ESPRESSIF WIFI MESH

1. Introduction

随着物联网的发展，物联网节点规模迅速扩张。但router可供接入的节点有限（通常小于20），因此，在大规模的物联网应用中，不可能把所有物联网节点都直接接入router。而解决这一问题的途径有两种：

1) 超级router：增强router的功能，使其可以接入更多的节点；

2) Mesh组网：物联网节点之间可以相互组成网络，并可以转发数据；

Mesh组网方案不要求对现有的router功能进行任何修改，就可以支持大量的IOT node接入互联网（目前，我们已经实现了在一个普通的家用router下挂载一个网络规模为80个节点的Espressif mesh网络）。

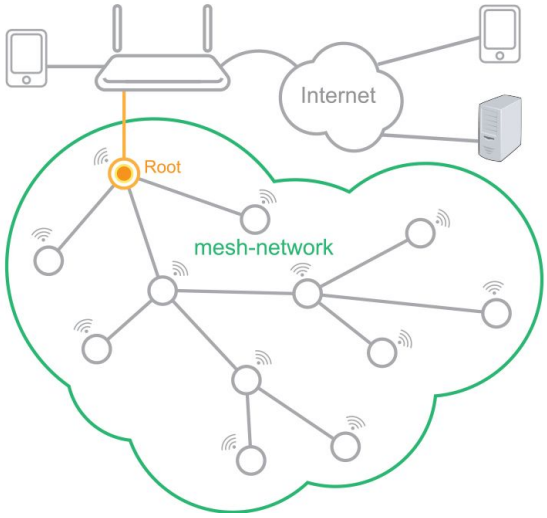


图1-1 Mesh网络示意图

Espressif的Mesh组网采用多跳的树形结构，如图1-1所示，网络中有两种类型的节点：root节点（黄色节点）和non-root节点。root节点负责把发往Server或Mobile-App的数据包转发给Server或Mobile-App，同时把来自Server或Mobile-App的数据包转发到mesh网络。而non-root节点包括mesh-non-leaf节点和mesh-leaf节点，mesh-non-leaf节点负责转发来自其子节点或父节点的数据包；而mesh-leaf节点只接收或发送数据包，不做数据包的转发。Mesh网络有两种工作模式：Online-mesh和Local-mesh。Online-mesh网络中的节点都支持互联网远程控制或不使用互联网的local控制，而Local-mesh网络中的节点只支持local控制，

不支持互联网远程控制。而mesh组网方式分为自动组网模式和用户配置模式。Mesh自动组网模式只需要用户配置root节点，告知其需要连接的router信息，而其他节点会自动扫描WIFI-AP，择优加入mesh网络。而用户配置mesh组网方式，需要用户为每一个mesh节点配置父节点。默认采用mesh自动组网方式。

2. Mesh 的拓扑结构

Espressif的Mesh网络的树形拓扑结构如图2-1所示。默认情况下，整个Mesh网络最多允许5跳，每一个mesh-non-leaf节点最多可以允许4个直接子节点接入网络，所以一个mesh网络最多支持341（$4^0+4^1+4^2+4^3+4^4$）个节点接入网络。每个节点都工作在STA+SoftAP模式，STA和SoftAP的IP地址对应关系如表2-1所示。

表 2-1 STA-IP 和 SoftAP-IP 的对应关系

Sta-IP	192.168.1.*	2.255.255.*	3.255.255.*	4.255.255.*	5.255.255.*
SoftAP-IP	2.255.255.1	3.255.255.1	4.255.255.1	5.255.255.1	6.255.255.1

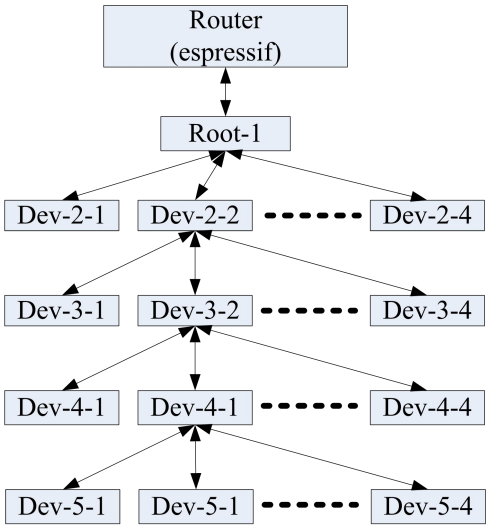


图 2-1 Mesh 网络的树形拓扑结构

3. Mesh protocol architecture

3.1 Mesh packet package in protocol architecture

There are three schemes to implement mesh as follows:

1) Mesh between application layer and TCP layer
2) Mesh between IP layer and MAC layer
3) Mesh between IP layer and MAC layer using espnow

3.1.1 Mesh between application layer and TCP layer

目前，mesh 的实现介于 application 和 TCP 之间，如图 3-1 所示，该方案的优势和不足如下：

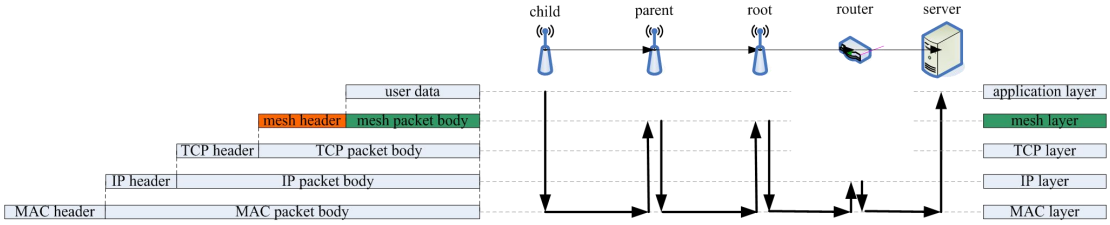


图 3-1. mesh between application layer and TCP layer

3.1.1.1 Advantage：

1) 不需要 NAT，整个 mesh 网络对于外部的 server 或控制器而言就如一个节点；
2) 整个 mesh 网络与 server 只有一个 TCP 连接；
3) Mesh 层不需要处理任何其它层的信息，层次结构十分清晰；
4) Mesh 层收到 mesh packet 后，直接根据 mesh header 来决定是接收还是转发，而转发过程不需要对 packet 做任何处理，直接转发即可；

3.1.1.2 Issue：

1) Mesh 基于 TCP 实现，因此只支持基于 TCP 的应用协议，无法支持 UDP；
2) Server 或控制器端要实现一个对等的 mesh 层；

3.1.2 Mesh between MAC layer and IP layer

将来，mesh 的实现可能会移到 IP 和 MAC 之间，如图 3-2 所示，该方案的优势和不足如下：

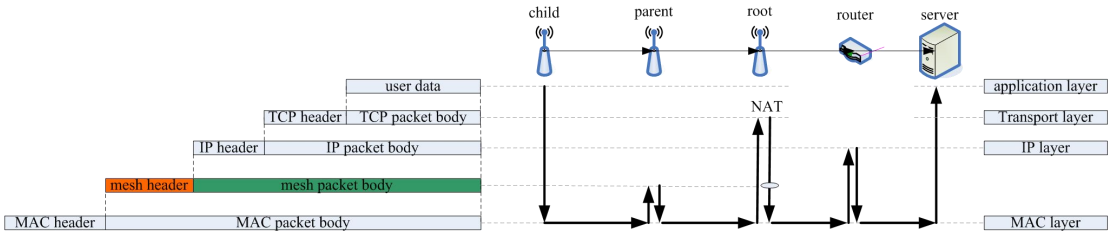


图 3-2. mesh between IP layer and MAC layer

3.1.2.1 Advantage：

1) 改方案与传统的基于 NAT 的方案相似，便于理解；
2) Server 或控制器端实现相对简单，mesh 对它们透明；
3) 由于 mesh 位于 IP 层以下，因此 mesh 可以支持任何基于 IP 的网络协议；

3.1.2.2 Issue：

1) Root 节点要具备双协议栈，即 mesh 协议栈和 TCP/IP 协议栈；
2) Root 节点要实现类似 NAT 的功能；
3) Root 节点收到 mesh packet 后，根据 mesh header 决定如果转发 packet，如果 packet 要转发 mesh 网络外部，其转发之前要根据 mesh header 更新 IP header，如图 3-3，同时去除 mesh header；

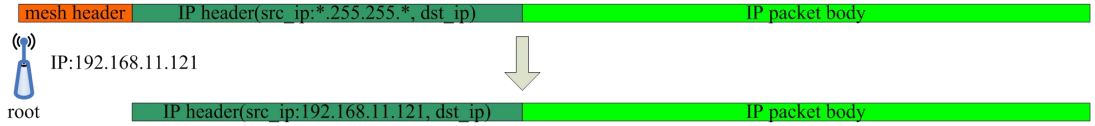


图 3-3. Mesh packet 的修改（forward out of mesh）

4) Root 节点收到来自外部发往 mesh 内部的 packet，在转发到 mesh 网络前要添加 mesh header，如图 3-4 所示；

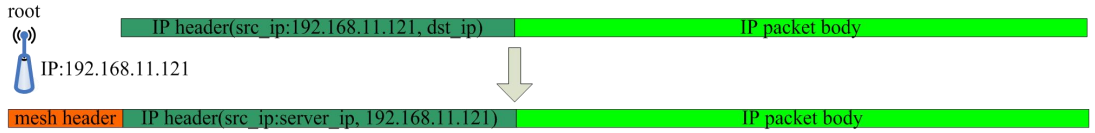


图 3-4. Mesh packet 的修改（forward into mesh）

5) 节点收到别人发给自己的 mesh 后，在传给 IP 处理之前，需要修改 IP 层的目的 IP 地址，如图 3-5 所示；

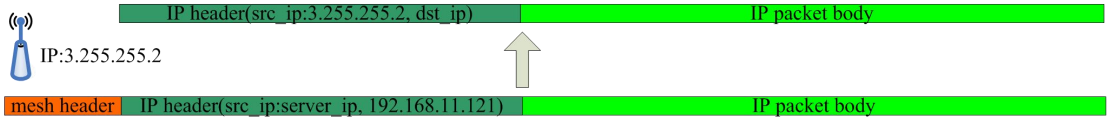


图 3-5. IP 层的修改（delivery to upper layer）

6) 整个网络建立时由 Root 节点分配与外部通信的端口号，并且不同节点一定要分配不同的端口，并且 Root 节点要保持端口与节点 MAC 地址的对应关系表，可以在 MAC 路由表再添加一个字段保存节点的端口，这个和 ZigBee 相似，也和 6LowPAN 的短标识方案相同，只不过 ZigBee/6LowPA 分配的是节点标识 ID，该 ID 用 2 个字节表示；
7) 每个节点都与 server 保持一个连接，server 需要增加处理能力；

3.1.3. Mesh between MAC layer and IP layer (espnow)

将来，mesh 可能直接借助于 espnow 来实现，如图 3-6 所示，该方案的优势和不足如下：

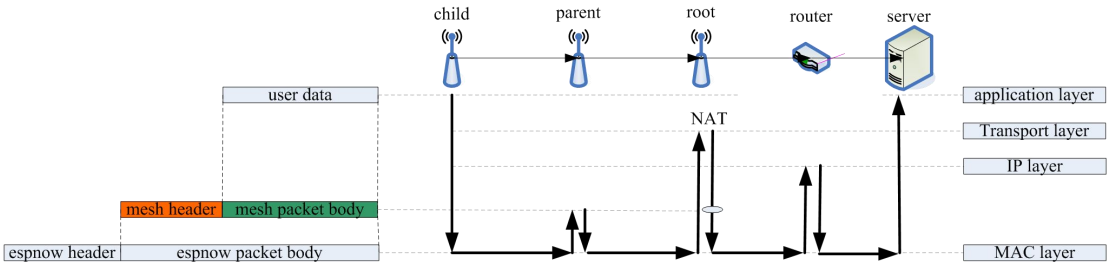


图 3-6. mesh between IP layer and MAC layer using espnow

3.1.3.1 Advantage：

1) 改方案的传输效率比较高；
2) Server 或控制器端实现相对简单，mesh 对它们透明；
3) 由于 mesh 位于 IP 层以下，因此 mesh 可以支持任何基于 IP 的网络协议；

3.1.3.2 Issue：

1) 每个节点都要具备双协议栈，即 mesh 协议栈和 TCP/IP 协议栈；
2) Root 节点要实现类似 NAT 的功能；
3) Root 节点收到 mesh packet 后，根据 mesh header 决定如果转发 packet，如果 packet 要转发 mesh 网络外部，其转发之前要根据 mesh header 生成所有上层协议的 header 如图 3-7，同时去除 mesh header；关于上层协议的 header 的构建可以考虑使用 mesh header 中的 option 来存储；

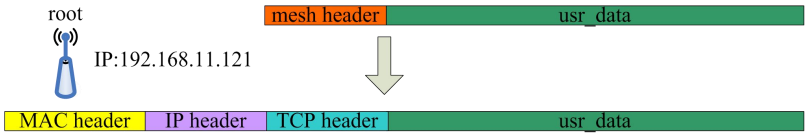


图 3-7. Mesh packet 的修改（forward out of mesh）

4) Root 节点收到来自外部发往 mesh 内部的 packet，在转发到 mesh 网络前要要根据上层协议的 header 构建 mesh header，如图 3-8 所示；

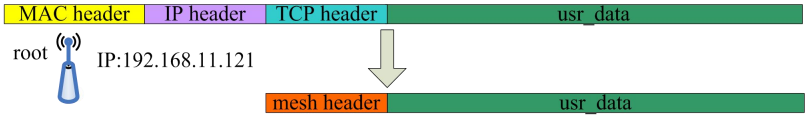


图 3-8. Mesh packet 的修改（forward into mesh）

5) 节点收到别人发给自己的 mesh 后，在传给上层协议处理之前，需要根据 mesh header 生成所有上层协议的 header，如图 3-9 所示；

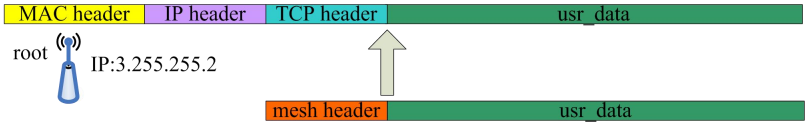


图 3-9. Mesh packet 的修改（delivery to upper layer）

6) 整个网络建立时由 Root 节点分配与外部通信的端口号，并且不同节点一定要分配不同的端口，并且 Root 节点要保持端口与节点 MAC 地址的对应关系表，可以在 MAC 路由表再添加一个字段保存节点的端口，这个和 ZigBee 相似，也和 6LowPAN 的短标识方案相同，只不过 ZigBee/6LowPA 分配的是节点标识 ID，该 ID 用 2 个字节表示；
7) 每个节点都与 server 保持一个连接，server 需要增加处理能力；

3.2 Format of mesh header

目前，mesh header 的 draft 已经演进到 draft-3，draft-2 和 draft-1 已经废弃，留在章节内仅供描述 draft of mesh header 的演进过程。

3.2.1 Mesh header with Draft-3

Draft-3 相对于 Draft-2，修改了 O(congest)、flags 和 proto 的顺序，另外 flags 字段由原来的 7bits（原来的 flags 中的 6bits 只用了 1bit）改成了 5bits（使用 2bits），proto 字段由原来的 6bits 改成了 8bits，并且修改了 proto 字段的定义；同时，仍然保持字节对齐。Draft-3 的 mesh header 如图 3-10 所示；

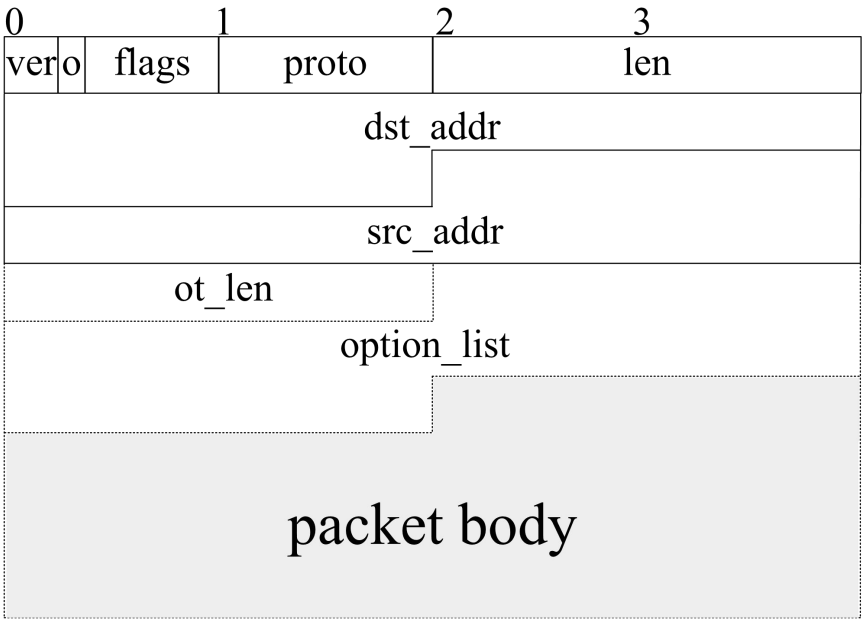
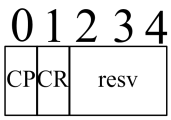


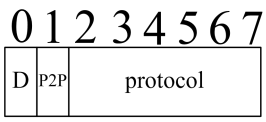
图 3-10. Mesh header (Draft-3)

3.2.1.1 Description：

- ver: 2 bits, version of mesh;
- ***0***: 1 bit, flag of options in mesh header.
- ***flags***:5 bits,



  - CP: piggyback flow permit in packet
  - CR: piggyback flow request in packet
  - resv: reserve for future.
- proto: 8 bits



  - D: direction of packet (0:downwards, 1:upwards)
    - ✓ 0: downwards
    - ✓ 1: upwards

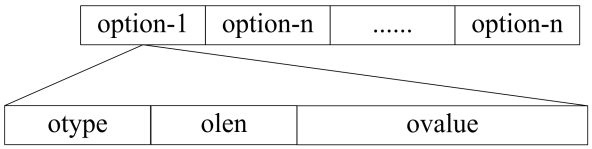
- P2P: Node to Node packet
- protocol:. protocol used by user data

The current protocol type is as follows:

```
enum mesh_usr_proto_type {
    M_PROTO_NONE = 0,       // used to delivery mesh management packet
    M_PROTO_HTTP,           // user data formated with HTTP protocol
    M_PROTO_JSON,           // user data formated with JSON protocol
    M_PROTO_MQTT,           // user data formated with MQTT protocol
    M_PROTO_BIN,            // user data is binary stream
};
```

- ***len: 2 Bytes, length of mesh packet in bytes(include mesh header)***
- ***dst_addr***: 6 Bytes, destination address
  - proto.D = 0 (downwards) or proto.P2P = 1 (Node-to-Node packet)
    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 (upwards) and proto.P2P = 0
    dst_addr represents the destination IP and port of Mobile or Server
- ***src_addr***: 6 Bytes, 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 (upwards)
    src_addr represents the mac address of source device
  - proto.D = 0 (downwards) and forward packet into mesh
    src_addr represents the IP and port of Mobile or Server
- ***Options:***

  ot_len: represent the total length of options (include ot_len field)
  option_list: the element list of the options



- ***Format of option element:***
  - otype: 1 Byte, option type
  - olen: 1 Byte, the length of current option
  - ovlaue: the value of current option

3.2.1.2 Option instances：

目前，mesh 所支持的 option type 如下：

```
enum mesh_option_type {
    M_O_CONGEST_REQ = 0,     // congest request option
    M_O_CONGEST_RESP,        // congest 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 leaves 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 fragment option
    M_O_USR_FRAG,            // user data fragment
    M_O_USR_OPTION,          // user option
};
```

1) M_O_CONGEST_REQ: used to request for congest

- **flow request option format:**

  flow request option has no option value as follow, so length of this option is 2 Bytes.

otype	olen	ovalue
0x00	0x02	

- **Instance of 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                                     ; ..
```

  - head.ver (00): current version of mesh is 00
  - head.O (1): option exist in this packet
  - head.flags.CP (0): not piggyback flow permit
  - head.flags.CR (0): not piggyback flow request
  - head.flags.resv(000): reserved
  - head.proto.D (1): upwards
  - head.proto.P2P (0): not 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): option length is 0x0004
  - head.option_list[0].otype (0x00): M_CONGEST_REQ
  - head.option_list[0].olen(0x02): option length is 0x02

2) M_O_CONGEST_RESP: used to give response for congest

- **flow response option format:**

  Size of flow response option value is 4 Byte, so the length of this option is 6 Bytes. The option value represents the capacity of congest.

otype	olen	ovalue
0x01	0x06	0x00000001

- **Instance of flow request 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
```

  - head.ver (00): current version of mesh is 00

- head.O (1): option exist in this packet
- head.flags.CP (0): not piggyback flow permit
- head.flags.CR (0): not piggyback flow request
- head.flags.resv(000): reserve
- head.proto.D (0): downwards
- head.proto.P2P (0): not node to node packet
- head.proto.protocol(000000): mesh management packet
- head.len(0x0018): the length of packet is 24 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): option length is 0x0008
- head.option_list[0].otype (0x01): M_CONGEST_REQ
- head.option_list[0].olen(0x06): option length is 0x06
- head.option_list[0].ovalue(0x000000001): option value is 0x01, the capacity of flow is 1.

3) M_O_ROUTER_SPREAD: used to spread information of router

- **Router spread option format:**

Router spread option is used to spread router information all over the mesh network. When new mesh node joins mesh network, its parent will sent mesh packet with router spread option to it. Router information is formatted with struct station_config, so the size of router spread option value is 104 Byte, and the length of this option is 106 Bytes. The option value represents the information of router.

otype	olen	ovalue
0x02	0x6A	router information

4) M_O_ROUTE_ADD: used to update route table when new node joins mesh network

- **Route add option format:**

This option is used to update mac route table. When mesh node joins mesh network, it uses the option to package its current mac route table, then sent packet with this option to its parent. When parent node receive the packet, it check its mac route table with the option, if it finds mac route need to been updated, it will delivery the packet to its parent.

otype	olen	ovalue
0x03	length	mac address list

5) M_O_ROUTE_DEL: used to update route table when node leave mesh network

- **Route delete option format:**

This option is also used to update mac route table. When mesh node detect its child node exits mesh network, it uses the option to package mac route of the child and delete its mac route table, then sent packet with this option to its parent. When parent node receive the packet, it check its mac route table and delete mac element according to this option. Finally, it deliveries this packet to its parent.

otype	olen	ovalue
0x04	length	mac address list

6) M_O_TOPO_REQ: used to get topology of mesh network.

- **Topology request option format:**

  Mobile or service can use this option to get topology of all node from root device.

otype	olen	ovalue
0x05	0x08	mac address of device searched

  - **Request topology of one device**

    The ovalue is the mac address of the device.

  - **Request topology of all the devices**

    The ovalue is broadcast mac address.

- **Instance of Request topology of one device (18FE34A53BAD) from root (18FE34A2C776)**

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 00 1A 00 18 FE 34 A2 C7 76 00 00 00 00 00 00 ; .....
00000010h: 0A 00 05 08 18 FE 34 A5 3B AD                   ; .....
```

- **Instance of Request topology of all devices from root (18FE34A2C776)**

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 00 1A 00 18 FE 34 A2 C7 76 00 00 00 00 00 00 ; .....
00000010h: 0A 00 05 08 00 00 00 00 00 00                   ; .....
```

7) M_O_TOPO_RESP: used to response topology of mesh network.

- **Topology response option format:**

  Root device use this option to package its mac route and sent packet with this option to controller (mobile or server).

otype	olen	ovalue
0x06	length	mac address list

  - **Topology response of one device**

    The ovalue is the mac address of the device.

  - **Topology response of all the devices**

    The ovalue is mac address list.

- **Instance of topology response**

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 00 20 00 C0 A8 0B 19 58 1B 18 FE 34 A2 C7 76 ; .. .X...4..v
00000010h: 10 00 05 0E 18 FE 34 A5 3B AD 18 FE 34 A5 2B C7 ; ......4.;...4.+.
```

  - **Description:**

    IP of controller: 192.168.11.25, port: 7000 (0x1B58)

    Mac of root: {0x18, 0xFE, 0x34, 0xA2, 0xC7, 0x76}

    Tow sub nodes: {0x18, 0xFE, 0x34, 0xA5, 0x3B, 0xAD},

    {0x18, 0xFE, 0x34, 0xA5, 0x2B, 0xC7}

4. API

3.1 Data structure

3.1.1 Mesh header format

```
117 struct mesh_header_format {
118     uint8_t ver:2;          // version of mesh
119     uint8_t oe: 1;          // option exist flag
120     uint8_t cp: 1;          // piggyback congest permit in packet
121     uint8_t cr: 1;          // piggyback congest request in packet
122     uint8_t rsv:3;          // reserve for fulture;
123     struct {
124         uint8_t d:  1;      // direction, 1:upwards, 0:downwards
125         uint8_t p2p:1;      // node to node packet
126         uint8_t protocol:6; // protocol used by user data;
127     } proto;
128     uint16_t len;           // packet total length (include mesh header)
129     uint8_t dst_addr[ESP_MESH_ADDR_LEN];  // destiny address
130     uint8_t src_addr[ESP_MESH_ADDR_LEN];  // source address
131     struct mesh_header_option_header_type option[0];  // mesh option
132 } __packed;
```

3.1.2 Mesh option header format

```
103 struct mesh_header_option_header_type {
104     uint16_t ot_len;        // option total length;
105     struct mesh_header_option_format olist[0];   // option list
106 } __packed;
107
```

3.1.3 Mesh option format

```
97 struct mesh_header_option_format {
98     uint8_t otype;          // option type
99     uint8_t olen;           // current option length
100    uint8_t ovalue[0];      // option value
101 } __packed;
```

3.1.4 Mesh option fragment format

```
108 struct mesh_header_option_frag_format {
109     uint16_t id;            // identify of fragment
110     struct {
111         uint16_t resv:1;    // reserve for future
112         uint16_t mf:1;      // more fragment
113         uint16_t idx:14;    // fragment offset;
114     } offset;
115 } __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 API:

3.2.1 espconn_mesh_create_packet

- Declaration:

  void espconn_mesh_create_packet(uint8_t *dst_addr, uint8_t *src_addr, bool p2p,
  bool piggyback_cr, enum mesh_usr_proto_type proto,
  uint16_t data_len, bool option uint16_t ot_len,
  bool frag, enum mesh_option_type frag_type, bool mf,
  uint16_t frag_idx, uint16_t frag_id);

- Description:

  The function is used to create mesh packet.

- Parameters:
  - dst_addr: destination address (6 Bytes)

  If the destination of packet is server or mobile, the dst_addr is the combination of IP address and port of server or mobile.

  For instance:

  memcpy(dst_addr, remote_ip, 4);

  memcpy(dst_addr + 4, &remote_port, 2);

  If the destination of packet is node, the dst_addr is the mac address of destination device.
  - src_addr: source address (6 Bytes)

  If mobile or server try to sent packet to device, mobile or server needs to fill the src_addr with combination of its IP address and port.

  Mobile or server can fill src_addr with default value which is zero.

  For instance:

  memset(src_addr, 0, 6);

  If the packet is produced by device, device need to fill src_addr with its mac address.
  - p2p: node-to-node packet
  - piggyback_cr: piggyback flow request.

  Device and mobile should set the piggyback_cr.
  - proto: protocol used by user data
  - data_len: length of user data.
  - Option: option flag
  - ot_len: option total length, include ot_len in mesh header.

- frag: fragment flag

  if the packet is too long to be packaged in one packet, user can use more than one mesh fragment to delivery the long packet.
- frag_type: fragment type
- mf: more fragment
- frag_idx: the index of fragment
- frag_id: the identify of fragment

■ Return value:
- NULL: create mesh packet fail.
- addr: the start address of mesh packet.

■ Instance:

```
uint8_t usr_data [800] = {*********};
Uint8_t src_adr[6] = {******}, dst_addr[6] = {******};
struct mesh_header_format *pkt_header = NULL;

pkt_header = (struct mesh_header_format *)espconn_mesh_create_packet(
                dst_addr, src_addr, false, true, M_PROTO_JSON, sizeof(usr_data),
                false, 0, false, 0, false, 0, 0);
if (!pkt_header) {
    printf("alloc packet fail\n");
    return;
}
```

3.2.2 espconn_mesh_create_option

■ Declaration:

```
void espocnn_mesh_create_option(uint8_t otype, uint8_t *ovalue, uint8_t val_len);
```

■ Description:

The function is used to create mesh option.

■ Parameters:
- otype: option type
- ovalue: option value
- val_len: option data value length

■ Return value:
- NULL: create mesh packet fail.
- addr: the start address of mesh option.

■ Instance:

```
uint8_t usr_op_val[8] = {*********};
uint8_t olen = sizeof(usr_op_val);
struct mesh_header_option_format *option = NULL;

option = (struct mesh_header_option_format *)espconn_mesh_create_option(
        M_O_USR_OPTION, usr_op_val, olen);

if (!option) {
```

```
        printf("alloc option fail\n");
        return;
    }
```

3.2.3 espconn_mesh_add_option

- Declaration:
  bool espconn_mesh_add_option(struct mesh_header_format *head,
  struct mesh_header_option_format *option)
- Description:
  The function is used to add mesh option in mesh packet.
- Parameters:
  - head: mesh packet header
  - option: option header
- Return value:
  - true: add success
  - false: add failure, please check the header->len, header->oe and header->option->ot_len.

3.2.3 espconn_mesh_get_option

- Declaration:
  bool espconn_mesh_add_option(struct mesh_header_format *head,
  uint8_t otype, uint16_t oidx,
  struct mesh_header_option_format **option)
- Description:
  The function is used to get mesh option in mesh packet.
- Parameters:
  - head: mesh packet header
  - otype: option type
  - oidx: the option index with the same otype
    It's possible there are more than one options with the same option type in packet.
  - option: if success, you can access option using *option
- Return value:
  - true: add success
  - false: add failure, please check the header->len, header->oe, header->option->ot_len, otype and oidx.

3.2.4 espconn_mesh_get_src_addr

- Declaration:
  bool espconn_mesh_get_src_addr(struct mesh_header_format *head, uint8_t **src_addr)
- Description:
  The function is used to get source address of mesh packet.
- Parameters:
  - head: mesh packet header

- src_addr: source address
  if success, you can access source address using *src_addr

- Return value:
  - true: success
  - false: failure

3.2.5 espconn_mesh_set_src_addr

- Declaration:
  bool espconn_mesh_set_src_addr(struct mesh_header_format *head, uint8_t *src_addr)
- Description:
  The function is used to set source address of mesh packet.
- Parameters:
  - head: mesh packet header
  - src_addr: source address
- Return value:
  - true: success
  - false: failure

3.2.6 espconn_mesh_set_dst_addr

- Declaration:
  bool espconn_mesh_set_src_addr(struct mesh_header_format *head, uint8_t *dst_addr)
- Description:
  The function is used to set destination address of mesh packet.
- Parameters:
  - head: mesh packet header
  - dst_addr: destination address
- Return value:
  - true: success
  - false: failure

3.2.7 espconn_mesh_get_dst_addr

- Declaration:
  bool espconn_mesh_get_dst_addr(struct mesh_header_format *head, uint8_t **dst_addr)
- Description:
  The function is used to get destination address of mesh packet.
- Parameters:
  - head: mesh packet header
  - dst_addr: destination address
    if success, you can access destination address using *dst_addr
- Return value:
  - true: success
  - false: failure

3.2.8 espconn_mesh_get_usr_data

- Declaration:
  bool espconn_mesh_get_usr_data(struct mesh_header_format *head, uint8_t **usr_data,
  uint16_t *data_len)
- Description:
  The function is used to get user data in mesh packet.
- Parameters:
  - head: mesh packet header
  - usr_data: user data
    if success, you can access user data using *usr_addr
  - data_len: user data length
- Return value:
  - true: success
  - false: failure

3.2.9 espconn_mesh_set_usr_data

- Declaration:
  bool espconn_mesh_set_usr_data(struct mesh_header_format *head, uint8_t *usr_data,
  uint16_t data_len)
- Description:
  The function is used to set user data in mesh packet.
- Parameters:
  - head: mesh packet header
  - usr_data: user data
  - data_len: user data length
- Return value:
  - true: success
  - false: failure

3.2.10 espconn_mesh_get_usr_data_proto

- Declaration:
  bool espconn_mesh_get_usr_data_proto(struct mesh_header_format *head,
  enum mesh_usr_proto_type *proto)
- Description:
  The function is used to get protocol used by user data in mesh packet.
- Parameters:
  - head: mesh packet header
  - proto: protocol used by user data
- Return value:
  - true: success
  - false: failure

3.2.11 espconn_mesh_set_usr_data_proto

- Declaration:
  bool espconn_mesh_get_usr_data_proto(struct mesh_header_format *head,
  enum mesh_usr_proto_type proto)
- Description:
  The function is used to set user data in mesh packet.
- Parameters:
  - head: mesh packet header
  - proto: protocol used by user data
- Return value:
  - true: success
  - false: failure

3.2.12 espconn_mesh_enable

- Declaration:
  void espconn_mesh_enable(espconn_mesh_callback enable_cb, enum mesh_type type);
- Description:
  The function is used to enable mesh.
- Attention:
  - ✓ When mesh network is enabled, the system will trigger enable_cb.
    Therefore, after enable mesh, user should wait for the enable_cb to be triggered.
- Parameters:
  - enable_cb: mesh enable callback
  - type: mesh type
- Return value: None
- Instance:

```
void user_init()
{
    espconn_mesh_enable(mesh_demo_enable_cb, MESH_ONLINE);
}
void mesh_demo_enable_cb(int8_t res)
{
    If (res == MESH_OP_FAILURE) {
        printf(“mesh enable fail\n”);
    } else if (res == MESH_ONLINE_SUC) {
        printf(“mesh enable online\n”);
    } else {
        printf(“mesh enable local\n”);
    }
}
```

3.2.13 espconn_mesh_disable

- Declaration:

void espconn_mesh_disable(espconn_mesh_callback disable_cb);

- Description:
  The function is used to disable mesh.
- Attention:
  - ✓ When mesh network is disabled, the system will trigger disable_cb.
    Therefore, after disable mesh, user should wait for the disable_cb to be triggered.
- Parameters:
  - disable_cb: mesh disable callback
- Return value: None

3.2.14 espconn_mesh_deauth_all()

- Declaration:
  void espconn_mesh_deauth_all();
- Description:
  The function is used to reject all the child node
- Parameters: None
- Return value: None

3.2.15 espconn_mesh_print_ver

- Declaration:
  void espconn_mesh_print_ver();
- Description:
  The function is used to print version of mesh
- Parameters: None
- Return value: None

3.2.16 espconn_mesh_scan

- Declaration:
  void espconn_mesh_scan(struct mesh_scan_para_type *para);
- Description:
  The function is used to scan AP around current node
- Attention:
  - ✓ user can scan all the AP or mesh node AP.
  - ✓ If you plan to scan all the AP, please clear grp_id and grp_set in para.
  - ✓ If you just plan to scan mesh node AP, you should set grp_id and grp_set in para.
- Parameters:
  - para: parameter of scan
- Return value: None

3.2.17 espconn_mesh_release_congest()

- Declaration:
  void espconn_mesh_release_congest();
- Description:

The function is used to discard all the packet to parent

- Parameters: None
- Return value: None

3.2.18 espconn_mesh_get_sub_dev_count()

- Declaration:
  void espconn_mesh_get_dev_count();
- Description:
  The function is used to get the number of all the sub nodes of current node
- Parameters: None
- Return value:
  - count: the number of sub nodes

3.2.19 espconn_mesh_local_addr

- Declaration:
  bool espconn_mesh_local_addr(struct ip_addr *ip)
- Description:
  The function is used to check whether the IP address is mesh local IP address or not.
- Attention:
  - ✓ The range of mesh local IP address is 2.255.255.* ~ max_hop.255.255.*.
  - ✓ IP pointer should not be NULL. If the IP pointer is NULL, it will return false.
- Parameters:
  - ip: ip address
- Return value:
  - true: the IP address is mesh local IP address
  - false: the IP address is not mesh local IP address

3.2.20 espconn_mesh_get_node_info

- Declaration:
  bool espconn_mesh_get_node_info(enum mesh_node_type type,
  uint8_t **info, uint16_t *count);
- Description:
  The function is used to get the information of mesh node.
- Attention:
  - ✓ Before enable mesh, you must not use the API.
  - ✓ If type is MESH_NODE_PARENT, count is the number of parent (always 1), info is the mac address of parent
  - ✓ If type is MESH_NODE_CHILD, count is the number of children whose hop away from current node is one. Info is the collection of sub node information, the sub node information is as follows:

```
struct mesh_sub_node_info {
    uint16_t sub_count;
    uint8_t mac[ESP_MESH_ADDR_LEN];
} __packed;
```

sub_count: the number of sub node of current child (exclusive the current child)
mac: the mac address of current child

- ✓ If type is MESH_NODE_ALL, count is the number of sub node and current node, info is NULL.

- Parameters:
  - type : mesh node type.
  - info : the information will be saved in *info.
  - count : the node count in *info.
- Return value:
  - true: Success
  - false: Failure
- Instance:

```
uint8_t child = 0;
uint16_t count = 0;
uint8_t *node_info = NULL;
if (!espconn_mesh_get_node_info(MESH_NODE_PARENT, &node_info, &count))
    return;
printf("number of parent is:%d\n", count);
printf("mac of parent is:" MACSTR "\n", MAC2STR(node_info));

if (!espconn_mesh_get_node_info(MESH_NODE_CHILD, &node_info, &count))
    return;
for (child = 0; child < count; child ++) {
    struct mesh_sub_node_info *info = (struct mesh_sub_node_info *)node_info + i;
    printf("number of all sub node under child %d is:%d\n", child, info->sub_count);
    Printf("mac of current child %d is:" MACSTR "\n", MAC2STR(info->mac))
}

if (!espconn_mesh_get_node_info(MESH_NODE_ALL, NULL, &count))
    return;
printf("number of node under current node is:%d\n", count);
```

3.2.21 espconn_mesh_get_max_hops

- Declaration:

```
uint8_t espconn_mesh_get_max_hops()
```

- Description:

  The function is used to get current max hop of mesh network
- Parameters: None
- Return value:
  - hop: current max_hop of mesh network

3.2.22 espconn_mesh_layer

- Declaration:

uint8_t espconn_mesh_layer(struct ip_addr *ip)

- Description:

  The function is used to get hop away from router according to IP address.

- Attention:
  - ✓ If ip is not local IP address, the layer will be one.
  - ✓ ip should not be NULL. If the ip is NULL, the layer will be one.
- Parameters:
  - ip: ip address
- Return value:
  - layer: hop away from router.

3.2.23 espconn_mesh_set_router

- Declaration:

  bool espconn_mesh_set_router(struct station_config *router);

- Description:

  The function is used to set router AP information for mesh node

- Attention:
  - ✓ The API should be called before enable mesh.
- Parameters:
  - router: router AP information (ssid, password, bssid (optional))
- Return value:
  - true: Success
  - false: Failure

3.2.24 espconn_mesh_get_router

- Declaration:

  bool espconn_mesh_get_router(struct station_config *router);

- Description:

  The function is used to get router AP information used by mesh node

- Attention:
  - ✓ The API should be called after user receives the first user packet from parent.
  - ✓ User should provide the router buffer to save router AP information
- Parameters:
  - router: if success, the router AP information will be saved in router
- Return value:
  - true: Success
  - false: Failure

3.2.25 espconn_mesh_encrypt_init

- Declaration:

  bool espconn_mesh_encrypt_init(AUTH_MODE mode, uint8_t *passwd, uint8_t pw_len);

- Description:

  The function is used to init encrypt algorithm and password for mesh AP.

- Attention:
  - ✓ The API should be called before enable mesh.
- Parameters:
  - mode: encrypt algorithm (WPA_PSK, WPA2_PSK, WPA_WPA2_PSK)
  - passwd: password of mesh AP
  - pw_len: password length
- Return value:
  - true: Success
  - false: Failure

3.2.26 espconn_mesh_group_id_init

- Declaration:
  bool espconn_mesh_group_id_init(uint8_t *grp_id, uint16_t gid_len);
- Description:
  The function is used to init group id for mesh node.
- Attention:
  - ✓ The API should be called before enable mesh.
  - ✓ The current group id length must be 6
- Parameters:
  - grp_id: group id
  - gid_len: group id length
- Return value:
  - true: Success
  - false: Failure

3.2.27 espconn_mesh_server_init

- Declaration:
  bool espconn_mesh_server_init(struct ip_addr *ip, uint16_t port);
- Description:
  The function is used to set server ip and port for mesh node.
- Attention:
  - ✓ The API should be called before enable mesh.
- Parameters:
  - ip: IP address of server
  - port: port used by server for mesh
- Return value:
  - true: Success
  - false: Failure

3.2.28 espconn_mesh_set_max_hop

- Declaration:
  bool espconn_mesh_set_max_hop(uint8_t max_hops);
- Description:

The function is used to set max_hops for mesh network.

- Attention:
  - ✓ The API should be called before enable mesh.
- Parameters:
  - max_hops: max hops of mesh network
- Return value:
  - true: Success
  - false: Failure

3.2.29 espconn_mesh_set_ssid_prefix

- Declaration:
  bool espconn_mesh_set_ssid_prefix(uint8_t *prefix, uint8_t prefix_len);
- Description:
  The function is used to set SSID prefix for mesh AP.
- Attention:
  - ✓ The API should be called before enable mesh.
- Parameters:
  - prefix: prefix of SSID
  - prefix_len: prefix length
- Return value:
  - true: Success
  - false: Failure

3.2.30 espconn_mesh_get_status

- Declaration:
  int8_t espconn_mesh_get_status();
- Description:
  The function is used to get current status of mesh node.
- Attention:
  - ✓ The API should be called after enable mesh.
- Parameters: None
- Return value:
  - MESH_DISABLE: mesh is disabled
  - MESH_WIFI_CONN: node is trying to connect parent WIFI AP
  - MESH_NET_CONN: node has got its IP address and tries to establish TCP connect with parent
  - MESH_ONLINE_AVAIL: online mesh is available
  - MESH_LOCAL_AVAIL: local mesh is available

3.2.31 espconn_mesh_connect

- Declaration:
  int8_t espconn_mesh_connect(struct espconn *usr_esp);
- Description:
  The function is used to establish virtual connect for user.

- Attention:
  - ✓ The API should be called after mesh_enable_cb.
  - ✓ If fail, returns non-0 value, there is no connection, so it won't enter any callback
- Parameters:
  - usr_esp: user virtual connection
- Return value:
  - 0 : succeed
  - Non-0 : error code
    - ESPCONN_RTE - Routing Problem
    - ESPCONN_MEM - Out of memory
    - ESPCONN_ISCONN - Already connected
    - ESPCONN_ARG - illegal argument

3.2.32 espconn_mesh_disconnect

- Declaration:
  int8_t espconn_mesh_disconnect(struct espconn *usr_esp);
- Description:
  The function is used to disconnect mesh TCP connection
- Attention:
  - ✓ If fail, returns non-0 value, there is no connection, so it won't enter any callback
- Parameters:
  - usr_esp: user virtual connection
- Return value:
  - 0 : succeed
  - Non-0 : error code
    - ESPCONN_RTE - Routing Problem
    - ESPCONN_MEM - Out of memory
    - ESPCONN_ISCONN - Already connected
    - ESPCONN_ARG - illegal argument

3.2.33 espconn_mesh_sent

- Declaration:
  int8_t espconn_mesh_sent(struct espconn *usr_esp, uint8_t *pdata, uint16_t len);
- Description:
  The function is used to sent packet using mesh
- Attention:
  - ✓ If fail, returns non-0 value.
  - ✓ If fail, user is responsible to save packet for next sent or discard packet.
- Parameters:
  - usr_esp: user virtual connection
  - pdata: packet buffer
  - len: packet length
- Return value:

- 0 : succeed
- Non-0 : error code
  - ESPCONN_RTE - Routing Problem
  - ESPCONN_MEM - Out of memory
  - ESPCONN_ISCONN - Already connected
  - ESPCONN_ARG - illegal argument

3.2.34 espconn_mesh_set_scan_retries

- Declaration:
  bool espconn_mesh_set_scan_retries(uint8_t retries);
- Description:
  The function is used to set scan retries if no available AP to been found
- Attention:
  - ✓ If no available AP mesh node, the scan retries will works.
  - ✓ If retries should be larger than zero (zero will be failed).
  - ✓ One scan will take about 15 * retries seconds at most.
    If retries is 2, the scan will take 30 seconds at most.
- Parameters:
  - retries: scan retry count
- Return value:
  - True : Succeed
  - False : Failure

3.2.35 espconn_mesh_regist_conn_ready_cb

- Declaration:
  bool espconn_mesh_regist_conn_ready_cb(espconn_mesh_usr_callback cb);
- Description:
  The function is used to register user callback. If TCP connection with parent node is ready, mesh will call the user callback.
- Parameters:
  - cb: user callback function
- Return value:
  - True : Succeed
  - False : Failure

3.2.36 espconn_mesh_regist_usr_cb

- Declaration:
  bool espconn_mesh_regist_usr_cb(espconn_mesh_usr_callback cb);
- Description:
  The function is used to register user callback. If child node joins parent, parent will trigger the callback to indicate user.
- Parameters:
  - cb: user callback function

- Return value:
  - True : Succeed
  - False : Failure

3.2.37 espconn_mesh_disp_route_table

- Declaration:
  void espconn_mesh_disp_route_table();
- Description:
  The function is used to display route table of current node.
- Parameters: None
- Return value: None

5. Instance:

The chapter will give some instance to guide implement application based on mesh. If you are confuse how to implement application for IOT device, please refer to Demo for device, it will guide you to build general packet parser, build response packet. If you are interested in mobile application or server to control IOT node, please read the code of Demo for mobile or server. It will guide you to build control packet with destination of IOT device. If you want to get topology of mesh network, please investigate the code of demo for topology.

The last instance is one simple and complete device demo based on mesh network. You can find the following features in this instance:

1) Initial flow of mesh and setup mesh network
2) Provide reference to implement general packet parser
3) How to build packet and response packet

5.1 Demo for device:

```
void usr_recv_entrance(struct espconn *esp, uint8_t *mesh_data, uint16_t len)
{
    ......
    // parameter check
    ......

    uint8_t *usr_data = NULL;
    uint16_t usr_data_len = 0;
    struct mesh_header_format *mesh_header = NULL;
    enum mesh_usr_proto_type proto = M_RPOTO_NONE;

    mesh_header = (struct mesh_header_format *)mesh_data;
    if (espconn_mesh_get_usr_data(mesh_header, &usr_data, &usr_data_len)) {
        printf("get user data fail\n");
        return;
    }

    if (espconn_mesh_get_usr_data_proto(mesh_header, &proto)) {
        printf("get user data proto fail\n");
        return;
    }

    // general user data parser which supports multiple protocols.
    usr_data_parser[proto]->proto_processor(usr_data, usr_data_len);

    // build response packet
    uint8_t *src_addr = NULL, *dst_addr = NULL;
```

```
    uint8_t resp_json_packet_body[] = {**************************};
    uint16_t resp_data_len = sizeof(resp_json_packet_body);

    espconn_mesh_get_src_addr(mesh_header, &src_addr);
    espconn_mesh_get_dst_addr(mesh_header, &dst_addr);

    mesh_header = (struct mesh_header_format *)espconn_mesh_create_packet(
                        src_addr, dst_addr, false, true, M_PROTO_JSON, resp_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 back the response
    espconn_mesh_sent(esp, mesh_header, mesh_header->len);
    free(mesh_header);
}
```

5.2 Demo for mobile or server:

```
void controller_entrance(.....)
{
    .....
    // status check
    .....

    // build control packet
    uint8_t json_control_data[] = {**********************************};
    uint16_t control_data_len = sizeof(json_control_data)
    struct mesh_header_format *mesh_header = NULL;
    uint8_t src_addr[] = {0,0,0,0,0,0}, dst_addr[] = {******};

    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);
}
```

5.3 Demo to get topology:

```
void topology_entrance(.....)
{
    .....
    // status check
    .....

    // build get 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[] = {******}; // mac address of root device
    Uint8_t dev_mac[6] = {******}; // zero represents topology of all the device
    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;
    }
    // sent get topology packet
    espconn_mesh_sent(esp, mesh_header, mesh_header->len);
    free(mesh_header);
}
```

5.4 Demo for parsing topology response:

```
void topology_parser_entrance(....., uint8_t *topo_resp, uint16_t len)
{
    ......
    // parameter check
    ......
    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;
    }

    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);
    }
}
```

5.5 Simple and complete demo for Dev-App:

```
/******************************************************************
 * Copyright 2015-2016 Espressif Systems
 *
 * FileName: mesh_demo.c
 *
 * Description: mesh demo
 *
 * Modification history:
 *       2016/01/12, v1.0 create this file.
*******************************************************************/
#include "mem.h"
#include "mesh.h"
#include "osapi.h"
#include "c_types.h"
#include "espconn.h"
#include "user_config.h"

/* please change ROUTER_SSID and ROUTER_PASSWD according to your router */
#define ROUTER_SSID         "ESP_IOT"          // the SSID of router
#define ROUTER_PASSWD    "123456789"    // the password of router
#define MESH_PREFIX         "esp_mesh"       // the SSID prefix of mesh node
#define MESH_PASSWD       "123456789"    // the password of mesh node AP
#define ESP_MESH_MAX_HOP (4)                // max hops of mesh network

#define MESH_DEMO_PRINT    ets_printf
#define MESH_DEMO_STRLEN ets_strlen
#define MESH_DEMO_MEMCPY ets_memcpy
#define MESH_DEMO_MEMSET ets_memset
#define MESH_DEMO_FREE      os_free

static esp_tcp ser_tcp;
static struct espconn ser_conn;
static const uint16_t server_port = 7000;
static const uint8_t server_ip[4] = {192, 168, 11, 116};

void esp_mesh_demo_test();
void esp_server_con_cb(void *);
void esp_recv_entrance(void *, char *, uint16_t);

void esp_recv_entrance(void *arg, char *pdata, uint16_t len)
{
      uint8_t *usr_data = NULL;
```

```
    uint16_t usr_data_len = 0;
    uint8_t *src = NULL, *dst = NULL;
    enum mesh_usr_proto_type proto;
    uint8_t *resp = "{\"rsp_key\":\"rsp_key_value\"}";
    struct mesh_header_format *header = (struct mesh_header_format *)pdata;

    MESH_DEMO_PRINT("recv entrance\n");

    if (!pdata)
        return;

    if (!espconn_mesh_get_usr_data_proto(header, &proto))
        return;
    if (!espconn_mesh_get_usr_data(header, &usr_data, &usr_data_len))
        return;
    /*
     * process packet
     * call packet_parser(...)
     * general packet_parser demo
     */

#if 0
     packet_parser[proto]->handler(arg, usr_data, usr_data_len);
#endif

    /*
     * then build response packet,
     * and give response to controller
     * the following code is response demo
     */
#if 0
    if (!espconn_mesh_get_src_addr(header, &src) || !espconn_mesh_get_dst_addr(header, &dst)) {
        MESH_DEMO_PRINT("get addr fail\n");
        return;
    }

    header = (struct mesh_header_format *)espconn_mesh_create_packet(
                                        src,      // destination address
                                        dst,      // source address
                                        false,    // not p2p packet
                                        true,    // piggyback flow request
                                        M_PROTO_JSON,    // packet with JSON format
                                        MESH_DEMO_STRLEN(resp),    // data length
                                        false,    // no option
```

```
                                        0,          // option len
                                        false,      // no frag
                                        0,          // frag type, this packet doesn't use frag
                                        false,      // more frag
                                        0,          // frag index
                                        0);         // frag length
    if (!header) {
        MESH_DEMO_PRINT("create packet fail\n");
        return;
    }

    if (!espconn_mesh_set_usr_data(header, resp, MESH_DEMO_STRLEN(resp))) {
        MESH_DEMO_PRINT("set user data fail\n");
        MESH_DEMO_FREE(header);
        return;
    }

    if (espconn_mesh_sent(&ser_conn, header, header->len)) {
        MESH_DEMO_PRINT("mesh sent fail\n");
        MESH_DEMO_FREE(header);
        return;
    }

    MESH_DEMO_FREE(header);
#endif
}

void esp_server_con_cb(void *arg)
{
    static os_timer_t tst_timer;
    struct espconn *server = (struct espconn *)arg;

    if (server != &ser_conn) {
        MESH_DEMO_PRINT("con_cb, para err\n");
        return;
    }

    os_timer_disarm(&tst_timer);
    os_timer_setfn(&tst_timer, (os_timer_func_t *)esp_mesh_demo_test, NULL);
    os_timer_arm(&tst_timer, 5000, true);
}

void mesh_enable_cb(int8_t res)
{
```

```
 MESH_DEMO_PRINT("mesh_enable_cb\n");

if (res == MESH_OP_FAILURE) {
        MESH_DEMO_PRINT("enable mesh fail\n");
       return;
}

/*
  * try to establish user virtual connect
  * user can use the virtual connect to sent packet to any node, server or mobile.
  * if you want to sent packet to one node in mesh, please build p2p packet
  * if you want to sent packet to server/mobile, please build unicast packet
  * if you want to sent bcast/mcast packet, please build bcast/mcast packet
  */
MESH_DEMO_MEMSET(&ser_conn, 0 ,sizeof(ser_conn));
MESH_DEMO_MEMSET(&ser_tcp, 0, sizeof(ser_tcp));

MESH_DEMO_MEMCPY(ser_tcp.remote_ip, server_ip, sizeof(server_ip));
ser_tcp.remote_port = server_port;
ser_tcp.local_port = espconn_port();
ser_conn.proto.tcp = &ser_tcp;

if (espconn_regist_connectcb(&ser_conn, esp_server_con_cb)) {
       MESH_DEMO_PRINT("regist con_cb err\n");
       espconn_mesh_disable(NULL);
       return;
}

if (espconn_regist_recvcb(&ser_conn, esp_recv_entrance)) {
       MESH_DEMO_PRINT("regist recv_cb err\n");
       espconn_mesh_disable(NULL);
       return;
}

/*
  * regist the other callback
  * sent_cb, reconnect_cb, disconnect_cb
  * if you donn't need the above cb, you donn't need to register them.
  */

if (espconn_mesh_connect(&ser_conn)) {
       MESH_DEMO_PRINT("connect err\n");
       espconn_mesh_disable(NULL);
       return;
```

```
    }
}

void esp_mesh_demo_test()
{
    uint8_t src[6];
    uint8_t dst[6];
    struct mesh_header_format *header = NULL;

    /*
     * the mesh data can be any content
     * it can be string(json/http), or binary(MQTT).
     */
    char *tst_data = "{\"req_key\":\"req_key_val\"}\r\n";
    // uint8_t tst_data[] = {'a', 'b', 'c', 'd'};}
    // uint8_t tst_data[] = {0x01, 0x02, 0x03, 0x04, 0x00};

    if (!wifi_get_macaddr(STATION_IF, src)) {
        MESH_DEMO_PRINT("get sta mac fail\n");
        return;
    }
    MESH_DEMO_MEMCPY(dst, server_ip, sizeof(server_ip));
    MESH_DEMO_MEMCPY(dst + sizeof(server_ip), &server_port, sizeof(server_port));

    header = (struct mesh_header_format *)espconn_mesh_create_packet(
                                        dst,      // destination address
                                        src,      // source address
                                        false, // not p2p packet
                                        true,    // piggyback flow request
                                        M_PROTO_JSON,    // packet with JSON format
                                        MESH_DEMO_STRLEN(tst_data),    // data length
                                        false, // no option
                                        0,         // option length
                                        false, // no frag
                                        0,         // frag type, this packet doesn't use frag
                                        false, // more frag
                                        0,         // frag index
                                        0);       // frag length
    if (!header) {
        MESH_DEMO_PRINT("create packet fail\n");
        return;
    }

    if (!espconn_mesh_set_usr_data(header, tst_data, MESH_DEMO_STRLEN(tst_data))) {
```

```
            MESH_DEMO_PRINT("set user data fail\n");
            MESH_DEMO_FREE(header);
            return;
        }

        if (espconn_mesh_sent(&ser_conn, (uint8_t *)header, header->len)) {
            MESH_DEMO_PRINT("mesh sent fail\n");
            MESH_DEMO_FREE(header);
            return;
        }

        MESH_DEMO_FREE(header);
}

bool esp_mesh_demo_init()
{
        struct station_config config;

        /*
         * mesh group id
         * mesh_group_id and mesh_ssid_prefix represent mesh network
         */
        uint8_t grp_id[6] = {0x18, 0xFE, 0x34, 0xA5, 0x3B, 0xAD};

        // print version of mesh
        espconn_mesh_print_ver();

        /*
         * set the AP password of mesh node
         */
        if (!espconn_mesh_encrypt_init(AUTH_WPA_WPA2_PSK, MESH_PASSWD,
                                       MESH_DEMO_STRLEN(MESH_PASSWD))) {
            MESH_DEMO_PRINT("set pw fail\n");
            return false;
        }

        /*
         * if you want set max_hop > 4
         * please make the heap is available
         * mac_route_table_size = (4^max_hop - 1)/3 * 6
         */
        if (!espconn_mesh_set_max_hops(ESP_MESH_MAX_HOP)) {
            MESH_DEMO_PRINT("fail, max_hop:%d\n",
                                       espconn_mesh_get_max_hops());
```

```
        return false;
    }

    /*
     * mesh_ssid_prefix
     * mesh_group_id and mesh_ssid_prefix represent mesh network
     */
    if (!espconn_mesh_set_ssid_prefix(MESH_PREFIX, MESH_DEMO_STRLEN(MESH_PREFIX))) {
        MESH_DEMO_PRINT("set prefix fail\n");
        return false;
    }

    /*
     * mesh_group_id
     * mesh_group_id and mesh_ssid_prefix represent mesh network
     */
    if (!espconn_mesh_group_id_init(grp_id, sizeof(grp_id))) {
        MESH_DEMO_PRINT("set grp id fail\n");
        return false;
    }

    if (!espconn_mesh_server_init(server_ip, server_port)) {
        MESH_DEMO_PRINT("server_init fail\n");
        return false;
    }

    MESH_DEMO_MEMSET(&config, 0, sizeof(config));
    MESH_DEMO_MEMSET(config.ssid, ROUTER_SSID, MESH_DEMO_STRLEN(ROUTER_SSID));
    MESH_DEMO_MEMSET(config.password, ROUTER_PASSWD, MESH_DEMO_STRLEN(ROUTER_PASSWD));
    /*
     * you can use esp-touch(smart configure) to sent information about router AP to mesh node
     * but, if you donn't use esp-touch, you should use espconn_mesh_set_router to set router for mesh node
     */
    if (!espconn_mesh_set_router(&config)) {
        MESH_DEMO_PRINT("set_router fail\n");
        return false;
    }

    return true;
}

void user_rf_pre_init(void){}
void user_pre_init(void){}
```

```
/******************************************************************************
 * FunctionName : user_init
 * Description  : entry of user application, init user function here
 * Parameters   : none
 * Returns      : none
*******************************************************************************/
void user_init(void)
{
    if (!esp_mesh_demo_init())
        return;
    /*
     * enable mesh
     * after enable mesh, you should wait for the mesh_enable_cb to be triggered.
     */
     espconn_mesh_enable(mesh_enable_cb, MESH_ONLINE);
}
```