TSN Centralized Controller Design Document

(version 1.1)

OpenTSN Open Source Project Team

May 2021

Version history

Version revision time revision content			file identification
1.0	**2021.4.23**	This version is the initial version of the interface specification document This, mainly contains the **OpenTSN** exchange Machine Data Plane Abstraction, OpenTSN Interchange **Switch configuration process, OpenTSN** exchange machine configuration format	
1.1	**2021.5.17**	This version uses a single-process single-threaded implementation Now, use a state machine to describe the state of the whole state; increase the interface for communication with upper-layer applications mouth.	**TSNLight3.0** single process Centralized Controller

content

1. Design goals

This scheme modifies the centralized controller, and the main modifications include the following:

Add the northbound protocol module that communicates with the upper-layer control application, and communicate with the upper-layer application through the proxy;

Use a state machine to describe the workflow and describe the specific function of each state in detail; use a single entry

In the form of a single thread, it is convenient for later porting to the embedded CPU. TSNLight3.0 can

Used in OpenTSN2.0 and OpenTSN3.0 projects.

2. Overall design

2.1 System structure

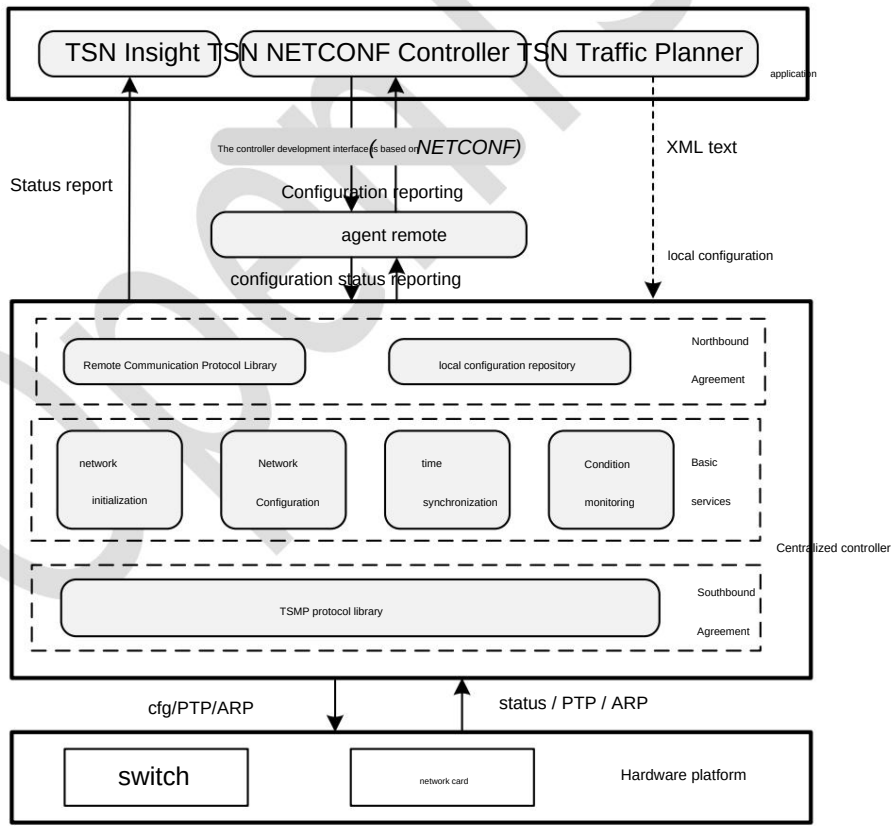


Figure 2-1 TSN centralized controller system structure

TSNLight is the local manager (centralized controller) of the TSN network and completes the

Functions for network initialization, configuration, time synchronization, and status monitoring. TSN agent as on

The bridge between the layer application and the centralized controller is responsible for the conversion of the communication protocol. Upper-layer applications can

Perform offline planning according to user traffic, and finally encapsulate the configuration information into data packets.

It is transmitted to the centralized controller through the proxy, or an xml text can be generated directly,

The text is directly read by the centralized controller to obtain offline planning configuration information.

The controller is generally implemented in one process, using libpcap non-blocking in the process

Receive messages, and only receive one message at a time. When no notification is received within the timeout period

When the message is received, jump out of this function, and execute the timeout processing function according to the current state; when the message is received

When the message is processed, different callback functions are called according to the current state to process the message.

2.2 Workflow

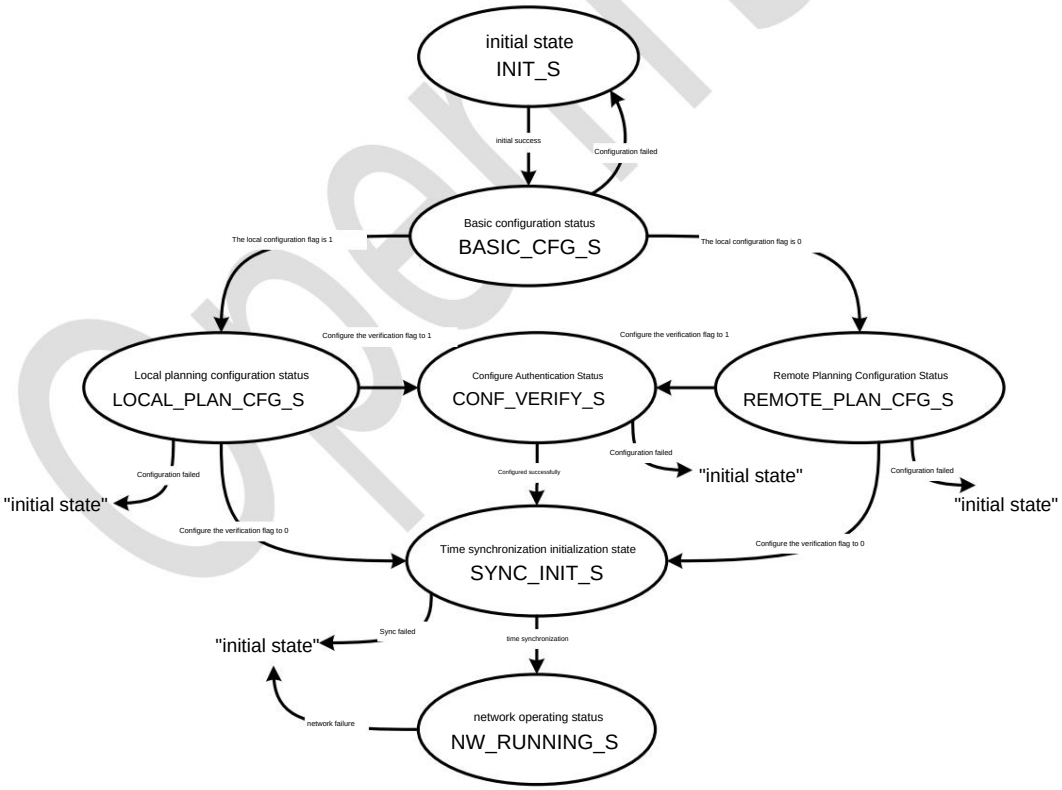


Figure 2-2 Working state diagram of TSN controller

The workflow is divided into seven states, namely initial state, basic configuration state, local

Planning configuration status, remote planning configuration status, configuration verification status, time synchronization initialization

status and network operational status. The specific operation process of each state is in the third section of the detailed design describe.

Initial state: complete the initialization of each variable, the initialization of libnet and libpcap interfaces

The initial configuration data structure is populated according to the initial configuration text. Jump to after initial success Basic configuration status.

Table 2-1 Initial state jump description

current state	Initial end of jump	next state base
initial state	condition	configuration state

Basic configuration state: Verify the data structure based on the topology obtained in the initial state, complete The configuration of basic parameters of nodes and the function of topology verification.

Table 2-2 Basic configuration state transition description

current state	The basic configuration	next state
Basic configuration status	of the jump condition is successful, The local configuration flag is 1	Local planning configuration status
	Basic configuration is successful, The local configuration flag is 0	Remote Planning Configuration Status
	Basic configuration failed, net_run=0	initial state

Remote planning configuration status: Receive offline planning configuration information and store it in the global configuration information data structure, and then configure the nodes in the order of reception, each time a nodes, configure a node.

Table 2-3 Description of remote planning and configuration status transition

current state	Jump condition	next state
Remote Planning Configuration Status	remote planning configuration is successful, Configure the verification flag to 1	Configure Authentication Status
	The remote planning configuration is successful, Configure the verification flag to 0	Time synchronization initialization state
	Remote planning and configuration failed, net_run=0	initial state

Local configuration status: obtain configuration information from the local offline planning xml configuration text, And the configuration information is parsed and stored in the global configuration information data structure.

The order in which the nodes appear in this book is parsed, one node is parsed at a time, and one node is configured.

point.

Table 2-3 Description of local configuration state transition

current state	Jump conditions	next state
Local planning configuration status	Local planning configuration is successful, Configure the verification flag to 1	Configure Authentication Status
	The local planning configuration is successful, Configure the verification flag to 0	Time synchronization initialization state
	Local planning and configuration failed, net_run=0	initial state

Configuration verification status: According to the received verification information, configure the reporting register,

It is used to obtain the information that needs to be verified (reporting information), and submit the received reporting information

To the upper-layer application, the upper-layer application determines whether the configuration is successful.

Table 2-4 Configuration verification status jump description

current state	Jump Condition	next state
Configure Authentication Status	Configuration Verification Successful	configuration validation state
	Configuration Verification Successful net_run=0	initial state

Time synchronization initialization state: complete the function of time synchronization, responsible for

Adjust from time offset to within a certain range.

Table 2-5 Time synchronization state transition description

current state	Jump Condition Time	next state network
Configure Authentication Status	Synchronization Successful Time	running state
	Synchronization Successful net_run=0	initial state

Network running status: The network is in normal working status, and by decoding the received packets

and perform corresponding operations (time synchronization, ARP response, status monitoring).

Table 2-6 Description of network operation status jump

current state	Jump condition	next state
network operating status	network failure net_run=0	initial state

2.3 Overall Process

2.3.1 Overall Process

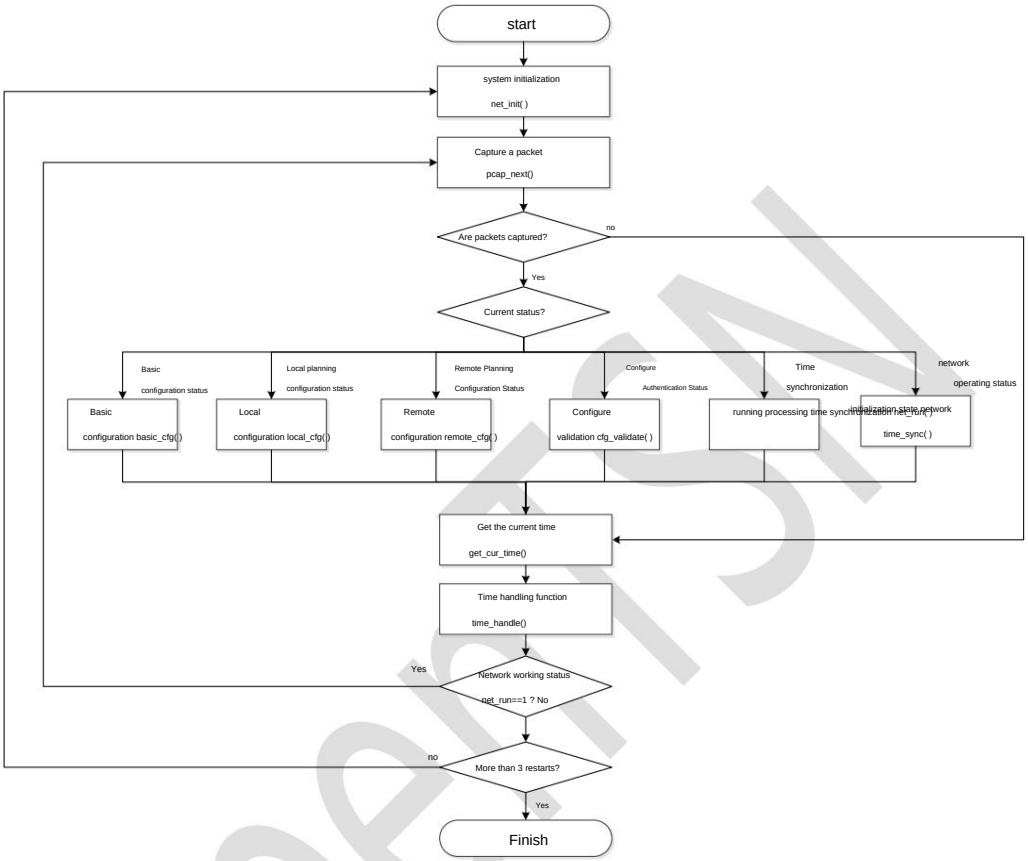


Figure 2-3 Overall flow chart

Main function pseudo code:

Table 2-7 Pseudo code of main function

```
int main()

{

    The flag of init://goto, the network restarts

        net_init()//Network initialization, initialize network configuration, time synchronization, status

Monitor the parameters used

        init_cfg( )//Initial configuration, first configure the nodes directly connected to the controller by default,
```

is that the node can report

```
while(1)
{
    u8 *pkt = pcap_next(p_head);//Non-blocking capture of a packet, timeout or
```

When the operator captures the message, it will execute down

```
    if(pkt != NULL)//The message is captured
    {
      switch(G_STATE)//The processing logic that needs to be judged according to the state
    {
        case BASIC_CFG_S: basic_cfg(p_head, pkt); //basic network configuration;
        case LOCAL_PLAN_CFG_S: local planning configuration;
        case REMOTE_PLAN_CFG_S: remote planning configuration;
        case CONF_VERIFY_S: configuration verification;
        case SYNC_INIT_S: Time synchronization initialization;
        case NW_RUNNING_S: time synchronization, status monitoring, dynamic configuration;
      }
    }
    gettimeofday (time);//Get the current time, used to determine whether the timeout
    time_handle(cur_state,time);//Determine whether it times out according to the local time
```

```
        if(work_run==1) //To determine whether the network is working normally, the flag is at the time
change the function
            continue
        else
            goto:init;//Jump to the initial state

        if(restart_num>3)//Determine whether the restart is more than three times
            break;
        else
            continue
    }
    return 0;
}
```

2.3.2 Timeout processing function flow

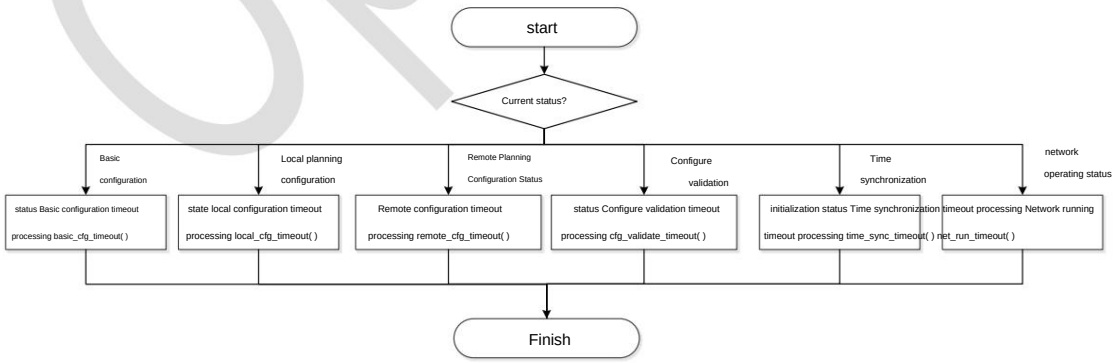


Figure 2-4 Flowchart of the timeout processing function

2.4 Overall data structure

The network data structure is used to represent all the information in the network, using nodes to represent each unit,

All nodes are combined into a network.

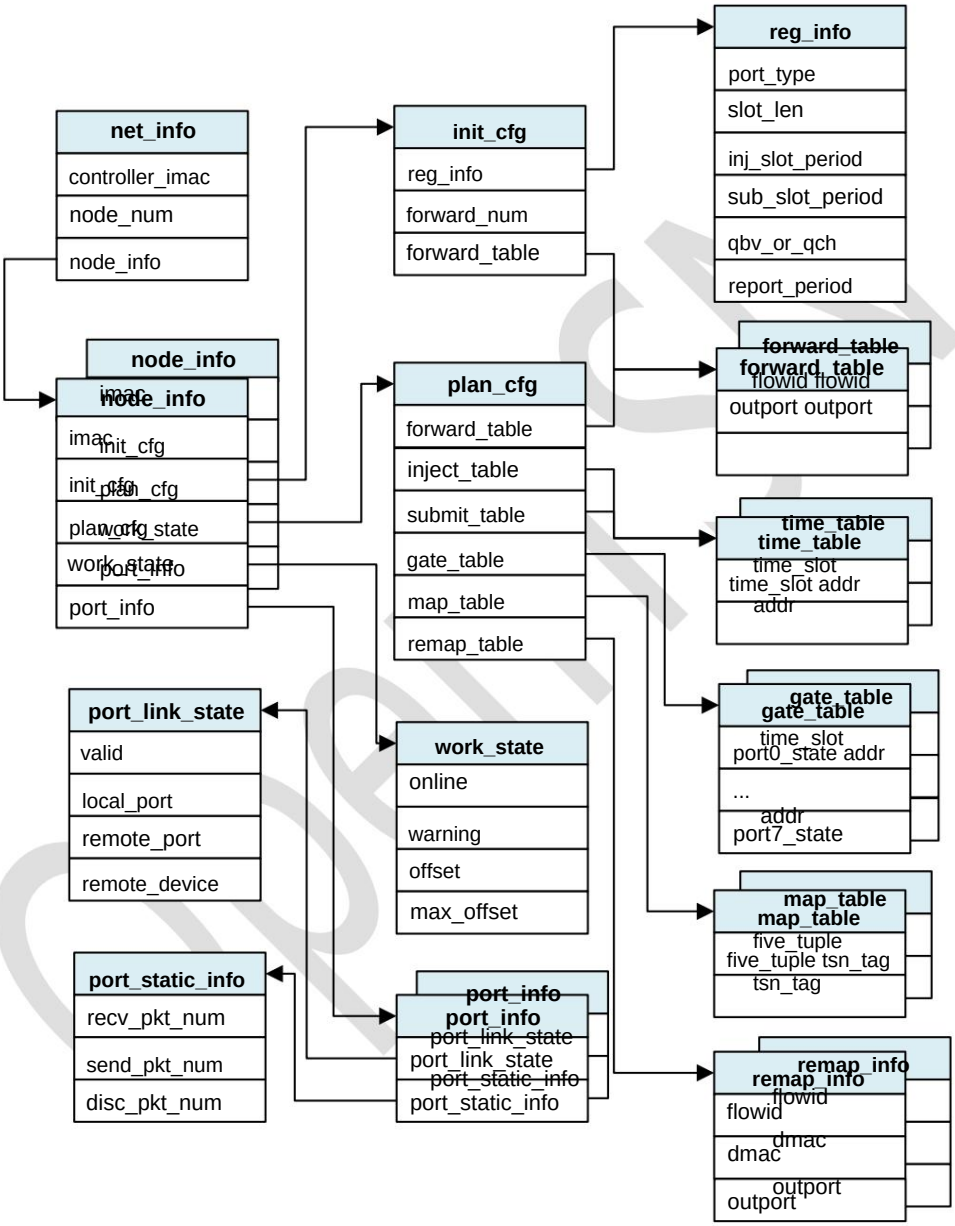


Figure 2-5 Data structure diagram

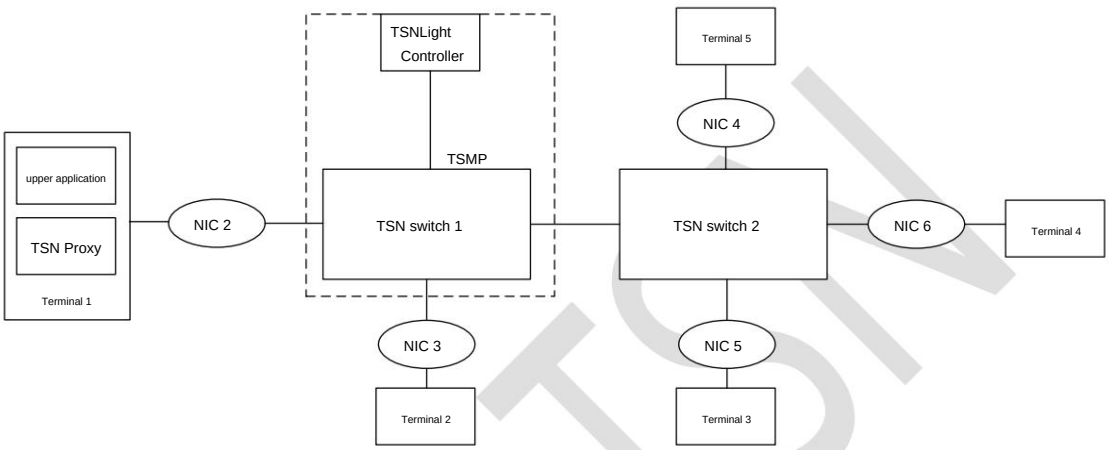
For the content of the specific data structure, refer to the data of each module in the detailed design of Chapter 3

structure.

Appendix 1. Networking Example

An example of networking is shown in the figure below. The controller is directly connected to the TSN switch, and the terminal is directly connected to the TSN switch.

A network card needs to be connected between switches, and upper-layer applications and agents run on one of the terminals above. Design based on OpenTSN 3.0 version



Icon 1-1 Network example diagram

Appendix **2. CNC_API**

cnc_api is the source code file of the API library, providing data reception, data transmission, TSMP Protocol, configuration chip and HCP, reporting and other related library functions are called by other applications.

Appendix 2.1 APIs related to data reception

(1) Data message reception initialization function

Function definition int data_pkt_receive_init(u8* rule); Input parameter filter rule	
string pointer, example: u8* rule= "ether[3:1]=0x01 and ether[12:2]=0xff01"; The return result successfully returns 0, return -1 on failure	
Function description Completes the initialization of data packet receiving resources. including initialization of libpcap handles, Turn on network devices, set filtering rules, etc. (2)	

Data message reception processing function (loop capture)

Function definition int data_pkt_receive_loop(pcap_handler callback);	

Input parameter data	packet processing function callback
Returns 0 for success, -1 for failure Function Description	
Receive data packets in a loop, and send the data packets to the callback function for processing. (3) Data	

message reception processing function (one packet at a time)

Function definition int	data_pkt_receive_dispatch (pcap_handler callback); Input parameter data
packet processing function callback Returns 0 for success, -1 for failure Function Description	
Receive data packets in a loop, and send the data packets to the callback function for processing.	
(4) Data message reception processing function (catch one packet at a time, non-blocking)	

Function definition int	data_pkt_receive_dispatch_1 (pcap_handler callback); Input parameter data
packet processing function callback Returns 0 for success, -1 for failure Function Description Receive	
data packets in a loop, and send the data packets to the callback function for processing. (5) Data	
message receiving and destroying function	

Function definition int	data_pkt_receive_destroy ( ); No input
parameter return result returns 0 for success, -1 for failure Function	
description Complete the destruction of data message receiving	
resources	

Appendix 2.2 APIs related to data sending

(1) Data message sending initialization function

Function definition int	data_pkt_send_init(); Input
parameter no return result Returns 0 for success, -1 for	
failure Function description Complete the initialization of	
data message sending resources. Including initialization of raw socket handle, specifying network card name, assignment of raw socket address structure, etc. (2) Data packet sending processing	

function

Function definition int	data_pkt_send_handle(u8* pkt,u16 len); Input parameter
data packet pointer, data packet length Return result returns 0 if successful, -1 if	
failed Function description Complete the data packet sending process (3) Data	
packet sending destroy function	

Function definition int	data_pkt_send_destroy(); no input
parameters	

Return Result Returns 0 for success, -1 for failure Function	
description Complete the destruction of resources related to data packet transmission	

Appendix 2.3 TSMP protocol related API

(1) TSMP header constructor

Function definition u8* build_tsmp_pkt(tsmp_sub_type type, u16 dimac, u16 pkt_len) Input parameter tsmp subtype,	
destination imac, message length except TSMP header Return result Successfully return the data message pointer address	
after the offset TSMP header, fail return NULL	
Function description Complete the application of tsmp message memory space and the assignment of TSMP header.	

(2) TSMP header (source and destination) address swap function

Function definition int tsmp_header_switch_handle(u8* pkt,u16 len); Input parameters tsmp header	
pointer, tsmp message length Return result Returns the address of the data message pointer after	
removing the TSMP header successfully, otherwise returns NULL	
Function description Complete the exchange of tsmp packet header source and destination mac	

(3) TSMP packet header extraction source MAC function

Function definition u16 get_simac_from_tsmp_pkt(u8* pkt,u16 len); Input parameters tsmp data	
packet pointer, tsmp data packet length Return result Return imac value (host sequence) if	
successful, return -1 if failed Function description is taken from the tsmp packet header The imac	
field of the source mac, the returned imac is the host sequence	

(4) TSMP message memory address release function

Function definition void free_pkt(u8* pkt); Input parameter	
tsmp Data message pointer Return result No function	
description Free the memory of the message	

Appendix 2.4 Configuration chip and HCP related API

(1) Chip configuration message structure and sending function

Function definition int build_send_chip_cfg_pkt(u16 dimac, u32 addr, u8 num, u32 *data); The input parameter dimac is the imac address of the configuration device, addr is the first address of the configuration register, num	
	Indicates the number of configurations, no more than 16, data indicates the configuration content pointer

Return Result Configuration and send successfully return 0, fail to return -1
Function description Build and send chip configuration message.

(2) HCP configuration message construction and sending function

Function definition int build_send_hcp_cfg_pkt(u16 dimac, u32 addr, u32 *data, u16 num) The input parameter dimac configures the imac address of the device, addr indicates the address of the configuration register,
data indicates the configuration content pointer, and num indicates the number of configurations. Return Result Configuration and send successfully return 0, fail to return -1 Function description Build and send HCP configuration
message. (3) Chip single register configuration function

Function definition int cfg_chip_single_register(u16 dimac,chip_reg_info chip_reg) The input parameter dimac configures
the imac address of the device, chip_reg indicates the value of a single register. Returns 0 for configuration and successful
transmission, and -1 for failure. Function description Configure all single registers of the chip, and perform confirm. (4)
Chip table entry configuration function

Function definition int cfg_chip_table(u16 dimac,u8 type,chip_cfg_table_info chip_cfg_table)
Enter the parameter dimac to configure the imac address of the device, and chip_cfg_table represents the table
entry. Return Result Configuration and send successfully return 0, fail to return -1 Function description to
configure the chip's table entry, and to confirm the function.

(5) HCP mapping table configuration function

Function definition int cfg_hcp_map_table(u16 imac,map_table_info *map_table) The input parameter imac configures
the imac address of the device, and map_table represents the data structure of the mapping table. Return Result Configure
and send successfully, return 0, fail to return -1 Function Description Configure the hcp mapping table, and perform the
confirmation function.

(6) HCP remapping table configuration function

Function definition int cfg_hcp_remap_table(u16 *remap_table) imac,remap_table_info
The input parameter dimac configures the imac address of the device, and remap_table indicates remapping.
Return Result Configuration and send successfully return 0, fail to return -1 Function description to configure the
chip's table entry, and to confirm the function. (7) 64-bit host sequence to network sequence function

Function definition u64 htonll(u64 value); Input parameter
64-bit host sequence data Return result 64-bit network
sequence data Function description 64-bit host sequence
to network sequence

(8) chip single register host sequence to network sequence function

Function definition void host_to_net_single_reg(u8 *host); input	
parameter host data pointer return result no function description chip	
single register host sequence to network sequence (9) chip table entry	
host sequence to network sequence function	

Function definition void host_to_net_chip_table(u8 *host,u16 len); Input	
parameter host data pointer, data length return result No function description	
chip table item host sequence to network sequence (10) Chip single register	
information printing function	

Function definition void printf_single_reg(chip_reg_info *chip_reg); Input parameter	
chip register information address pointer Return result No function description	
Print single chip register information (11) Chip table entry information printing	
function	

Function definition void	print_chip_report_table(chip_report_table_info *report_entry);
Input parameter on-chip report item information structure	
pointer Return result No function description Print on-chip	
report item information	

Appendix 2.5 Reporting related APIs

(12) Get report message type function

Function definition u16 get_chip_report_type(u8 *pkt,u16 len); input	
parameter report message pointer, message length return result report	
message report type function description Get the report type from the report	
message	