ESP-TOUCH
User Guide

Version 1.1

Copyright © 2016

About This Guide

This document introduces ESP-TOUCH protocol and the relevant application with the structure as below.

Chapter	Title	Subject
Chapter 1	Technology Overview	Provides technical principles of ESP-TOUCH.
Chapter 2	ESP-TOUCH Operations	Provides instructions on how to use ESP-TOUCH.
Chapter 3	API Development	Provides information on APIs.
Chapter 4	Performance Analysis	Introduces ESP-TOUCH error correcting arithmetic and provides performance analysis.

Release Notes

Date	Version	Release notes
2015.12	V1.0	First release.
2016.04	V1.1	Updated Chapter 2 & 3.

Table of Contents

1. Technology Overview 1

2. ESP-TOUCH Operations....3

2.1. ESP-TOUCH Functional Overview....3

2.2. ESP-TOUCH Operation Process....3

3. API Development....4

3.1. smartconfig_start....4

3.2. smartconfig_stop....6

3.3. smartconfig_set_type....7

3.4. Struct....7

4. ESP-TOUCH Performance Analysis....9

1. Technology Overview

Espressif's ESP-TOUCH protocol implements Smart Config technology to help users connect ESP8266EX-embedded devices to a Wi-Fi network through simple configuration on a smartphone.

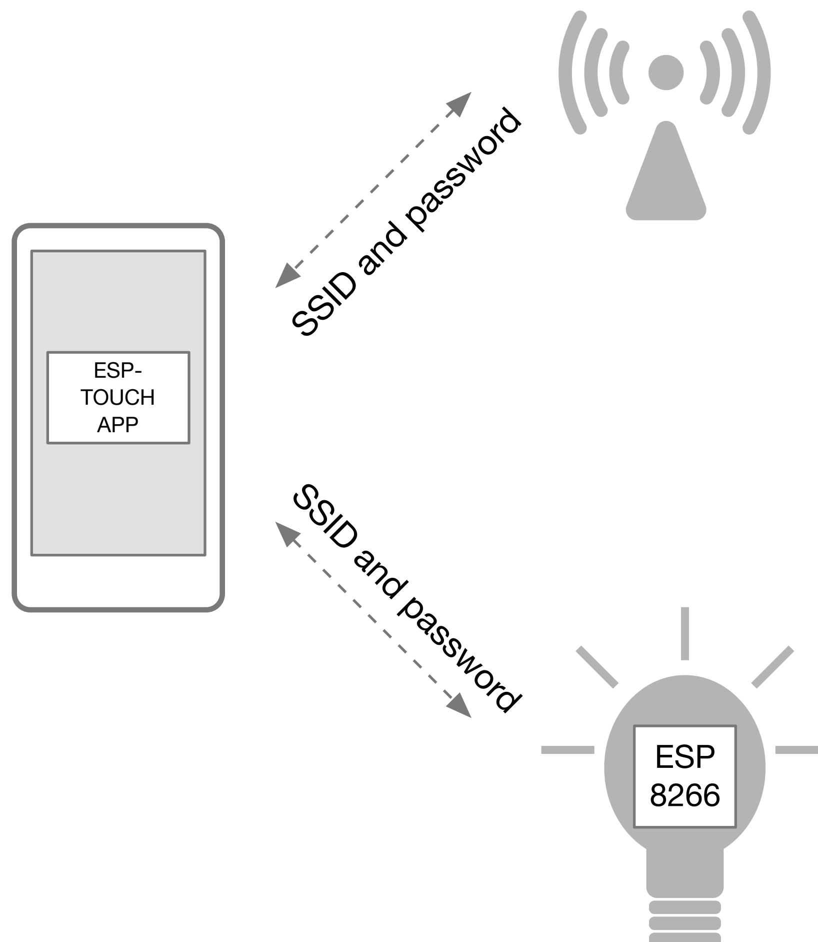

Figure 1-1 Typical ESP-TOUCH Application

Since the ESP8266 device (hereinafter referred to as the device) is not connected to the network at the beginning, the ESP-TOUCH application cannot send the information to the device directly. With ESP-TOUCH communication protocol, a Wi-Fi enabled device such as a smartphone sends UDP packets to the Wi-Fi Access Point (AP), and encodes the SSID and password into the Length field of a sequence of UDP packets where the ESP8266 device can reach and decode the information.

6	6	2	3	5	Variable	4
DA	SA	Length	LLC	SNAP	DATA	FCS

Figure 1-2 Data Packet Structure

2. ESP-TOUCH Operations

2.1. ESP-TOUCH Functional Overview

The ESP8266 RTOS SDK and NONOS SDK both support ESP-TOUCH.

The SDKs also integrate AIRKISS protocol developed by Wechat so that users can configure the device either via ESP-TOUCH App or on the Wechat client-side.

> **Note:**
>
> Users can download ESP-TOUCH App source code at: https://github.com/espressifAPP.

2.2. ESP-TOUCH Operation Process

1. Prepare a device that supports ESP-TOUCH, and enable its Smart Config function.
2. Connect your smartphone to the router.
3. Open ESP-TOUCH App installed on the smartphone.
4. Input the router's SSID and password (you do not need to input password if the router is not encrypted) to connect the device.

> **Notes:**
>
> - It only takes the device a few seconds to connect to the router if the two are close. It will take longer to establish the connection with greater distance.
> - Make sure the router is powered on before configuration, or the device is not able to scan the APs around.
> - Sequence of data transmitted from ESP-TOUCH App has an overtime monitoring mechanism. If the device cannot connect to the router within a specified period of time, the App will return the configuration failure message (please refer to the App source code). Similarly, the time period the device takes to obtain the SSID and password will be calculated. If the device cannot obtain SSID and password within a certain period of time, the device will start the next round of Smart Config process. Users can define the overtime settings through `esptouch_set_timeout(uint8 time_s)`.
> - Sniffer mode should be enabled during the Smart Config process. Station and soft-AP modes of the device should be disabled. Other APIs shouldn't be called.
> - After the configuration process is completed, the transmitter side will get IP of the device, and the device will return the IP of the transmitter side. If the user wants to customize the information exchange between the transmitter side and the device, IP information can be explored.
> - If the AP isolation mode is enabled for the router, the App may not get the configuration success message even if the connection has been established.
> - Users can configure multiple devices to connect to the same router simultaneously. Users can choose for multiple returned messages on the App.

3. API Development

Users can call the following APIs to realize ESP-TOUCH configuration. Please use the latest App and firmware. The SDKs provide ESP-TOUCH demo for your reference.

3.1. smartconfig_start

Function:

Configure the device and connect it to the AP.

> ⚠ **Notice:**
>
> - This API can be called in the Station mode only.
> - Call `smartconfig_stop` to stop the Smart Config process first before repeating the process or calling other APIs.

Defintion:

```
bool smartconfig_start(sc_callback_t cb, uint8 log)
```

Parameters:

`sc_callback_t cb`	Smart Config callback; executed when smart-config status changes; parameter `status` of this callback shows the status of Smart Config: • When the `status` is `SC_STATUS_GETTING_SSID_PSWD`, parameter `void *pdata` is a pointer of `sc_type`, meaning Smart Config type: AirKiss or ESP-TOUCH. • When the `status` is `SC_STATUS_LINK`, parameter `void *pdata` is a pointer of `struct station_config`. • When the `status` is `SC_STATUS_LINK_OVER`, parameter `void *pdata` is a pointer of mobile IP address (4 bytes). It applies to ESP-TOUCH configuration process only, otherwise parameter `void *pdata` will be NULL. • When the `status` is others, parameter `void *pdata` is NULL.

`uint8 log`	1 indicates connection process will be printed out via UART interface. Otherwise only the connection result will be printed. For example, • `smartconfig_start(smartconfig_done,1)`: DEBUG information during the connection process will be printed out via serial port. • `smartconfig_start(smartconfig_done)`: DEBUG information will not be printed out, only the connection result will be printed.

Returned Value:

`TRUE`	Succeed
`FALSE`	Failed

Example:

```
void ICACHE_FLASH_ATTR
smartconfig_done(sc_status status, void *pdata)
{
    switch(status) {
        case SC_STATUS_WAIT:
            os_printf("SC_STATUS_WAIT\n");
            break;
        case SC_STATUS_FIND_CHANNEL:
            os_printf("SC_STATUS_FIND_CHANNEL\n");
            break;
        case SC_STATUS_GETTING_SSID_PSWD:
            os_printf("SC_STATUS_GETTING_SSID_PSWD\n");
            sc_type *type = pdata;
            if (*type == SC_TYPE_ESPTOUCH) {
                os_printf("SC_TYPE:SC_TYPE_ESPTOUCH\n");
            } else {
                os_printf("SC_TYPE:SC_TYPE_AIRKISS\n");
            }
            break;
```

```
            case SC_STATUS_LINK:
                os_printf("SC_STATUS_LINK\n");
                struct station_config *sta_conf = pdata;
                wifi_station_set_config(sta_conf);
                wifi_station_disconnect();
                     wifi_station_connect();
                break;
            case SC_STATUS_LINK_OVER:
                os_printf("SC_STATUS_LINK_OVER\n");
                    if (pdata != NULL) {
                    uint8 phone_ip[4] = {0};
                    memcpy(phone_ip, (uint8*)pdata, 4);
                    os_printf("Phone ip: %d.%d.%d.%d
\n",phone_ip[0],phone_ip[1],phone_ip[2],phone_ip[3]);
                    }
                smartconfig_stop();
                break;
        }
}
smartconfig_start(smartconfig_done);
```

3.2. smartconfig_stop

Function:

Stop Smart Config process, and free the buffer taken by `smartconfig_start`.

Note:

After the connection has been established, users can call this API to free the memory taken by `smartconfig_start`.

Defintion:

```
bool smartconfig_stop(void)
```

Parameters:

Null

Returned Value:

TRUE	Succeed

FALSE	Failed

3.3. smartconfig_set_type

Function:

Set the protocol type of `smartconfig_start` mode.

> **Note:**
>
> Please call this API before `smartconfig_start`。

Definition:

```
bool smartconfig_set_type(sc_type type)
```

Parameters:

```
typedef enum {
    SC_TYPE_ESPTOUCH = 0,
    SC_TYPE_AIRKISS,
    SC_TYPE_ESPTOUCH_AIRKISS,
} sc_type;
```

Returned Value:

TRUE	Succeed
FALSE	Failed

3.4. Struct

```
typedef enum {
SC_STATUS_WAIT = 0,
SC_STATUS_FIND_CHANNEL = 0,
SC_STATUS_GETTING_SSID_PSWD,
SC_STATUS_LINK,
SC_STATUS_LINK_OVER,
} sc_status;
```

> ⚠ **Notice:**
>
> `SC_STATUS_FIND_CHANNEL` status: Users may open the App for configuration only when the device is scanning the channels.

```
typedef enum {
SC_TYPE_ESPTOUCH = 0,
SC_TYPE_AIRKISS,
SC_TYPE_ESPTOUCH_AIRKISS,
} sc_type;
```

4. ESP-TOUCH Performance Analysis

The mechanism implied in ESP-TOUCH communication technology can be understood as a specific error ratio over a single communication channel, the value of which varies with different bandwidth. The packet error ratio is 0 ~ 5% when the bandwidth is 20 MHz, and is 0 ~ 17% when the bandwidth reaches 40 MHz. Supposing that the maximum length of data to be transferred is 104 bytes, if error correcting arithmetic is not applied, it will be really difficult for these data to be transferred within limited times.

By adopting cumulative error correcting arithmetic, transmission can be completed within limited times. The theoretical basis of cumulative error correcting arithmetic is that, during multi-round data transmission process, bit error probability of a same data bit is very low, therefore, cumulative results of multi-round data transmission can be analyzed. The correct value of one bit error can be very likely to be found at other rounds, thus data transmission within limited times can be guaranteed. The success rate of data transmission can be generalized as $[1-[1-p]^k]^l$ (p : packet success rate, k : round of transmission, l : length of transmitted data).

Assuming that the length of data to be transmitted is 104 bytes and 72 bytes, the success rate can reach 0.95 when the bandwidth is 20 MHz, and 0.83 when the bandwidth is 40 MHz.

Tables below show probability of data transmission success rate and transmission time when cumulative error correcting arithmetic is adopted.

Table 4-1. ESP-TOUCH Error Correcting Analysis (20 MHz Bandwidth)

Round	Length: 104 Bytes		Length: 72 Bytes	
	Transmission Time (s)	Success Rate	Transmission Time (s)	Success Rate
1	4.68	0.0048	3.24	0.0249
2	9.36	0.771	6.48	0.835
3	14.04	0.987	9.72	0.991
4	18.72	0.9994	12.9	0.9996
5	23.40	0.99997	16.2	0.99998
6	28.08	0.999998	19.4	0.99999

Table 4-2 ESP-TOUCH Error Correcting Analysis (40 MHz Bandwidth)

Round	Length: 104 Bytes		Length: 72 Bytes	
	Transmission Time (s)	Success Rate	Transmission Time (s)	Success Rate
1	4.68	3.84e-9	3.24	1.49e-6
2	9.36	0.0474	6.48	0.121
3	14.04	0.599	9.72	0.701
4	18.72	0.917	12.9	0.942
5	23.40	0.985	16.2	0.989
6	28.08	0.997	19.4	0.998

Espressif IOT Team

www.espressif.com

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2016 Espressif Inc. All rights reserved.