ESP8266

Mesh User Guide

Version 1.2

About This Guide

This document introduces the user to the specifications of ESP32 hardware, including the following topics:

Chapter	Title	Subject
Chapter 1	Overview	Provides an overview of ESP-Mesh, including some concepts and network structure.
Chapter 2	Mesh Header	Introduces the mesh header format and details about the fields and codes.
Chapter 3	API Reference	Introduces the data structures and the APIs.
Chapter 4	Sample Code	Provides some sample codes for mesh development.

Release Notes

Date	Version	Release notes
2015.07	V1.0	First release.
2015.09	V1.1	Update Chapter 3.
2016.01	V1.2	Add Chapter 2 and 4, update Chapter 1 and 3.

Note :

This current version is an early release to support initial product developers. The content is subject to change without advance notice.

Table of Contents

1. Overview

With the development of the Internet of Things (IoT), an increasing number of nodes need to connect to internet. However, only limited number of nodes can directly connect to the same router (usually fewer than 32), making it impossible to have all nodes directly connected to one router when there are a large number of nodes. Two solutions are currently available for this problem.

- Super router: we can increase the capacity of the router, so that more nodes can directly connect to the router.
- Mesh network: the nodes can establish a network and forward packets.

The ESP8266 uses mesh network as shown in **Figure 1-1**. As a result, a large number of nodes can connect to the internet without any improvements of the current router.

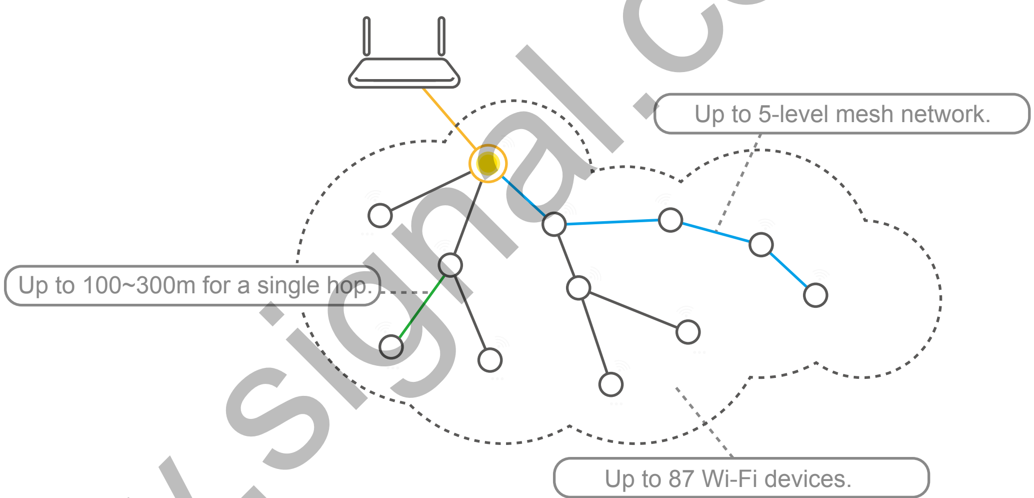


Figure 1-1. ESP-Mesh Network

1.1. Concept

IOT Espressif App

IOT Espressif App (hereinafter referred to as the IOT app) is a mobile application developed by Espressif. It can be used to achieve local and remote control of WiFi devices, including smart lights and smart switches.

ESP-Touch

ESP-Touch is a technology developed by Espressif to connect WiFi devices to router.

Smart Config Mode for ESP-Touch

The device can be configured by ESP-Touch only when it is in Smart Config Mode. This status is called ESP-Touch status. For details of configuration, refer to 2. Mesh Network.

Local Device

As shown in **Figure 1-2**, a device that is configured to the router by ESP-Touch, but not activated on the server-end is a local device.

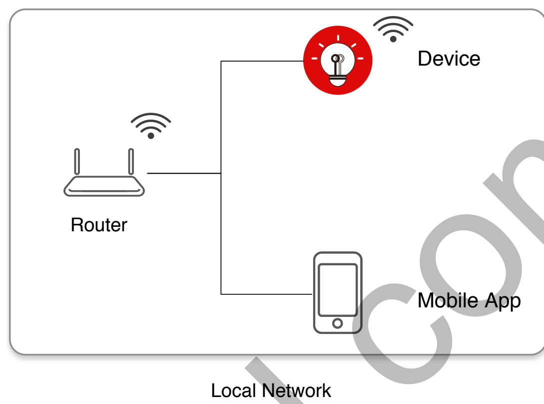


Figure 1-2. Local Network

Cloud Device

As shown in **Figure 1-3**, a device that is configured to the router by ESP-Touch, and activated on the server-end is a cloud device.

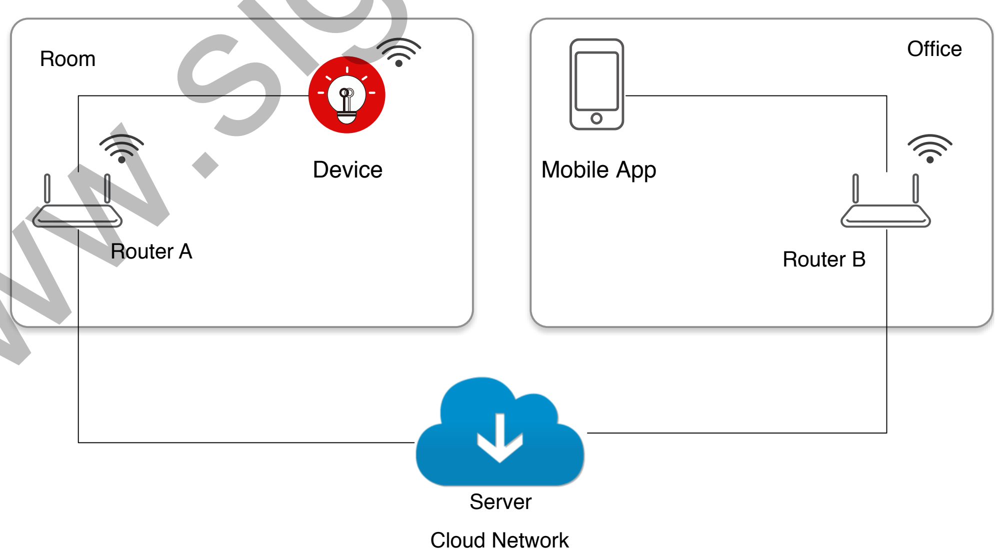


Figure 1-3. Cloud Network

There are three statuses in mobile app.

- **Cloud status**: The device is a cloud device; the device and the IOT app connect to different routers.
- **Online status**: The device is a local device or cloud device; the device and the IOT app connect to the same router.
- **Offline status**: The device is a cloud device, and is not connected to the router.

Device Type and Status

Device status	Cloud status	Online status	Offline status
Cloud device	✓	✓	✓
Local device	✗	✓	✗

1.2. Network Structure

1.2.1. Network Principle

Mesh network supports auto-networking. When ESP-Touch is used to configure the mesh network, the device automatically scans the Wi-Fi AP nearby.

1.2.2. Network Diagram

The mesh network diagram is shown in **Figure 1-4**.

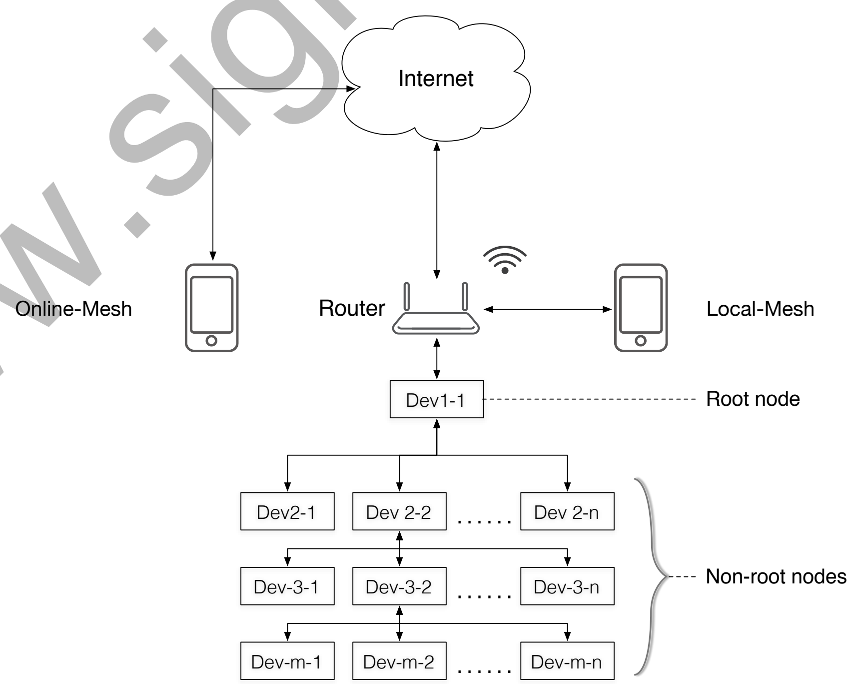


Figure 1-4. Mesh Network Diagram

- The node that directly connects to the router is the root node while others are non-root nodes. For more information, refer to **1.2.3 Network Node**.
- Online-mesh: When the router connects to the internet, you can use the mobile to control the **Cloud Devices** via the Internet.
- Local-mesh: You can only control the **Local Devices** via the router.

1.2.3. Network Node

According to the location in a mesh network, nodes can be divided into:

Root Node

- It receives and sends packets.
- It forwards the packets from server, mobile apps and its child nodes.

Non-root Node

- Non-leaf node: It receives and sends packets, as well as forwards the packets from its parent node and child nodes.
- Leaf node: It only receives and sends packets, and not forwards packets.

2. Mesh Header

2.1. Mesh Header Format

The format of mesh header is shown in **Figure 2-1**.

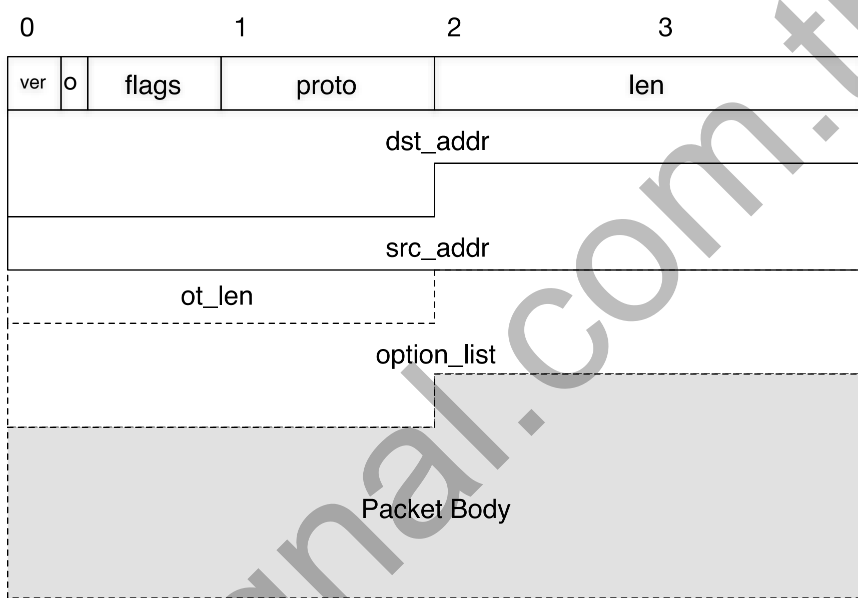
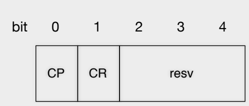
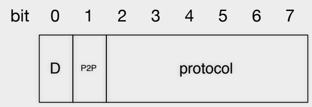
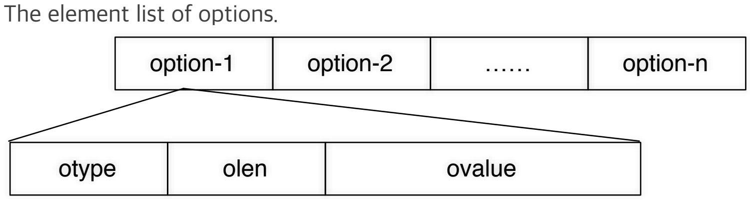


Figure 2-1. Mesh Header Format

The fields of Mesh header are defined in **Table 2-1**.

Table 2-1. Mesh Header Format

Field Name	Length	Description
ver	2 bits	The version of mesh.
o	1 bit	Option flag.
flags	5 bits	
	CP	Piggyback flow permit in packet.
	CR	Piggyback flow request in packet.
	resv	Reserved.

<table>
<tr><th>Field Name</th><th>Length</th><th>Description</th></tr>
<tr><td rowspan="5">proto</td><td>8 bits</td><td>
</td></tr>
<tr><td>D</td><td>The direction of packet:
• 0:downwards
• 1:upwards</td></tr>
<tr><td>P2P</td><td>Node to Node packet.</td></tr>
<tr><td>protocol</td><td>Protocol used by user data.</td></tr>
<tr><td colspan="2">mesh_usr_proto_type is defined as bellow.<pre>
enum mesh_usr_proto_type {
  M_PROTO_NONE = 0,   // used to deliver mesh management packet
  M_PROTO_HTTP,   // user data in HTTP protocol format
  M_PROTO_JSON,   // user data in JSON protocol format
  M_PROTO_MQTT,   // user data in MQTT protocol format
  M_PROTO_BIN,   // user data is binary stream
};
</pre></td></tr>
<tr><td>len</td><td>2 Bytes</td><td>The length of mesh packet in bytes (mesh header included).</td></tr>
<tr><td>dst_addr</td><td>6 Bytes</td><td>Destination address
• proto.D = 0 or proto.P2P = 1 : dst_addr represents the mac address of destination device.
• Bcast or mcast packet: dst_addr represents the bcast or mcast mac address.
• proto.D = 1 and proto.P2P = 0: dst_addr represents the destination IP and port of Mobile or Server.</td></tr>
<tr><td>src_addr</td><td>6 Bytes</td><td>Source address
• proto.P2P = 1: src_addr represents the mac address of source device
• Bcast or mcast packet : src_addr represents the mac address of source device
• proto.D = 1: src_addr represents the mac address of source device
• proto.D = 0 and forward packet into mesh: src_addr represents the IP and port of Mobile or Server</td></tr>
<tr><td>ot_len</td><td></td><td>Represents the total length of options (including itself).</td></tr>
<tr><td>option_list</td><td></td><td>The element list of options.

</td></tr>
<tr><td>otype</td><td>1 Byte</td><td>Option type.</td></tr>
<tr><td>olen</td><td>1 Byte</td><td>The length of current option.</td></tr>
<tr><td>ovlaue</td><td>User defined</td><td>The value of current option.</td></tr>
</table>

2.2. Mesh Option Type

2.2.1. Structure

The mesh option type is defined by the structure of `mesh_option_type`.

```
enum mesh_option_type {
  M_O_CONGEST_REQ = 0,//flow request option
  M_O_CONGEST_RESP,   //flow response option
  M_O_ROUTER_SPREAD,  //router information spread option
  M_O_ROUTE_ADD,      //route table update (node joins mesh) option
  M_O_ROUTE_DEL,      //route table update (node exits mesh) option
  M_O_TOPO_REQ,       //topology request option
  M_O_TOPO_RESP,      //topology response option
  M_O_MCAST_GRP,      //group list of mcast
  M_O_MESH_FRAG,      //mesh management fragmentation option
  M_O_USR_FRAG,       //user data fragmentation
  M_O_USR_OPTION,     //user option
};
```


Table 2-2. Mesh Header Type

Field Name	Length	Description	Format
M_O_CONGEST_REQ	2 Bytes	Used for flow request.	otype olen ovalue 0x00 0x02
M_O_CONGEST_RESP	6 Bytes	Used to respond to flow.	otype olen ovalue 0x01 0x06 congest capacity
M_O_ROUTER_SPREAD	106 Bytes	Used to spread information of router.	otype olen ovalue 0x02 0x6A Router information
M_O_ROUTE_ADD	6*n+2 Bytes	Used to update route table when new node joins mesh network.	otype olen ovalue 0x03 length MAC address list
M_O_ROUTE_DEL	6*n+2 Bytes	Used to update route table when node exits mesh network.	otype olen ovalue 0x04 length MAC address list
M_O_TOPO_REQ	1 Byte	Used to get topology of mesh network.	otype olen ovalue 0x05 0x06 MAC address of the device searched
M_O_TOPO_RESP	6*n+2 Bytes	Used to respond to topology of mesh network.	otype olen ovalue 0x06 length MAC address list

2.2.2. Example

Flow Request Packet

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 01 14 00 18 FE 34 A5 3B AD 18 FE 34 A2 C7 76
00000010h: 04 00 00 02
```

Table 2-3. Flow Request Packet

Field Name	Value	Description
head.ver	00	Current version of mesh is 00.
head.O	1	The option exists in this packet.
head.flags.CP	0	Without piggyback flow permit.
head.flags.CR	0	Without piggyback flow request.
head.flags.resv	000	Reserved.
head.proto.D	1	Upwards.
head.proto.P2P	0	Without node to node packet.
head.proto.protocol	000000	Mesh management packet.
head.len	0x0014	The length of packet is 20 Bytes.
head.dst_addr	18 FE 34 A5 3B AD	MAC address of destination device.
head.src_addr	18 FE 34 A2 C7 76	MAC address of source device.
head.ot_len	0x0004	The option length is 0x0004.
head.option_list[0].otype	0x00	M_CONGEST_REQ.
head.option_list[0].olen	0x02	The option length is 0x02.

Flow Response Packet

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 00 18 00 18 FE 34 A2 C7 76 18 FE 34 A5 3B AD
00000010h: 08 00 01 06 01 00 00 00
```

Table 2-4. Flow Response Packet

Field Name	Value	Description
head.ver	00	Current version of mesh is 00.
head.O	1	The option exists in this packet.
head.flags.CP	0	Without piggyback flow permit.
head.flags.CR	0	Without piggyback flow request.

Field Name	Value	Description
head.flags.resv	000	Reserved.
head.proto.D	0	Downwards.
head.proto.P2P	0	Without node to node packet.
head.proto.protocol	000000	Mesh management packet.
head.len	0x0015	The length of packet is 21 Bytes.
head.dst_addr	18 FE 34 A2 C7 76	MAC address of destination device.
head.src_addr	18 FE 34 A5 3B AD	MAC address of source device.
head.ot_len	0x0008	The option length is 0x0008.
head.option_list[0].otype	0x01	M_CONGEST_RESP.
head.option_list[0].olen	0x06	The option length is 0x06.
head.option_list[0].ovalue	0x01	Option value is 0x00000001, flow capacity is 0x00000001.

3. API Reference

3.1. Data Structure

3.1.1. Mesh Header Format

```
struct mesh_header_format {
    uint8_t ver:2;              // version of mesh
    uint8_t oe: 1;              // option flag
    uint8_t cp: 1;              // piggyback flow permit in packet
    uint8_t cr: 1;              // piggyback flow request in packet
    uint8_t rsv:3;              // reserved;
    struct {
        uint8_t d:  1;          // direction, 1:upwards, 0:downwards
        uint8_t p2p:1;          // node to node packet
        uint8_t protocol:6;     // protocol used by user data;
    } proto;
    uint16_t len;       // packet total length (mesh header included)

    uint8_t dst_addr[ESP_MESH_ADDR_LEN];  // destination address
    uint8_t src_addr[ESP_MESH_ADDR_LEN];  // source address
    struct mesh_header_option_header_type option[0];  // mesh option
} __packed;
```

3.1.2. Mesh Option Header Format

```
struct mesh_header_option_header_type {
  uint16_t ot_len;                            // option total length;
  struct mesh_header_option_format olist[0]; // option list
} __packed;
```

3.1.3. Mesh Option Format

```
struct mesh_header_option_format {
    uint8_t otype;          // option type
    uint8_t olen;           // current option length
```

```
        uint8_t ovalue[0];     // option value
    } __packed;
```

3.1.4. Mesh Option Fragmentation Format

```
    struct mesh_header_option_frag_format {
        uint16_t id;            // identity of fragmentation
        struct {
            uint16_t resv:1;     // reserved
            uint16_t mf:1;       // more fragmentation
            uint16_t idx:14;     // fragmentation offset;
        } offset;
    } __packed;
```

3.1.5. Mesh Callback Format

```
    typedef void (* espconn_mesh_callback)(int8_t result);
```

3.1.6. Mesh Scan Callback Format

```
    typedef void (* espconn_mesh_scan_callback)(void *arg, int8_t status)
```

3.1.7. Mesh Scan User Callback Format

```
    typedef void (* espconn_mesh_usr_callback)(void *arg);
```

3.2. Packet APIs

Note:

For the packet APIs, refer to **2C-ESP8266_SDK_Programming Guide**.

4. Sample Code

4.1. Device

For the details of the codes, refer to **mesh_demo.c**.

4.2. Mobile or Server

```
void controller_entrance(Parameter list)
{
    /*Add your codes to check status*/
    /*Add your codes to build control packet*/
    uint8_t json_control_data[] = {/*Add your codes*/};
    uint16_t control_data_len = sizeof(json_control_data)
    struct mesh_header_format *mesh_header = NULL;
    /* src_addr should be the combination of IP and port of
    Mobile or Server. You can set the address to zero, then the
    root device will fill in the section. If you fill in the
    section by yourself, please make sure the value is right.*/
    uint8_t src_addr[] = {0,0,0,0,0,0},
    dst_addr[] = {xx,xx,xx,xx,xx,xx};
    mesh_header = (struct mesh_header_format
*)espconn_mesh_create_packet(dst_addr, src_addr, false, true,
M_PROTO_JSON, control_data_len,
    false, 0, false, 0, false, 0, 0);
    if (!mesh_header)
    {
        printf("alloc resp packet fail\n");
        return;
    }
    if (espconn_mesh_set_usr_data(mesh_header,
resp_json_packet_body, resp_data_len))
        {
        printf("set user data fail\n");
        free(mesh_header);
```

```
                return;
            }
            // sent control packet
            espconn_mesh_sent(esp, mesh_header, mesh_header->len);
            free(mesh_header);
        }
```

4.3. Getting Topology

```
void topology_entrance(Parameter list)
{
    /*Add your codes to check status*/
    /*Add your codes to build getting topology packet*/
    bool res;
    struct mesh_header_format *mesh_header = NULL;
    struct mesh_header_option_format *topo_option = NULL;
    uint8_t src_addr[] = {0,0,0,0,0,0};
    uint8_t dst_addr[] = {xx,xx,xx,xx,xx,xx}; // mac address of root 
device
    uint8_t dev_mac[6] = {xx,xx,xx,xx,xx,xx}; // zero represents 
topology of all devices
    uint16_t ot_len = sizeof(*topo_option) + sizeof(struct 
mesh_header_option_header_type) + sizeof(dev_mac);
    mesh_header = (struct mesh_header_format 
*)espconn_mesh_create_packet(
    dst_addr, src_addr, false, true, M_PROTO_NONE, 0,
    true, ot_len, false, 0, false, 0, 0);
    if (!mesh_header) {
        printf("alloc resp packet fail\n");
        return;
    }
    topo_option = (struct mesh_header_option_format 
*)espconn_mesh_create_option(
    M_O_TOPO_REQ, dev_mac, sizeof(dev_mac));
    if (!topo_option) {
```

```
            printf("alloc topo option fail\n");
            free(mesh_header);
            return;
        }
        res = espconn_mesh_add_option(mesh_header, topo_option);
        free(topo_option);
        if (res) {
            printf("add topo option fail\n");
            free(mesh_header);
            return;
        }
        // send packet of getting topology
        espconn_mesh_sent(esp, mesh_header, mesh_header->len);
        free(mesh_header);
    }
```

4.4. Parsing Topology Response

```
    void topology_parser_entrance(uint8_t *topo_resp, uint16_t len)
    {
        /*Add your codes to check parameter*/
        uint16_t oidx = 1;
        struct mesh_header_format *mesh_header = NULL;
        struct mesh_header_option_format *topo_option = NULL;
        mesh_header = (struct mesh_header_format *)topo_resp;
        if (!mesh_header->oe) {
            printf("no option exist\n");
            return;
        }
        /* you need parse all the options one by one in the packet header
        */
        while(espconn_mesh_get_option(mesh_header, M_O_TOPO_RESP,
        oidx++, &topo_option)) {
            uint16_t dev_count = topo_option->olen/6;
```

```
                process_dev_list(topo_option->ovalue, dev_count);
            }
        }
```

4.5. Dev-App

For details of the example codes, refer to:

- **user_config.h**
- **mesh_demo.c**

Espressif System

IOT Team

http://bbs.espressif.com