ESP8266
Mesh User Guide

Version 1.2

About This Guide

This document introduces users to the ESP8266 mesh network, including the following topics:

Chapter	Title	Subject
Chapter 1	Overview	Provides an overview of the ESP-Mesh, including some concepts and the network structure.
Chapter 2	Mesh Header	Introduces the mesh header format and gives 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	Chapter 3 added.
2016.01	V1.2	Chapter 2 and Chapter 4 added, Chapter 1 and Chapter 3 updated.

📖 ***Note:***

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

Table of Contents

1. Overview 1
   1.1. Concepts 1
   1.2. Network Structure 3
      1.2.1. Networking Principle 3
      1.2.2. Networking Diagram 3
      1.2.3. Network Node 4
2. Mesh Header 5
   2.1. Mesh Header Format 5
   2.2. Mesh Option 7
      2.2.1. Structure 7
      2.2.2. Example 8
3. API Reference 10
   3.1. Data Structure 10
      3.1.1. Mesh Header Format 10
      3.1.2. Mesh Option Header Format 10
      3.1.3. Mesh Option Format 10
      3.1.4. Mesh Option Fragmentation Format 11
      3.1.5. Mesh Callback Format 11
      3.1.6. Mesh Scan Callback Format 11
      3.1.7. Mesh Scan User Callback Format 11
   3.2. Packet APIs 11
4. Sample Code 12
   4.1. Device 12
   4.2. Mobile or Server 12
   4.3. Getting Topology 13
   4.4. Parsing Topology Response 14
   4.5. Dev-App 14

1. Overview

The development of the Internet of Things (IoT) requires an increasing number of nodes to connect to the internet. However, only a limited number of nodes (usually fewer than 32) can directly connect to the same router. There are two solutions currently available for this problem.

- Super router: the higher capacity router allows more nodes to directly connect to it.
- Mesh network: the nodes can establish a network and forward packets.

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.

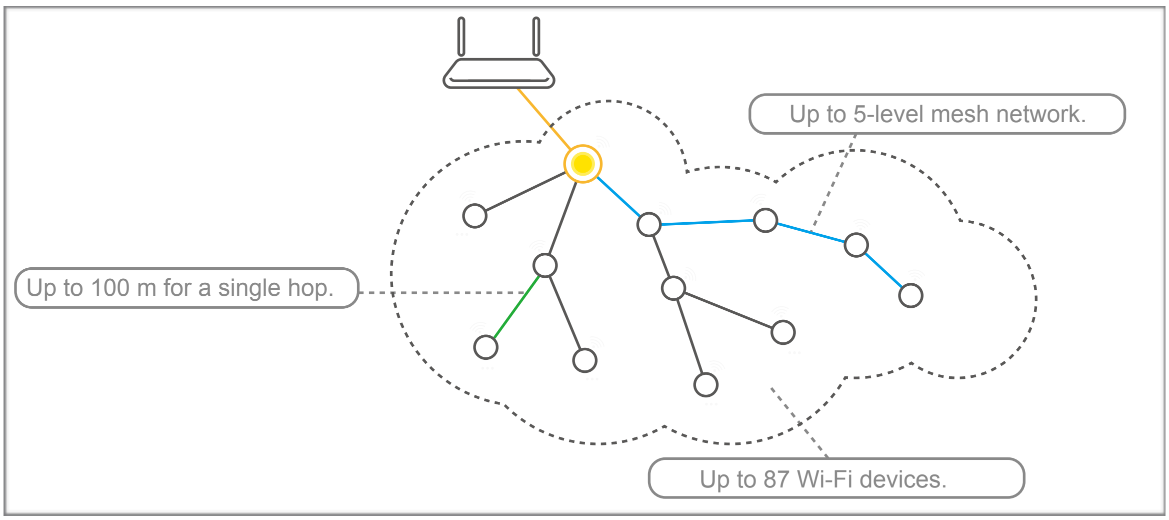


Figure 1-1. ESP-Mesh Network

1.1. Concepts

IOT Espressif App

The IOT Espressif app (hereinafter referred to as IOT app) is a mobile application developed by Espressif. It can realize the local and remote control of Wi-Fi devices, including smart lights and smart plugs.

ESP-TOUCH

ESP-TOUCH is a technology developed by Espressif, which connects Wi-Fi devices to the router.

SmartConfig Mode for ESP-Touch

Users can configure Wi-Fi devices by ESP-TOUCH only when the devices are in SmartConfig Mode. This status is called ESP-TOUCH status. For details on the configuration, please see ***1.2. Network Structure***.

Local Device

As shown in Figure 1-2, if users configure a device to connect to the router via ESP-TOUCH but do not activate it on the server-side, then the device is a local device.

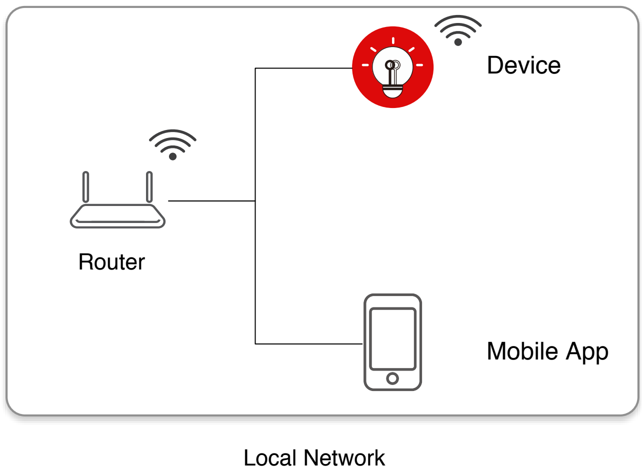


Figure 1-2. Local Network

Cloud Device

As shown in Figure 1-3, if users configure a device to connect to the router via ESP-TOUCH and activate it on the server-side, then the device is a cloud device.

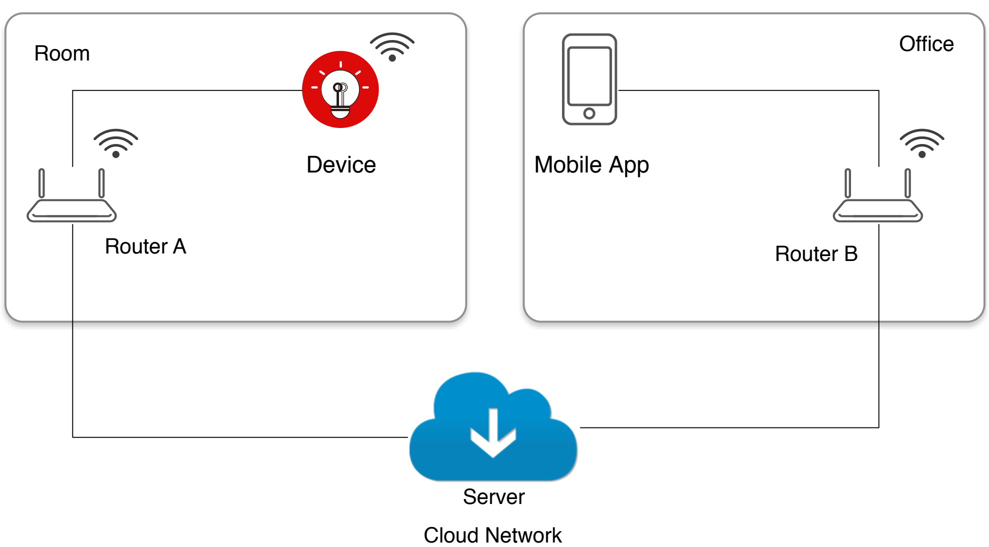


Figure 1-3. Cloud Network

There are three different statuses for a device on the IOT app.

- Cloud status: The device is a cloud device that connects to a different router with the IOT app.
- Online status: The device is a local device or cloud device; both the device and the IOT app connect to the same router.
- Offline status: The device is a cloud device that does not connect 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. Networking Principle

Mesh network supports auto-networking. When users set up a mesh network via ESP-TOUCH, the device automatically scans the Wi-Fi access points (APs) nearby.

1.2.2. Networking Diagram

Figure 1-4 shows the mesh network diagram.

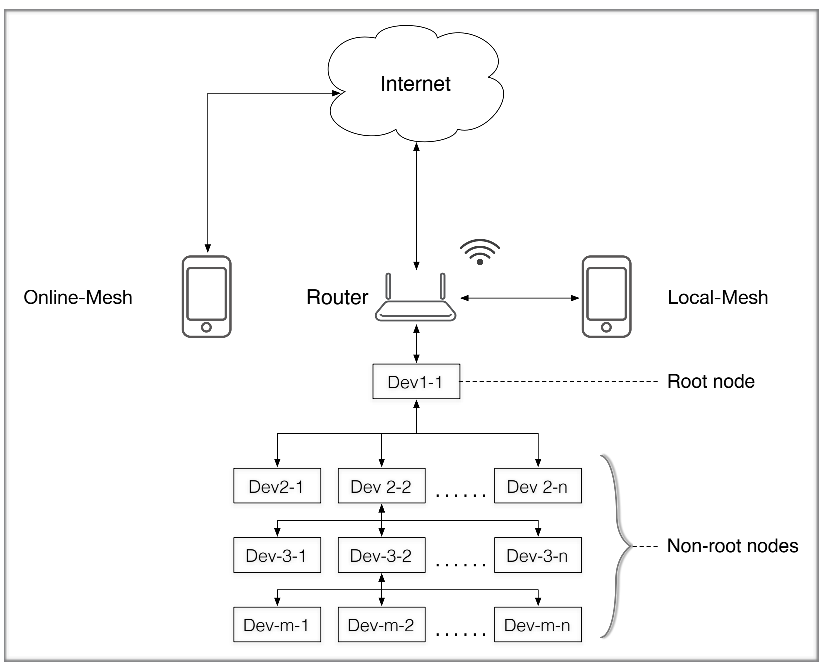


Figure 1-4. Mesh Network Diagram

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

1.2.3. Network Node

According to the location in a mesh network, a node can be:

A Root Node

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

Or,

A 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, but does not forward packets.

2. Mesh Header

2.1. Mesh Header Format

Figure 2-1 shows the mesh header format.

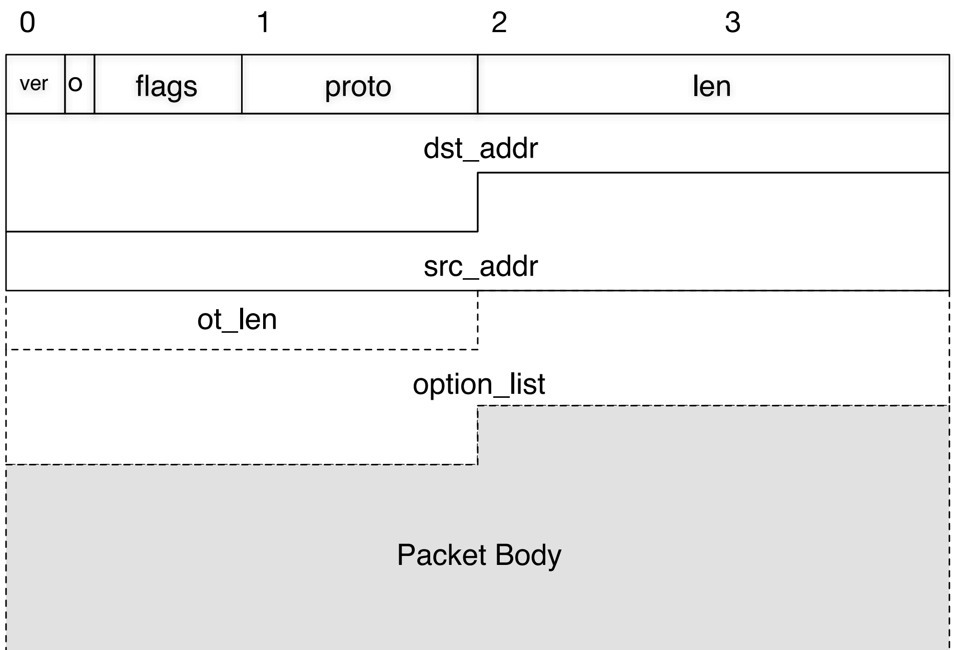
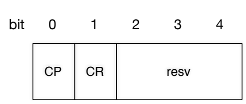
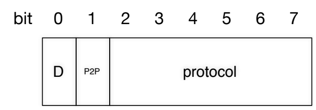
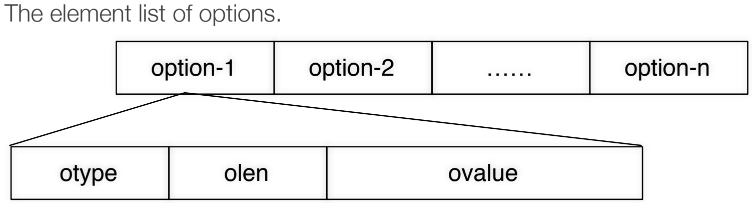


Figure 2-1. Mesh Header Format

Table 2-1 provides the definitions of the mesh header fields.

Table 2-1. Mesh Header Format

Field Name	Length	Description
ver	2 bits	Mesh version.
o	1 bit	Option flag.
flags	5 bits	
	FP	Piggyback flow permit in packet.
	FR	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><code>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
};</code></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 the 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>
</table>

Field Name	Length	Description
olen	1 Byte	The length of current option.
ovlaue	User defined	The value of current option.

2.2. Mesh Option

2.2.1. Structure

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

```
enum mesh_option_type {
  M_O_FLOW_REQ = 0,//flow request option
  M_O_FLOW_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_FLOW_REQ	2 Bytes	Used for flow request.	otype: 0x00, olen: 0x02, ovalue
M_O_FLOW_RESP	6 Bytes	Used to respond to flow.	otype: 0x01, olen: 0x06, ovalue: congest capacity
M_O_ROUTER_SPREAD	106 Bytes	Used to spread information of the router.	otype: 0x02, olen: 0x6A, ovalue: Router information
M_O_ROUTE_ADD	6*n+2 Bytes	Used to update route table when a new node joins the mesh network.	otype: 0x03, olen: length, ovalue: MAC address list
M_O_ROUTE_DEL	6*n+2 Bytes	Used to update the route table when a node exits mesh network.	otype: 0x04, olen: length, ovalue: MAC address list

Field Name	Length	Description	Format
M_O_TOPO_REQ	8 Bytes	Used to get the topology of the mesh network.	otype: 0x05; olen: 0x06; ovalue: MAC address of the device searched
M_O_TOPO_RESP	6*n+2 Bytes	Used to respond to the topology of the mesh network.	otype: 0x06; olen: length; ovalue: 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.FP	0	Without piggyback flow permit.
head.flags.FR	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_FLOW_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.FP	0	Without piggyback flow permit.
head.flags.FR	0	Without piggyback flow request.
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_FLOW_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 fp: 1;            // piggyback flow permit in packet
    uint8_t fr: 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 more information about the packet APIs, please refer to ESP8266 Non-OS SDK API Guide.*

4. Sample Code

4.1. Device

For details, please refer to:
ESP8266_MESH_DEMO/blob/master/mesh_demo/demo/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 on the sample codes, please refer to:

- *ESP8266_MESH_DEMO/blob/master/mesh_demo/include/user_config.h*
- *ESP8266_MESH_DEMO/blob/master/mesh_demo/demo/mesh_demo.c*