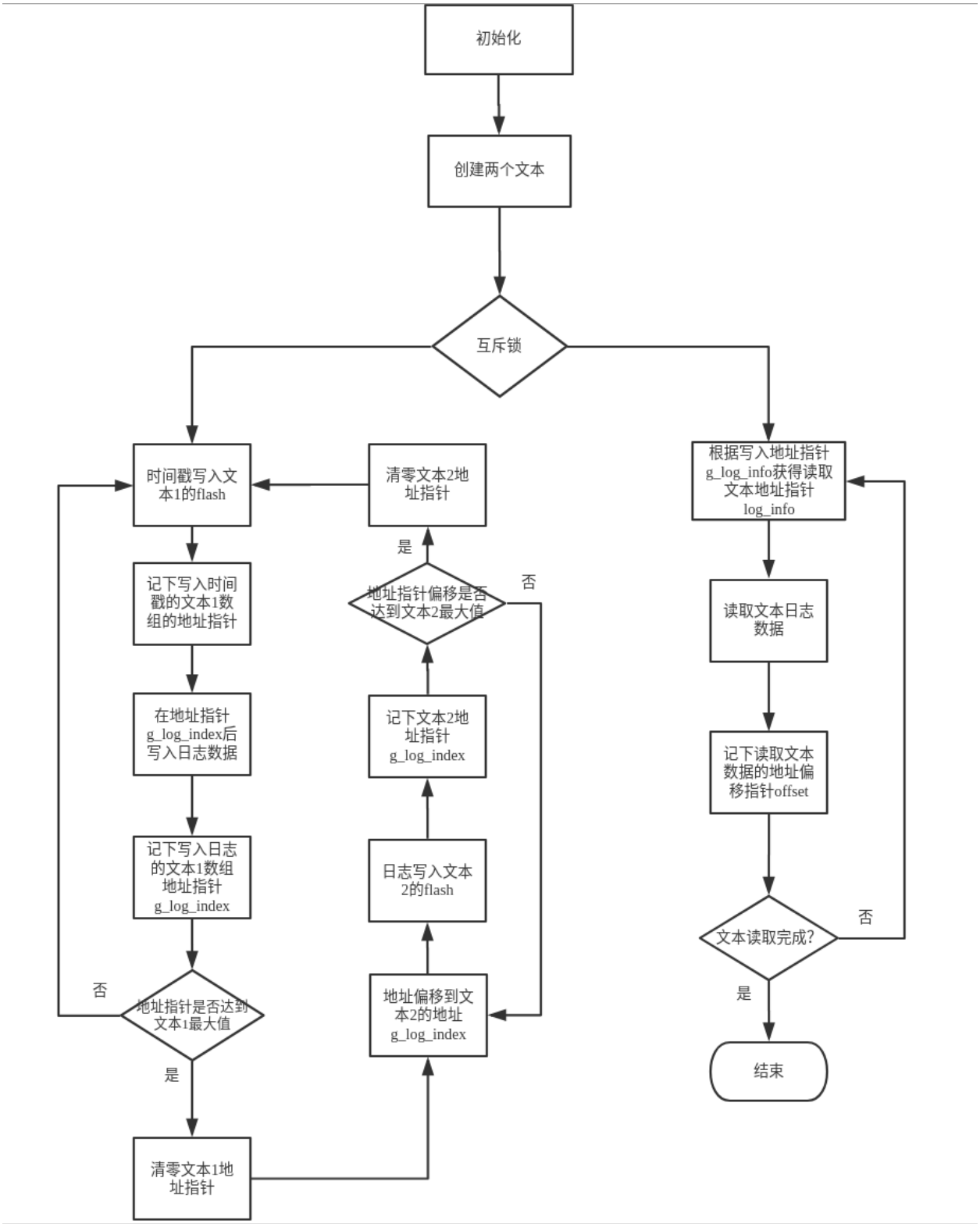
1. 更新分区表前，需要先擦除整个 flash；

2. 分区表无法通过 OTA 的方式进行修改；

3. 文件空间大小可以根据用户合理的选择进行内存分配。

2.2 日志信息存取

1. 日志初始化，创建两个文本 flash 内存空间，并获取文本的状态 `stat`，判断写入还是读取，使用到主要函数为 `esp_vfs_spiffs_register`、`esp_spiffs_info`、`sprintf`、`fopen`；


初始化
创建两个文本
互斥锁
时间戳写入文本1的flash
记下写入时间戳的文本1数组的地址指针
在地址指针g_log_index后写入日志数据
记下写入日志的文本1数组地址指针g_log_index
地址指针是否达到文本1最大值
否
是
清零文本1地址指针
清零文本2地址指针
是
地址指针偏移是否达到文本2最大值
否
记下文本2地址指针g_log_index
日志写入文本2的flash
地址偏移到文本2的地址g_log_index
根据写入地址指针g_log_info获得读取文本地址指针log_info
读取文本日志数据
记下读取文本数据的地址偏移指针offset
文本读取完成?
否
是
结束

2. 存取中加入一个互斥锁，当读取日志信息时，则会关闭写入功能，主要用到的函数为 `xSemaphoreTake`，`xSemaphoreGive`；
3. 写入 flash 中，先写入时间戳，并记下文本数组的地址指针 *g_log_info[g_log_index]*，主要用到的函数为 `time`，`localtime_r`，`strftime`；
4. 写入日志数据，同时记下写入文本 1 数组的地址指针 *g_log_index*, 为了下次日志写入 flash 做好地址寻址，使用到主要函数为 `fseek`、`fwrite`;
5. 判断，如果文本 1 数组的地址指针已满，清零文本 1 的地址指针，地址偏移到文本 2 的地址指针，开始写入日志数据；如果未满，将会继续在文本 1 中写入日志数据；
6. 同理，当文本 2 数组的地址指针已满，清零文本 2 的地址指针，地址偏移到文本 1 的地址指针，开始写入日志数据；如果未满，将会继续在文本 2 中写入日志数据；
7. 根据写入地址指针 *g_log_info* 获取读取文本地址指针 *log_info* ，然后读取文本中日志数据，同时也记下读取文本数据的地址偏移指针 *offset*，为下次从 flash 读取日志做好地址寻址，使用到主要函数为 `fseek`、`fread`;
8. 判断，如果文本未读取，将会继续读取文本日志数据；如果文本读取完成，将会结束读取任务。

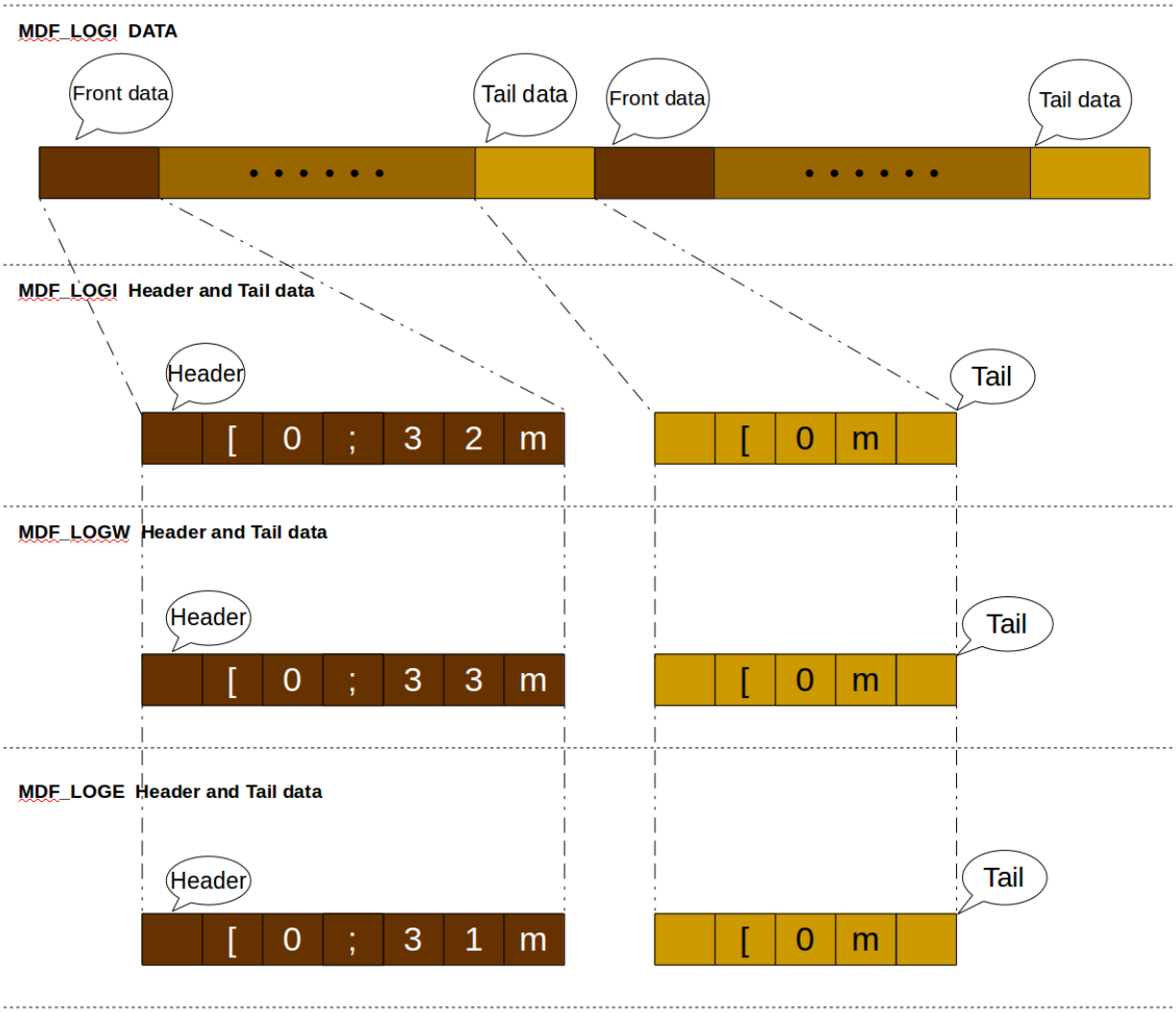
注解:

1. 日志数据的头是增加了时间戳，这里只是作为实验，并没有实时校准，用户可以根据自己的需求进行修改；
2. 日志存储的文件大小为 CONFIG_MDEBUG_FLASH_FILE_MAX_SIZE = 16384，用户可以根据自己需求修改日志文件的存储空间;
3. 日志重定向将日志存储信息输出重新进行定义，这是为了对于日志写入 flash 进行调试，当日志输出信息出现问题时，可以更好的调试日志输出信息是否正确，使用到的调试函数为 `MDEBUG_PRINTF(fmt, ...)`；
4. 增加了数据的擦除，当数据存满，将会清除数据指针，重新开始从文件头指针地址开始写入或者读取，用到的主要函数 `rewind`。

2.3 日志数据的格式

日志数据将来自 *ESP_MDF* 中 *MDF_LOGI*、*MDF_LOGD*、*MDF_LOGW*、*MDF_LOGE* 等，这是通过 IDF 的日志库会默认使用类 vprintf 的函数将格式化的字符串输出到专用的 UART 上。提取出来的数据如下图所示：

因在 MDF 中的日志信息存在头和尾有不需要的数据，这就需要提取和选择有效的字符串数据信息，因此需要将其进行去除和筛选，然后再进行日志数据提取。`Front data` 中包含了字体颜色


MDF_LOGI DATA
Front data
Tail data
Front data
Tail data
MDF_LOGI Header and Tail data
Header
[0 ; 3 2 m
Tail
[0 m
MDF_LOGW Header and Tail data
Header
[0 ; 3 3 m
Tail
[0 m
MDF_LOGE Header and Tail data
Header
[0 ; 3 1 m
Tail
[0 m

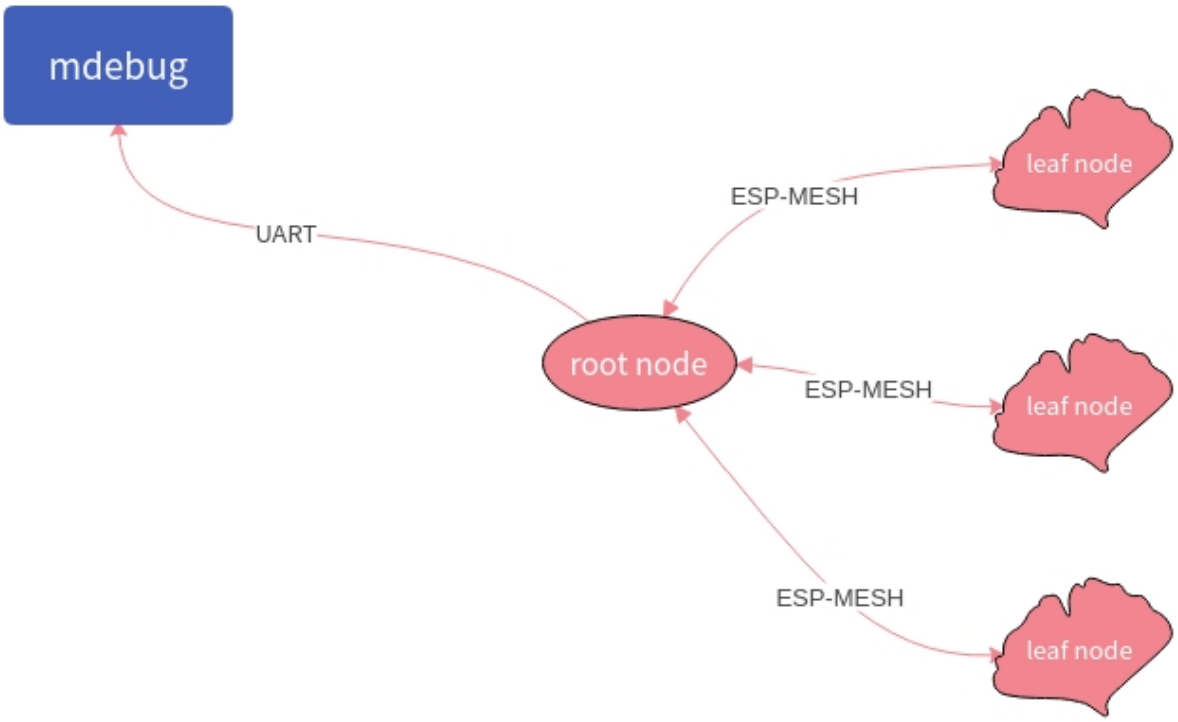
等添加的信息，所以需要将头部部分数据去除，`Tail data` 中包含了换行等数据，同样是需要去除，对于 *MDF_LOGD* 没有这些无用数据，所以不需要进行处理。

3. ESPNOW 使能

3.1 ESP-NOW 特性

- 收发双方必须在同一个信道上
- 接收端在非加密通信的情况下可以不添加发送端的 MAC 地址（加密通信时需要添加），但是发送端必须要添加接收端的 MAC 地址
- ESP-NOW 最多可添加 20 个配对设备，同时支持其中最多 6 个设备进行通信加密
- 通过注册回调函数的方式接收数据包，以及检查发送情况（成功或失败）
- 利用 CTR 和 CBC-MAC 协议 (CCMP) 保护数据的安全

3.2 ESP-NOW 使能流程



通过使能 *espnow*，可以将子节点的日志数据通过 espnow 发送给根节点，使得从根节点读取子节点的日志信息，然后通过串口读取日志信息。

这个可以使用 example:*wireless_debug* 进行测试，ESP-NOW debug 接收板可以不直接与路由器进行连接，只需与 ESP-MESH 网络在同一信道上即可。若需要与接收其他 ESP-MESH 设备的 log，需要在监听的设备中添加以下代码：

```
MDF_ERROR_ASSERT(mdebug_console_init());
MDF_ERROR_ASSERT(mdebug_espnow_init());
mdebug_cmd_register_common();
```

关于更多 *espnow* 可以参看 example:*wireless_debug* 以及官方文档 ESPNOW。

注解: 由于 ESP-NOW 和 ESP-MESH 一样，都是通过 Wi-Fi 接口进行数据包收发，因此，当 ESP-MESH 设备数据传输量较大时，会对其控制命令接收或数据传输产生一些延时。

经实际测试，在网络环境良好的情况下，以下配置参数导致的 ESP-MESH 设备延时是可以忽略的阈值:

- 50 个 ESP-MESH 设备（设备数量越多，网络环境越差）
- ESP-NOW 接收端添加 10 个 ESP-MESH 设备（接收端添加数量越多，网络环境越差）
- 传输日志级别为 info （日志级别越低，网络环境越差）

CHAPTER 5

ESP-MDF 贡献指南

[English]

本指南简要说明如何在 ESP-MDF 中添加代码，详细说明参见 贡献指南。

5.1 添加代码

若您在 ESP-MDF 中增加了新的功能，请在 `examples` 下增加使用示例，并将示例名称和路径添加到文件 `.gitlab-ci.yml` 中，使其能在 `gitlab` 测试并运行。请参考如下方式添加:

```
build_example_get_started:
  <<: *build_template
  variables:
    ARTIFACTS_NAME: "get_started"
    EXAMPLE_PATH: "examples/get_started"
```

5.2 格式化代码

您在提交代码之前需要对代码进行格式化，ESP-MDF 使用的格式化式工具为：astyle 和 dos2unix。您需要先安装他们，在 Linux 安装命令如下:

```
sudo apt-get install astyle
sudo apt-get install dos2unix
```

运行如下代码格式化命令:

```
tools/format.sh new_file.c
```

5.3 API 文档生成

在编写代码注释时，请遵循 Doxygen 格式。您可以通过 Doxygen 按如下步骤自动生成 API 文档:

```
sudo apt-get install doxygen
export MDF_PATH=~/esp/esp-mdf
cd ~/esp/esp-mdf/docs
pip install -r requirements.txt
cd ~/esp/esp-mdf/docs/en
make html
```

CHAPTER 6

ESP-MDF Versions

The ESP-MDF GitHub repository is updated regularly, especially on the "master branch" where new development happens. There are also stable releases which are recommended for production use.

6.1 Releases

Documentation for the current stable version can always be found at this URL:

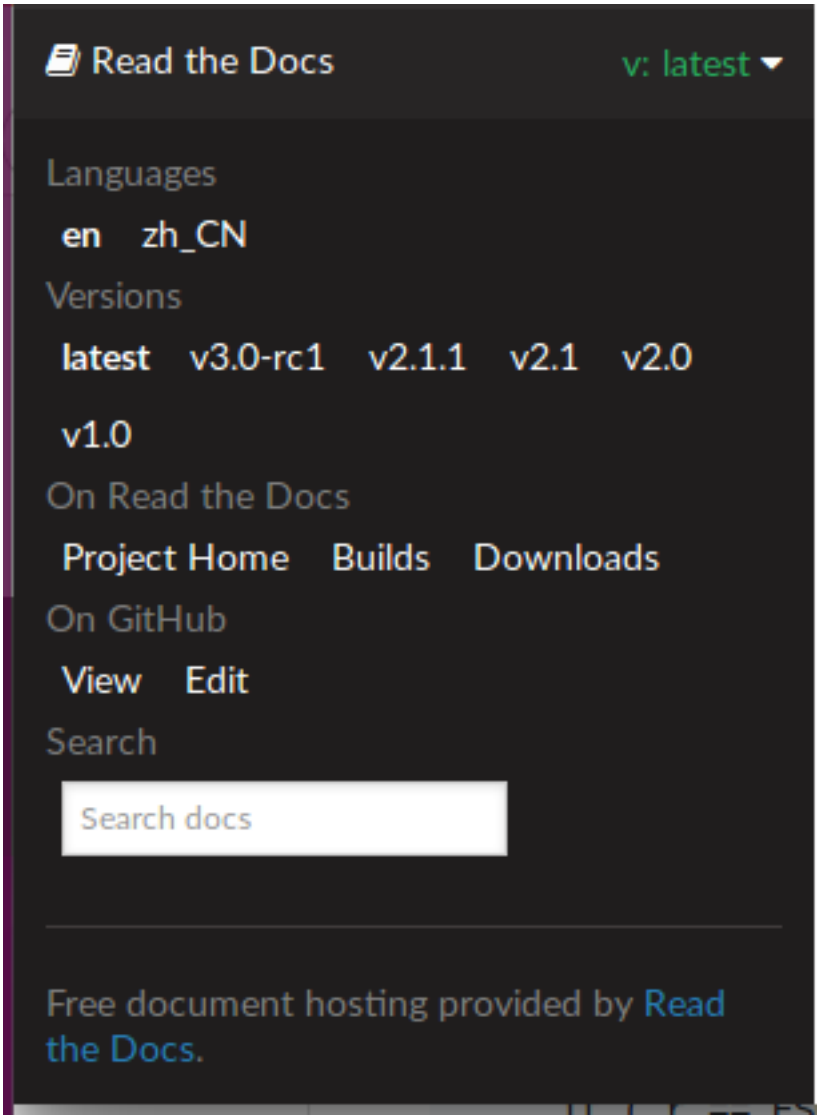
https://docs.espressif.com/projects/esp-mdf/en/stable/

Documentation for the latest version ("master branch") can always be found at this URL:

https://docs.espressif.com/projects/esp-mdf/en/latest/

The full history of releases can be found on the GitHub repository Releases page. There you can find release notes, links to each version of the documentation, and instructions for obtaining each version.

Documentation for all releases can also be found in the HTML documentation by clicking the "versions" pop up in the bottom-left corner of the page. You can use this popup to switch between versions of the documentation.



6.2 Which Version Should I Start With?

- For production purposes, use the current stable version. Stable versions have been manually tested, and are updated with "bugfix releases" which fix bugs without changing other functionality (see *Versioning Scheme* for more details).
- For prototyping, experimentation or for developing new ESP-MDF features, use the latest version (master branch in Git). The latest version in the master branch has all the latest features and has passed automated testing, but has not been completely manually tested ("bleeding edge").
- If a required feature is not yet available in a stable release, but you don' t want to use the master branch, it is possible to check out a pre-release version or a release branch. It is recommended to start from a stable version and then follow the instructions for *Updating to a Pre-Release Version* or *Updating to a Release Branch*.

See *Updating ESP-MDF* if you already have a local copy of ESP-MDF and wish to update it.

6.3 Versioning Scheme

ESP-MDF uses Semantic Versioning. This means:

- Major Releases like `v3.0` add new functionality and may change functionality. This includes removing deprecated functionality.

  When updating to a new major release (for example, from `v2.1` to `v3.0`), some of your project' s code may need updating and functionality will need to be re-tested. The release notes on the Releases page include lists of Breaking Changes to refer to.

- Minor Releases like `v3.1` add new functionality and fix bugs but will not change or remove documented functionality, or make incompatible changes to public APIs.

  If updating to a new minor release (for example, from `v3.0` to `v3.1`) then none of your project' s code should need updating, but you should re-test your project. Pay particular attention to items mentioned in the release notes on the Releases page.

- Bugfix Releases like `v3.0.1` only fix bugs and do not add new functionality.

  If updating to a new bugfix release (for example, from `v3.0` to `v3.0.1`), you should not need to change any code in your project and should only need to re-test functionality relating directly to bugs listed in the release notes on the Releases page.

6.4 Checking The Current Version

The local ESP-MDF version can be checked using git:

```
cd $MDF_PATH
git describe --tags --dirty
```

The version is also compiled into the firmware and can be accessed (as a string) via the macro `MDF_VER`. The default ESP-MDF bootloader will print the version on boot (these versions in code will not always update, it only changes if that particular source file is recompiled).

Examples of ESP-MDF versions:

Version String	Meaning
`v0.5.0`	Master branch pre-release, in development for version 3.2. 306 commits after v3.2 development started. Commit identifier `beb3611ca`.

6.5 Git Workflow

The development (Git) workflow of the Espressif ESP-MDF team is:

- New work is always added on the master branch (latest version) first. The ESP-MDF version on `master` is always tagged with `-dev` (for "in development"), for example `v3.1-dev`.
- Changes are first added to an internal Git repository for code review and testing, but are pushed to GitHub after automated testing passes.
- When a new version (developed on `master`) becomes feature complete and "beta" quality, a new branch is made for the release, for example `release/v3.1`. A pre-release tag is also created, for example `v3.1-beta1`. You can see a full list of branches and a list of tags on GitHub. Beta pre-releases have release notes which may include a significant number of Known Issues.
- As testing of the beta version progresses, bug fixes will be added to both the `master` branch and the release branch. New features (for the next release) may start being added to `master` at the same time.
- Once testing is nearly complete a new release candidate is tagged on the release branch, for example `v3.1-rc1`. This is still a pre-release version.
- If no more significant bugs are found or reported then the final Major or Minor Version is tagged, for example `v3.1`. This version appears on the Releases page.
- As bugs are reported in released versions, the fixes will continue to be committed to the same release branch.
- Regular bugfix releases are made from the same release branch. After manual testing is complete, a bugfix release is tagged (i.e. `v3.1.1`) and appears on the Releases page.

6.6 Updating ESP-MDF

Updating ESP-MDF depends on which version(s) you wish to follow:

- *Updating to Stable Release* is recommended for production use.
- *Updating to Master Branch* is recommended for latest features, development use, and testing.
- *Updating to a Release Branch* is a compromise between these two.

注解: These guides assume you already have a local copy of ESP-MDF.

6.6.1 Updating to Stable Release

To update to new ESP-MDF releases (recommended for production use), this is the process to follow:

- Check the Releases page regularly for new releases.
- When a bugfix release for a version you are using is released (for example if using `v3.0.1` and `v3.0.2` is available), check out the new bugfix version into the existing ESP-MDF directory:

```
cd $MDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

- When major or minor updates are released, check the Release Notes on the releases page and decide if you would like to update or to stay with your existing release. Updating is via the same Git commands shown above.

注解: If you installed the stable release via zip file rather than using git, it may not be possible to change versions this way. In this case, update by downloading a new zip file and replacing the entire `MDF_PATH` directory with its contents.

6.6.2 Updating to a Pre-Release Version

It is also possible to `git checkout` a tag corresponding to a pre-release version or release candidate, the process is the same as *Updating to Stable Release*.

Pre-release tags are not always found on the Releases page. Consult the list of tags on GitHub for a full list. Caveats for using a pre-release are similar to *Updating to a Release Branch*.

6.6.3 Updating to Master Branch

注解: Using Master branch means living "on the bleeding edge" with the latest ESP-MDF code.

To use the latest version on the ESP-MDF master branch, this is the process to follow:

- Check out the master branch locally:

```
cd $MDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- Periodically, re-run `git pull` to pull the latest version of master. Note that you may need to change your project or report bugs after updating master branch.

- To switch from `master` to a release branch or stable version, run `git checkout` as shown in the other sections.

重要: It is strongly recommended to regularly run `git pull` and then `git submodule update --init --recursive` so a local copy of `master` does not get too old. Arbitrary old master branch revisions are effectively unsupportable "snapshots" that may have undocumented bugs. For a semi-stable version, try *Updating to a Release Branch* instead.

6.6.4 Updating to a Release Branch

In stability terms, using a release branch is part-way between using `master` branch and only using stable releases. A release branch is always beta quality or better, and receives bug fixes before they appear in each stable release.

You can find a list of branches on GitHub.

For example, to follow the branch for ESP-MDF v3.1, including any bugfixes for future releases like `v3.1.1`, etc:

```
cd $MDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule --update --init --recursive
```

Each time you `git pull` this branch, ESP-MDF will be updated with fixes for this release.

注解: The is no dedicated documentation for release branches. It is recommended to use the documentation for the closest version to the branch which is currently checked out.

CHAPTER 7

资源

- 您可以在 ESP32 论坛 中提出您的问题，访问社区资源。
- 您可以通过 GitHub 的 Issues 版块提交 bug 或功能请求。在提交新 Issue 之前，请先查看现有的 Issues。
- 您可以在 ESP32 IoT Solution 库中找到基于 ESP-IDF 的 解决方案、应用实例、组件和驱动 等内容。
- 通过 Arduino 平台开发应用，请参考 ESP32 Wi-Fi 芯片的 Arduino 内核。
- 关于 ESP32 的书籍列表，请查看 乐鑫 网站。
- 如果您有兴趣参与到 ESP-IDF 的开发，请查阅 Contributions Guide。
- 关于 ESP32 的其它信息，请查看官网 文档 版块。
- 本文档的镜像网址为：https://esp-mdf.readthedocs.io/zh_CN/latest/ 或 https://docs.espressif.com/projects/esp-mdf/zh_CN/latest/。

CHAPTER 8

Copyrights and Licenses

8.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2018 Espressif Systems. This source code is licensed under the ESPRESSIF MIT License as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses:

Please refer to the COPYRIGHT in ESP-MDF Programming Guide

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

CHAPTER 9

About

This is documentation of ESP-MDF, the framework to develop mesh applications for ESP32 chip by Espressif.

The **ESP32** is 2.4 GHz Wi-Fi and Bluetooth combo, 32 bit dual core chip running up to 240 MHz, designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It has several peripherals on board including I2S interfaces to easy integrate with dedicated mesh chips. These hardware features together with the ESP-MDF software provide a powerful platform to implement mesh applications including native wireless networking and powerful user interface.

The **ESP-MDF** provides a range of API components including **Mesh Streams**, **Codecs** and **Services** organized in **Mesh Pipeline**, all integrated with mesh hardware through **Media HAL** and with **Peripherals** onboard of **ESP32**.

The ESP-MDF also provides integration with **Aliyun** cloud services.

The **ESP-MDF** builds on well established, FreeRTOS based, Espressif IOT Development Framework ESP-IDF.

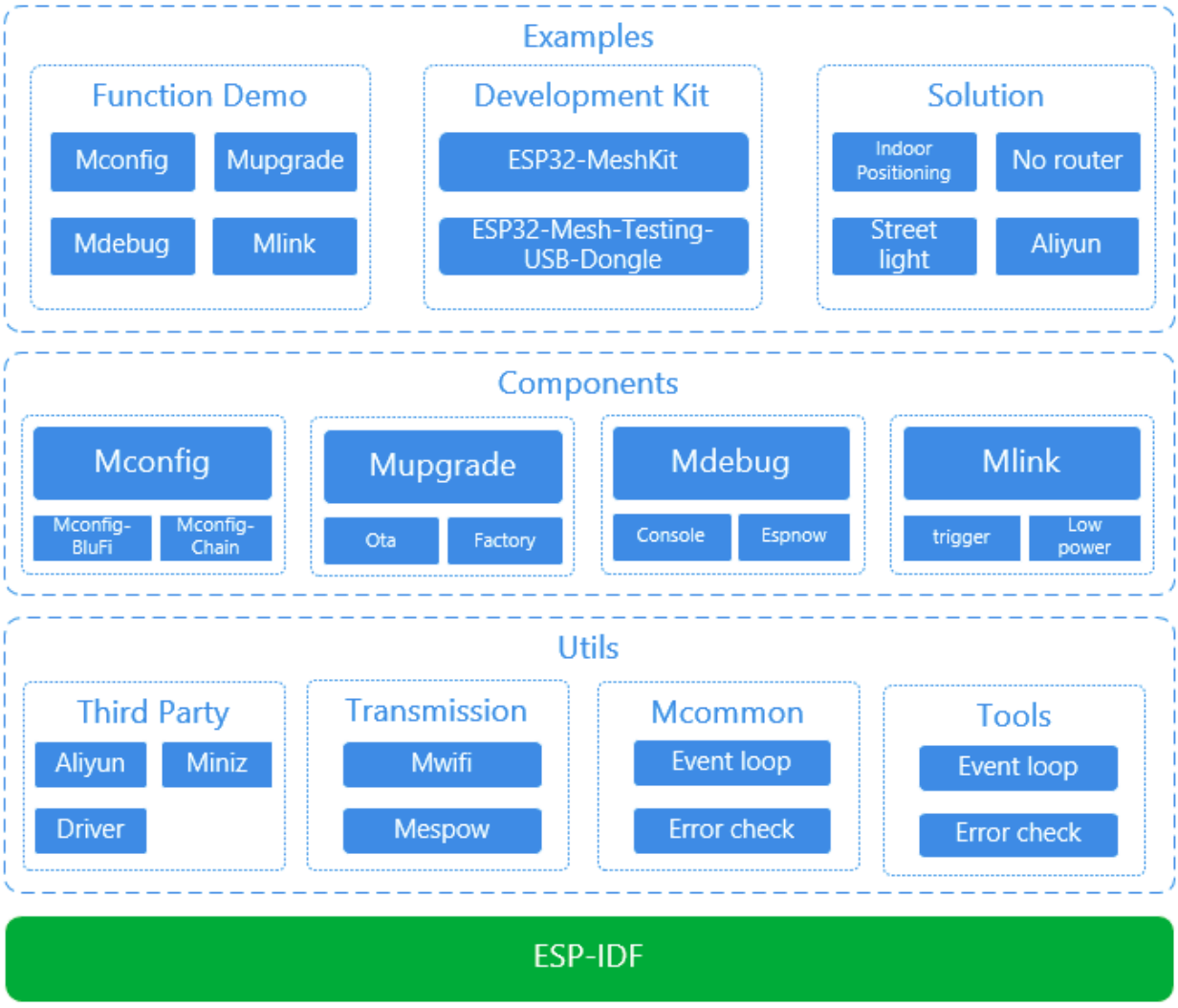


图 1: Espressif Mesh Development Framework

CHAPTER 10

Switch Between Languages/切换语言

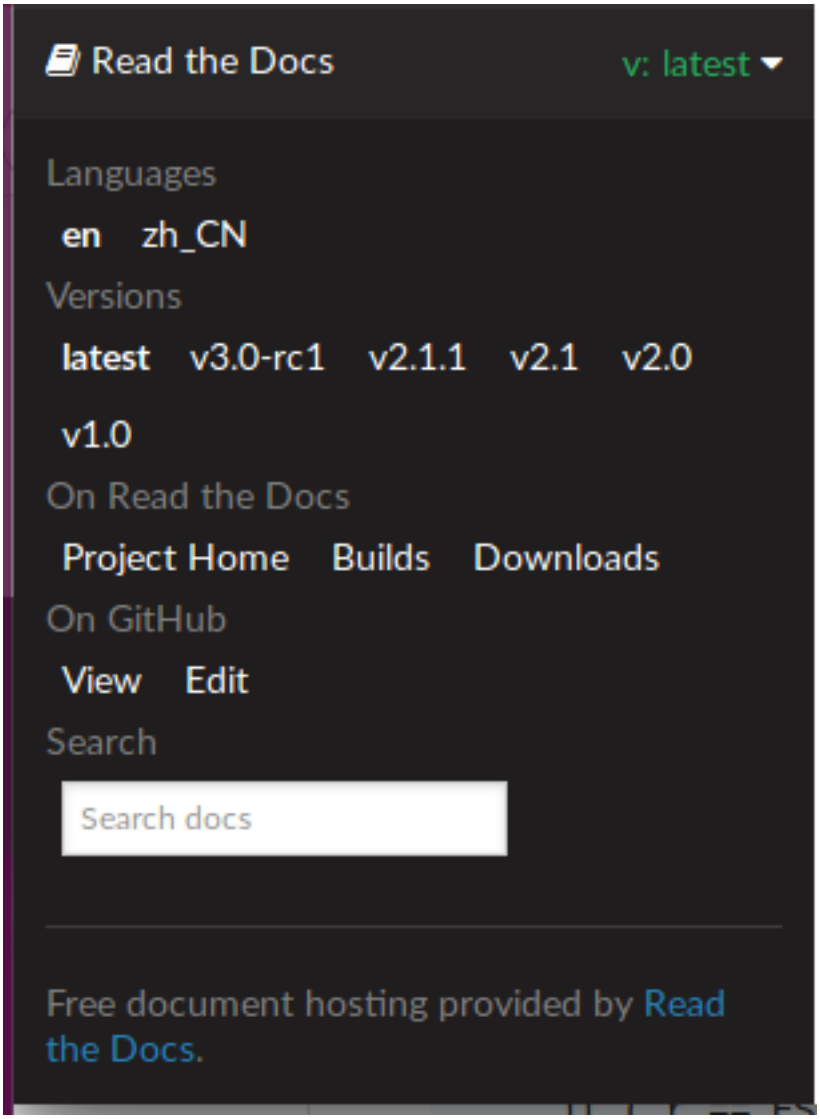
The ESP-MDF Programming Manual is now available in two languages. Please refer to the English version if there is any discrepancy.

《ESP-MDF 编程手册》部分文档现在有两种语言的版本。如有出入请以英文版本为准。

- English/英文
- Chinese/中文

You can easily change from one language to another by the panel on the sidebar like below. Just click on the **Read the Docs** title button on the left-bottom corner if it is folded.

如下图所示，你可使用边栏的面板进行语言的切换。如果该面板被折叠，点击左下角 **Read the Docs** 标题按钮来显示它。


Read the Docs
v: latest
Languages
en
zh_CN
Versions
latest
v3.0-rc1
v2.1.1
v2.1
v2.0
v1.0
On Read the Docs
Project Home
Builds
Downloads
On GitHub
View
Edit
Search
Search docs
Free document hosting provided by Read the Docs.

- genindex

索引

符号

C

E

L

M

N

P

环境变量