WCHNET Protocol Stack Library Application Note

Version: 1C

http://wch.cn

1. Overview

With the popularity of the Internet of Things, more and more MCU systems require network communication.

The WCHNET devices with on-chip Ethernet MAC and PHY support 10M Ethernet, full-duplex, half-duplex, auto-negotiation, automatic line conversion and other functions, and can directly interact with network terminals such as PCs and embedded devices.

libwchnet.a provides a TCP/IP subroutine library that integrates Ethernet protocol stacks such as TCP, UDP, ICMP, ARP, RARP, ICMP, and IGMP. It can support TCP, UDP and IPRAW modes.

2. Parameter description

2.1 Configuration

Call WCHNET_ConfigLIB for library configuration. Parameters are as follows:

Name	Macro definition	Bit definition	Description
TxBufSize	Reserved	0-31	Size of MAC transmit buffer
TCPMss	WCHNET_TCP_MSS	0-31	Size of TCP MSS
HeapSize	WCHNET _MEM_HEAP_SIZE	0-31	Size of heap memory
ARPTableNum	WCHNET_NUM_ARP_TABLE	0-31	Number of ARP tables
MiscConfig0	CFG0_TCP_SEND_COPY	0	TCP transmit buffer copy 1: Copy.　　　0: Non-copy.
	CFG0_TCP_RECV_COPY	1	TCP receive copy optimization, for internal debug
	CFG0_TCP_OLD_DELETE	2	Delete old TCP connection 1: Enabled.　　0: Disabled.
	CFG0_IP_REASS_PBUFS	3-7	Number of reassembled IP PBUFs
	Reserved	8-31	-
MiscConfig1	WCHNET_MAX_SOCKET_NUM	0-7	Number of sockets

	Reserved	8-12	-
	WCHNET_PING_ENABLE	13	PING enable 1: Enabled. 0: Disabled.
	TCP_RETRY_COUNT	14-18	TCP retry count
	TCP_RETRY_PERIOD	19-23	TCP retry period The unit is 50 milliseconds.
	Reserved	24	-
	SOCKET_SEND_RETRY	25	Failure and retry sending 1: Enabled. 0: Disabled.
	Reserved	26-31	-
led_link	led_callback	-	PHY Link status indicator
led_data	led_callback	-	Ethernet communication indicator
net_send	eth_tx_set		Ethernet Tx configuration
net_send	eth_rx_set		Ethernet Rx configuration
CheckValid	WCHNET_CFG_VALID	0-31	Configuration value valid flag. A fixed value.

2.2 SocketInf

SocketInf is a socket information list, defined as follows:

    SOCK_INF SocketInf[WCHNET_MAX_SOCKET_NUM]

The address is aligned with 4 bytes. WCHNET_MAX_SOCKET_NUM is the number of sockets. SocketInf stores the information of each socket. Please refer to the definition of SOCK_INF for its information members. This variable is read and written inside the library. If it is not necessary, please do not write to it in the application program (refers to the user program that calls the library function, which is called the application program in this manual, the same below).

2.3 Memp_Memory

The pool allocated memory used internally by WCHNET is mainly used for data reception buffer. For calculation of its size, please refer to the macro definition of WCHNET_MEMP_SIZE in WCHNET.h.

2.4 Mem_Heap_Memory

The heap-allocated memory used internally by WCHNET is mainly used for data transmission buffer. For calculation of its size, please refer to the macro definition of WCHNET_RAM_HEAP_SIZE in WCHNET.h.

2.5 Mem_ArpTable

ARP buffer table is used to record the correspondence between IP addresses and MAC addresses. The size of the ARP buffer table can be configured.

2.6 MemNum, MemSize

MemNum and MemSize are arrays generated according to user configuration. WCHNET uses these two arrays to manage memory allocation, and they cannot be modified.

3. Subroutines

3.1 General table of library subroutines

Classification	Function name	General description
Basic functions	WCHNET_Init	Library initialization
	WCHNET_GetVer	Get library version
	WCHNET_NetInput	Ethernet data input
	WCHNET_PeriodicHandle	Handle periodic tasks
	WCHNET_ETHIsr	Ethernet interrupt service function
	WCHNET_GetPHYStatus	Get PHY status
	WCHNET_QueryGlobalInt	Query global interrupt
	WCHNET_GetGlobalInt	Read and clear global interrupt
	WCHNET_Aton	ASCII address to network address
	WCHNET_Ntoa	Network address to ASCII address
	WCHNET_ConfigLIB	Library parameter configuration
	WCH_GetMacAddr	Get MAC address
socket functions	WCHNET_GetSocketInt	Get socket interrupt and clear
	WCHNET_SocketCreat	Create socket

	WCHNET_SocketClose	Close socket
	WCHNET_SocketRecvLen	Get socket reception length
	WCHNET_SocketRecv	socket receive data
	WCHNET_SocketSend	socket send data
	WCHNET_SocketListen	TCP listen
	WCHNET_SocketConnect	TCP connect
	WCHNET_ModifyRecvBuf	Modify reception buffer
	WCHNET_SocketUdpSendTo	UDP send, specify the target IP and target port
	WCHNET_QueryUnack	Query unsuccessfully sent packets
	WCHNET_SetSocketTTL	Set TTL of socket
	WCHNET_RetrySendUnack	Start TCP retry sending immediately
DHCP functions	WCHNET_DHCPStart	DHCP start
	WCHNET_DHCPStop	DHCP stop
	WCHNET_DHCPSetHostname	Configure DHCP host name
DNS functions	WCHNET_InitDNS	Initialize DNS
	WCHNET_DNSStop	DNS stop
	WCHNET_HostNameGetIp	Get IP address base on host name
KEEPLIVE functions	WCHNET_ConfigKeepLive	Configure library KEEP LIVE parameter
	WCHNET_SocketSetKeepLive	Configure socketKEEPLIVE enable

Interrupt:

The global interrupt and socket interrupt of the library are actually just a sign of the variable, not the hardware interrupt generated by the WCHNET devices.

3.2 WCHNET_Init

Prototype	uint8_t WCHNET_Init(const uint8_t *ip, const uint8_t *gwip, const uint8_t *mask, const uint8_t *macaddr)
Input	Ip - IP address pointer

	Gwip - Gateway address pointer
	Mask - Subnet mask pointer
	Macaddr – MAC address pointer
Output	None
Return	0 – Success. Others – Error. Please refer to WCHNET.h for specific error codes.
Function	Library initialization.

Subnet mask pointer can be set to NULL. If it is set to NULL, the library selects 255.255.255.0 as subnet mask.

3.3 WCHNET_GetVer

Prototype	uint8_t WCHNET_GetVer(void)
Input	None
Output	None
Return	Library version
Function	To get library version.

3.4 WCHNET_NetInput

Prototype	void WCHNET_NetInput(void)
Input	None
Output	None
Return	None
Function	Ethernet data input. Always called in the main program, or called after the reception interrupt is detected.

3.5 WCHNET_PeriodicHandle

Prototype	void WCHNET_PeriodicHandle(void)
Input	None
Output	None

Return	None
Function	Handle periodic tasks in the protocol stack.

3.6 WCHNET_ETHIsr

Prototype	void WCHNET_ETHIsr(void)
Input	None
Output	None
Return	None
Function	Ethernet interrupt service function. Called after Ethernet interrupt is generated.

3.7 WCHNET_GetPHYStatus

Prototype	uint8_t WCHNET_GetPHYStatus(void)
Input	None
Output	None
Return	PHY status
Function	Get current status of PHY. Main status: 0x09 – PHY disconnects 0x2D – PHY establishes connection and negotiation is completed.

3.8 WCHNET_QueryGlobalInt

Prototype	uint8_t WCHNET_QueryGlobalInt(void)
Input	None
Output	None
Return	Global interrupt status
Function	Query global interrupt status. For specific status codes, please refer to WCHNET.h.

3.9 WCHNET_GetGlobalInt

Prototype	uint8_t WCHNET_GetGlobalInt(void)
Input	None
Output	None
Return	Global interrupt status
Function	Read global interrupt and clear it. For specific status codes, please refer to WCHNET.h.

3.10 WCHNET_Aton

Prototype	uint8_t WCHNET_Aton(const char *cp, uint8_t *addr)
Input	*cp - ASCII address to be converted, such as "192.168.1.2" *addr - First address of the memory stored in the converted network address
Output	*addr – Converted network address, such as 0xC0A80102
Return	0 – Success. Others – Failure.
Function	Convert ASCII address to network address.

3.11 WCHNET_Ntoa

Prototype	uint8_t *WCHNET_Ntoa(uint8_t *ipaddr)
Input	*ipaddr – socketID value
Output	None
Return	Converted ASCII address
Function	Convert network address to ASCII address.

3.12 WCHNET_ConfigLIB

Prototype	uint8_t WCHNET_ConfigLIB(struct _WCH_CFG *cfg)
Input	cfg –Configuration parameter
Output	None

Return	0 – Success. Others – Failure.
Function	Library parameter configuration.

3.13 WCHNET_GetMacAddr

Prototype	void WCH_GetMac(uint8_t *macaddr)
Input	*macaddr – Memory address
Output	mac address
Return	None
Function	Get MAC address.

3.14 WCHNET_GetSocketInt

Prototype	uint8_t WCHNET_GetSocketInt(uint8_t socketid)
Input	socketid – socketID value
Output	None
Return	Return socket interrupt. For specific status codes, please refer to WCHNET.h.
Function	Get socket interrupt, and clear socket interrupt.

3.15 WCHNET_SocketCreat

Prototype	uint8_t WCHNET_SocketCreat(uint8_t *socketid,SOCK_INF *socinf)
Input	*socketid – Memory address socinf – Create socket configuration parameter
Output	*socketid – socketID value
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Create socket.

socketinf is only passed as a variable. WCHNET_SocketCreat analyzes the list information. If the information is valid, it will find a free list (n) from SocketInf[WCHNET_MAX_SOCKET_NUM], copy socketinf into SocketInf[n], lock SocketInf[n] and create corresponding UDP/TCP/IPRAW connection. If the creation is successful, write n to socketed, and a success code is returned.

When creating UDP, TCP client or IPRAW, the receive buffer and the size of it should be allocated

before creation. The allocation method of the TCP server is different. The WCHNET_ModifyRecvBuf function should be called to allocate the receive buffer after receiving the successful connection interrupt.

For details, please refer to the related routines.

3.16 WCHNET_SocketClose

Prototype	uint8_t WCHNET_SocketClose(uint8_t socketid, uint8_t mode)
Input	socketid – socketID value mode – socket is a TCP connection, and the parameter is closed.
Output	None
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Close socket.

In UDP and IPRAW modes, "mode" is invalid. Call this function to close socket immediately.

In TCP mode, "mode" can be:
TCP_CLOSE_NORMAL: Normal. Close after 4 handshakes. The closing speed is slower.
TCP_CLOSE_RST: Reset connection. WCHNET sends RST to the target for reset. The closing speed is faster.
TCP_CLOSE_ABANDON: Abandon directly. No information is sent to the target. Close socket. The closing speed is the fastest.

Call this function, and it may generally take a certain period of time to close. This is mainly because the library needs a certain period of time to terminate the TCP connection. As long as the SINT_STAT_TIM_OUT or SINT_STAT_DISCONNECT interrupt is generated, the socket must be in the closed state.

3.17 WCHNET_SocketRecvLen

Prototype	uint32_t WCHNET_SocketRecvLen(uint8_t socketid, uint32_t *bufaddr)
Input	socketid – socketID value *bufaddr – Memory address
Output	*bufaddr – First address of socket data
Return	Length of the received data
Function	Get the length of the data received by socket.

This function is mainly used to get the length of the data received by socket and the address of the receive buffer. The application program can directly use the address output by this function, and

can use the data in the internal receive buffer without copying. This can save RAM to a certain extent. If bufaddr is set to NULL, this function only returns the length of the data received by socket.

3.18 WCHNET_SocketRecv

Prototype	uint8_t WCHNET_SocketRecv(uint8_t socketid, uint8_t *buf, uint32_t *len)
Input	socketid – socketID value *buf – First address of the received data *len - Memory address and length of data expected to be read
Output	*buf - The read data content that is written *len - The actual read length of read
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Socket receives data.

This function copies the data in the socket receive buffer into buf, and the actual length of the copied data will be written into len.

WCHNET supports two modes to receive data. One is the interrupt mode, and the other is the callback mode.

Interrupt mode: After WCHNET receives data, an interrupt is generated. User can read the received data through the WCHNET_SocketRecvLen and WCHNET_SocketRecv functions. IPRAW, UDP and TCP all can receive data in interrupt mode. If buf is not NULL, WCHNET_SocketRecv copies the data of the internal buffer to buf. If buf is NULL, the application layer reads the data in a non-copying way. The function is called only for the pointer offset, and *len is equal to the length of remaining data.

Callback mode: Only valid in UDP mode. WCHNET notifies the application layer to receive data by calling back the AppCallBack function in the SocketInf structure after receiving the data. AppCallBack is implemented by the application layer, and the application layer must read all data in this function. Otherwise WCHNET will forcibly clear it. If the callback mode is not needed, AppCallBack must be cleared to 0 when creating socket. The prototype of the callback function is as follows:

Prototype	void(*AppCallBack)(structSCOK_INF*socinf,uint32_tipaddr,uint16_tport,uint8_t *buf,uint32_t len)
Input	Socinf – Through this parameter, WCHNET transfers the socket information list to the application layer, and the application layer can know the socket information. ipaddr - Source IP address of the message

	port - Source port of the message buf - Buffer address len - Data length
Output	None
Return	None
Function	Receive callback function in UDP mode.

3.19 WCHNET_SocketSend

Prototype	uint8_t WCHNET_SocketSend(uint8_t socketid, uint8_t *buf, uint32_t *len)
Input	socketid – socketID value *buf – First address of the sent data *len - Memory address and length of data expected to be sent
Output	*len – Length of the data that is sent actually
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Socket sends data.

This function copies the data in buf to the transmit buffer of the internal protocol stack. The data is sent, and the length of the data sent actually is output through len. The application layer needs to check len during actual processing, to determine the length of the data sent actually. If too much data is sent, this function will automatically retry to send multiple times. If it returns 0 (success) in this case, this does not mean that all data has been sent.

3.20 WCHNET_SocketUdpSendTo

Prototype	uint8_t WCHNET_SocketUdpSendTo(uint8_t socketid, uint8_t *buf, uint32_t *slen, uint8_t *sip, uint16_t port)
Input	socketid – socketID value *buf – Address of the sent data *slen – Address of the sent length *sip – Target IP address port – Target port number
Output	*slen – Length sent actually

| Return | Execution status. For specific status codes, please refer to WCHNET.h. |
| Function | UDP send, specify the target IP and target port |

In UDP mode, the difference between WCHNET_SocketSend and WCHNET_SocketUdpSendTo is that the former can only send data to the target IP and port specified when the socket is created, while the latter can send data to any IP and port. WCHNET_SocketUdpSendTo is generally used in UDP server mode.

3.21 WCHNET_SocketListen

Prototype	uint8_t WCHNET_SocketListen(uint8_t socketid)
Input	socketid – socketID value
Output	None
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	TCP listen. Used in TCP SERVER mode.

If the application layer needs to establish a TCP SERVER, first use WCHNET_SocketCreat to create a TCP, and then call this function to make TCP enter the Listen mode. TCP in Listen mode does not receive or send data, but only listens for TCP connections. Once a client connects to this server, the library will automatically allocate a socket and generate the SINT_STAT_CONNECT connection interrupt. So the listened TCP does not need to allocate receive buffer.

3.22 WCHNET_SocketConnect

Prototype	uint8_t WCHNET_SocketConnect(uint8_t socketid)
Input	socketid – socketID value
Output	None
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	TCP connect. Used in TCP Client mode.

If the application layer needs to establish a TCP Client, first use WCHNET_SocketCreat to create a TCP, and then call this function to connect. After successful connection, the SINT_STAT_CONNECT connection interrupt is generated. If the remote end is not online or the port is not open, the library will automatically retry a certain number of times, and if it is still unsuccessful, the SINT_STAT_TIM_OUT timeout interrupt will be generated.

3.23 WCHNET_ModifyRecvBuf

Prototype	void WCHNET_ModifyRecvBuf(uint8_t socketid, uint32_t bufaddr, uint32_t bufsize)
Input	socketid – socketID value bufaddr – Address of the receive buffer bufsize – Size of the receive buffer
Output	None
Return	None
Function	Modify socket receive buffer.

In order to make the application layer process data conveniently and flexibly, the library allows to dynamically modify the address and size of the socket receive buffer. Before the receive buffer is modified, it is better to call WCHNET_SocketRecvLen to check whether there is any remaining data in the buffer. Once WCHNET_ModifyRecvBuf is called, the original buffer data will be cleared. In TCP mode, if the connection has been established, call WCHNET_ModifyRecvBuf, and the library will notify the current window size to the remote end.

3.24 WCHNET_SetSocketTTL

Prototype	uint8_t WCHNET_SetSocketTTL(uint8_t socketid, uint8_t ttl)
Input	socketid – socketID value ttl – TTL value
Output	None
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Set socket TTL.

Note: TTL cannot be 0, and it default to 128.

3.25 WCHNET_QueryUnack

Prototype	uint8_t WCHNET_QueryUnack(uint8_t socketid, uint32_t *addrlist, uint16_t lislen)
Input	socketid – socketID value *addrlist – First address of the stored memory lislen – Length of the stored memory

Output	*addrlist – Address list of the data packets that are not sent successfully
Return	Number of unsent and unacknowledged segments
Function	Query the packets that are not sent successfully.

Unack Segment: TCP message that is not sent successfully.

WCHNET_QueryUnack is used to query for the number of messages that are not sent successfully by socket and the address of the messages. Two methods:

(1) Query for the number of "Unack Segment". The addrlist is NULL. WCHNET_QueryUnack returns the number of "socket Unack Segment".

(2) Query for the Unack Segment information. WCHNET_QueryUnack writes the address of these data messages to addrlist.

The application layer can also query whether there is an Unack Segment by looping WCHNET_QueryUnack(sockinf,NULL,0). If it queries for an Unack Segment, call WCHNET_QueryUnack again to get the information:

```
While(1)
{
    If(WCHNET_QueryUnack(sockinf,NULL,0))
    {
        WCHNET_QueryUnack(sockinf, addrlist,sizeof(addrlist));
    }
    /*Other tasks*/
}
```

3.26 WCHNET_RetrySendUnack

Prototype	void WCHNET_RetrySendUnack(uint8_t socketid)
Input	socketid – socketID value
Output	None
Return	None
Function	Start TCP retry sending immediately.

WCHNET_RetrySendUnack is valid only in TCP mode, used to retry sending the messages that are not sent successfully. The application program can check whether the socket data has been successfully sent through WCHNET_QueryUnack. If necessary, call WCHNET_RetrySendUnack to retry sending the data messages immediately. Normally, the application layer is not required to re-send, and WCHNET will retry automatically.

3.27 WCHNET_DHCPStart

Prototype	uint8_t WCHNET_DHCPStart(dhcp_callback dhcp)
Input	dhcp – Application layer callback function
Output	None
Return	0 – Success. Others – Failure.
Function	Start DHCP.

When DHCP succeeds or fails, the library calls the dhcp_callback function, to notify the application layer of the DHCP status. WCHNET passes two parameters to this function, the one is DHCP status (0-Suceed. Others-Fail). When DHCP succeeds, user can get a pointer through the other parameter, and the address pointed to by the pointer stores the IP address, gateway address, subnet mask, primary DNS and secondary DNS in turn, a total of 20 bytes. Note that the pointer is invalid after returned as the dhcp_callback temporary variable.

If there is no DHCP server in the current network, a timeout of about 10 seconds will occur. After the timeout, call the dhcp_callback function to notify the application layer. In this case, DHCP will not stop and will always search for the DHCP Server. User can call WCHNET_DHCPStop to stop DHCP.

Note:
(1) Start DHCP after WCHNET_Init succeeds (necessary).
(2) Create socket after DHCP succeeds (recommended).
(3) DHCP function occupies a UDP socket.

If DHCP fails, communicate via the IP address used in WCHNET_Init.

3.28 WCHNET_DHCPStop

Prototype	uint8_t WCHNET_DHCPStop(void)
Input	None
Output	None
Return	0 – Success. Others – Failure.
Function	Stop DHCP.

3.29 WCHNET_DHCPSetHostname

Prototype	uint8_t WCHNET_DHCPSetHostname(char *name)
Input	*name – First address of DHCP host name

Output	None
Return	0 – Success. Others – Failure.
Function	Configure DHCP host name.

3.30 WCHNET_InitDNS

Prototype	void WCHNET_InitDNS(uint8_t *dnsip, uint16_t port)
Input	*dnsip – dns server IP address port – dns server port number
Output	None
Return	None
Function	Initialize DNS.

If DNS is used, a UDP socket is occupied. Call this function after WCHNET_Init, to start DNS.

3.31 WCHNET_DNSStop

Prototype	void WCHNET_DNSStop (void)
Input	None
Output	None
Return	None
Function	Stop DNS.

After this function is executed, the occupancy of the UDP socket will be released. If it is required to use DNS, call WCHNET_InitDNS again.

3.32 WCHNET_HostNameGetIp

Prototype	uint8_t WCHNET_HostNameGetIp(const char *hostname, uint8_t *addr, dns_callback found, void *arg)
Input	hostname - Host domain name *addr - First address of the memory that stores the parsed ip found - Callback function arg - found parameter

Output	addr: Output the host IP address. Only valid when the function returns 0. Must be 4-byte aligned.
Return	Execution status. For specific status codes, please refer to WCHNET.h.
Function	Get host IP address.

found: Function pointer of the application layer. Its basic prototype is as follows:
typedef void (*dns_callback)(const char *name, uint8_t *ipaddr, void*callback_arg); This function is used to get the IP address of the host. "hostname" represents host name. "addr" represents IP address pointer. If the IP address corresponding to hostname is already in DNS buffer, it returns a success code directly, and outputs the IP address corresponding to the host to addr. If the IP address is not in the buffer, the DNS module initiates the DNS transaction, and asks the DNS server. After failure or success, it calls the found function, and the arg failure parameter is NULL.

arg is a parameter required by found, and it can be NULL. But found cannot be NULL.

Note:

(1) addr must be 4-byte aligned.
(2) The maximum length of the hostname string cannot be greater than 63.

3.33 WCHNET_ConfigKeepLive

Prototype	void WCHNET_ConfigKeepLive(struct _KEEP_CFG *cfg)
Input	*cfg – KEEPLIVE configuration parameter
Output	None
Return	None
Function	Configure library KEEP LIVE parameter.

This function is used to configure the KEEPLIVE parameter in WCHNET. The struct_KEEP_CFG structure is defined in WCHNET.h.

```
struct _KEEP_CFG
{
 uint32_t KLIdle;
 uint32_t KLIntvl;
 uint32_t KLCount;
};
```

KLIdle: Idle time. The KEEPLIVE detection is started after the TCP connection is idle for a certain period of time. The unit is milliseconds. The default value is 20000. The time precision is the TCP retry interval.

KLIntvl: Interval. KEEPLIVE detection timeout interval. The unit is milliseconds. The default

value is 15000.

KLCount: Count. KEEPLIVE detection count. The default value is 9.

This setting is a global setting for the library, and is valid for TCP connections that enable KEEPLIVE. When there is no data transfer between the two parties within the KLIdle time, KEEPLIVE is started. If the other party has no response after KEEPLIVE sends KLCount times, the connection is considered invalid and it will be disconnected. This function should be called after the library is initialized. The time precision is the TCP retry period.

3.34 WCHNET_SocketSetKeepLive

Prototype	uint8_t WCHNET_SocketSetKeepLive(uint8_t socketid, uint8_t enable)
Input	socketid – socketID value enable – 1: Enabled. 0: Disabled.
Output	None
Return	0 – Success. Others – Failure.
Function	Configure socket KEEP LIVE enable.

This function is used to enable or disable socket KEEPLIVE. If "enable" is 1, KEEPLIVE is enabled. If "enable" is 0, KEEPLIVE is disabled. After socket is created, KEEPLIVE defaults to be disabled. After the TCP client enables socket, call this function to enabled KEEPLIVE. For the TCP server, call this function to enable KEEPLIVE after SINT_STAT_CONNECT is generated. When KEEPLIVE is enabled, the parameter of KEEPLIVE is read. After that, the KEEPLIVE parameter change does not affect the KEEPLIVE parameter of the started socket.

4. Guidance

4.1 Initilization

WCHNET_Init is the library initialization function. For the usage of WCHNET_Init, please refer to Section 3.2. After WCHNET is initialized, the application layer needs to enable the Ethernet interrupt, and call the WCHNET_ETHIsr interrupt service function in the corresponding interrupt function. In addition, an external clock needs to be provided for the library function for periodic tasks such as refreshing the ARP list, TCP timeout, etc. Call the WCHNET_TimeIsr function to update time. The parameter passed by this function is the time difference of the last call, in milliseconds.

To sum up, after calling WCHNET_Init to initialize the library, the application layer needs to call NET_Init to initialize the Ethernet physical layer.

4.2 Configuration

The configuration information is passed to the library through WCHNET_ConfigLIB. For detailed configuration information, please refer to Section 2.1. It must be called before WCHNET_Init.

This section mainly introduces the meaning and method of configuration such as IPRAW, UDP, TCP, and memory allocation.

(1) WCHNET_NUM_IPRAW

Used to configure the number of IPRAW (IP raw sockets) connections. The minimum value is 1. For IPRAW communication.

(2) WCHNET_NUM_UDP

Used to configure the number of UDP connections. The minimum value is 1. For UDP communication.

(3) WCHNET_NUM_TCP

Used to configure the number of TCP connections. The minimum value is 1. For TCP communication.

(4) WCHNET_NUM_TCP_LISTEN

Used to configure the number of TCP listeners. The minimum value is 1. The socket for TCP listening is only used for listening. Once a TCP connection is listened, a TCP connection will be allocated immediately, occupying the number of WCHNET_NUM_TCP.

(5) WCHNET_MAX_SOCKET_NUM

Used to configure the number of sockets, equal to the sum of WCHNET_NUM_IPRAW, WCHNET_NUM_UDP, WCHNET_NUM_TCP and WCHNET_NUM_TCP_LISTEN.

(6) WCHNET_NUM_PBUF

Used to configure the number of PBUF structures. The PBUF structure is mainly used to manage memory allocation, including applying for UDP, TCP, IPRAW memory and reception/transmission memory. If the application requires more socket connections and there is a large amount of data to send and receive, this value must be set larger.

(7) WCHNET_NUM_POOL_BUF

Number of POOL BUF. POOL BUF is mainly used for data reception. If a large amount of data needs to be received, this value should be set larger.

(8) WCHNET_NUM_TCP_SEG

Number of TCP Segments. Every time WCHNET sends a TCP packet, it must first apply for a TCP Segment. If the number of TCP connections is large and the amount of data to be sent and received is large, this value should be set larger. For example, there are currently 4 TCP connections, and each receive buffer is set to 2 TCP_MSS, assuming that an ACK is performed every time a packet of data is received, the WCHNET_NUM_TCP_SEG should be configured to be greater than (4*2), which is only the most serious case. In fact, every time an ACK (or sent data) is received, the Segments of this data will be released.

(9) WCHNET_NUM_IP_REASSDATA

Number of reassembled IP packets. The size of each packet is WCHNET_SIZE_POOL_BUF, and the minimum value can be set to 1.

(10) WCHNET_TCP_MSS

Length of the maximum TCP segment. The maximum value is 1460 and the minimum is 60. Considering transfer and resources, it is recommended that this value should not be less than 536 bytes.

(11) WCHNET_MEM_HEAP_SIZE

Heap allocation memory size, mainly used for some memory allocation of indeterminate length, such as sending data. If TCP has a large batch of data to send and receive, this value should be set larger. If the application layer memory needs to be used when sending data, please refer to CFG0_TCP_SEND_COPY in this section.

(12) WCHNET_NUM_ARP_TABLE

ARP buffer. Used to store IP and MAC. The minimum value can be set to 1 and the maximum value is 0x7F. If WCHNET needs to communicate with 4 PCs, two of which send and receive data in bulk, it is recommended to set it to 4. If it is less than 2, it will seriously affect the communication efficiency.

(13) CFG0_TCP_SEND_COPY

Valid only for TCP communication.

When CFG0_TCP_SEND_COPY is 1, the send copy function is enabled. WCHNET copies the data in application layer to the internal heap memory, and then packs and sends it.

When CFG0_TCP_SEND_COPY is 0, the send copy function is disabled. For details, please refer to Section 3.25.

(14) CFG0_TCP_RECV_COPY

For debug. Enabled by default. If this value is 1, the speed is faster.

(15) CFG0_TCP_OLD_EDLETE

When CFG0_TCP_OLD_EDLETE is 1 and WCHNET applies for no TCP connection, older TCP connections will be deleted. Disabled by default.

4.3 Interrupt

The interrupts are divided to global interrupts and socket interrupts. The states of the global interrupts are defined in the following table:

Bit	Name	Description
[5:7]	-	Reserved
4	GINT_STAT_SOCKET	socket interrupt
3	-	Reserved
2	GINT_STAT_PHY_CHANGE	PHY status change interrupt
1	-	Reserved
0	GINT_STAT_UNREACH	Unreachable interrupt

①　GINT_STAT_UNREACH: Unreachable interrupt.　When the library receives the ICMP unreachable interrupt packet, it saves the IP address, port, and protocol type of the unreachable IP packet into the unreachable information list, and then this interrupt is generated. The application program can query the UnreachCode, UnreachProto and UnreachPort in the

_NET_SYS structure to get unreachable information.

② GINT_STAT_PHY_CHANGE: PHY change interrupt. It is generated when the PHY connection of WCHNET changes, for example, the PHY state changes from a connected state to a disconnected state or from a disconnected state to a connected state. The application can get the current PHY status through WCHNET_GetPHYStatus.

③ GINT_STAT_SOCK: Socket interrupt. When the socket has an interrupt event, the library generates this interrupt, and the application can get the interrupt status of the socket through WCHNET_GetSocketInt.

The states of socket interrupts are defined in the following table:

Bit	Name	Description
7	-	Reserved
6	SINT_STAT_TIM_OUT	Timeout
5	-	Reserved
4	SINT_STAT_DISCONNECT	TCP disconnect
3	SINT_STAT_CONNECT	TCP connect
2	SINT_STAT_RECV	Receive buffer not empty
[0:1]	-	Reserved

① SINT_STAT_RECV: Receive buffer not empty interrupt. It is generated after socket receives data. After the application layer receives this interrupt, WCHNET_SocketRecvLen can be used to get the length of the received data, and WCHNET_SocketRecv can be used to read the data in the receive buffer according to the length.

② SINT_STAT_CONNECT: TCP connect interrupt. Only valid in TCP mode. It is generated after TCP connects successfully. The application layer can only transfer data after that.

③ SINT_STAT_DISCONNECT: TCP disconnect interrupt. Only valid in TCP mode. It is generated after TCP disconnects. The application layer must no longer transfer data after that.

④ SINT_STAT_TIM_OUT: In TCP mode, this interrupt is generated when a timeout occurs in the process of TCP connection, disconnection, sending data, etc. It is also generated when the connection is closed internally by the library if some exception occurs. In TCP mode, the socket is disabled once this interrupt is generated, and the related configuration of the socket is cleared. So if the application layer needs to use the socket again, it must re-initialize and connect or listen.

Note: The interrupts described in this section are flags of the variable. For the convenience of expression and easy understanding, they are described as interrupts in the library and this

document, actually not the hardware interrupts of the MCU.

4.4 Socket

Socket types supported by the library: IPRAW, UDP, TCP client and TCP server.

4.5 IPRAW

Procedures to create an IPRAW socket:

① Set the protocol field, that is SourPort in IPRAW mode.

② Set the target IP address.

③ Set the start address and the length of the receive buffer.

④ Set the protocol type to PROTO_TYPE_IP_RAW.

⑤ Call the WCHNET_SocketCreat function, and pass the above settings to this function.

WCHNET_SocketCreat will find a free information list in the socket information list, and copy the above configuration to it. If no free list is found, it returns error after the creation is successful, and the free information table will be output to the application layer.

IP message structure:

Target MAC	Source MAC	Type	IP header	IPRAW data	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	Max 1480 bytes	4 Bytes

The application layer can call WCHNET_SocketSend to send data. There is no limit to the length of the data to be sent. The library loops internally and sends data automatically in turn. The maximum length of a packet that can be sent by IPRAW is 1480 bytes. If the length of the data stream written by the application layer is greater than 1480 bytes, the library will package the data stream into several IP packets for sending. An error code will be returned immediately if it fails.

When the library receives the IP data packet, it first checks whether the protocol field and the protocol field set by the socket are the same. If they are the same, the IPRAW data packet is copied to the receive buffer and a SINT_STAT_RECV interrupt is generated. After the application layer receives this interrupt, call WCHNET_SocketRecvLen to get the effective length of the current socket buffer. According to the length, the application layer calls WCHNET_SocketRecv to read the data in the socket receive buffer. The application can read all the data at one time, or read it in multiple times. Since flow control cannot be performed in IPRAW mode, it is recommended that the application layer read all data immediately after querying the receive data interrupt, to avoid being overwritten by subsequent data.

Notes on protocol field settings:
The priority of the library processing IPRAW is higher than that of UDP and TCP. If the IP protocol field is set to 17 (UDP) or 6 (TCP), there may be a possibility of conflict with other sockets, which should be avoided. Two cases are listed below:

① Socket0 is in IPRAW mode, the IP protocol field is 17, and socket1 is in UDP mode. In UDP mode, the protocol field of the IP packet is also 17, and this causes the data for socket1 communication to be intercepted by socket0 and the data cannot be received.

② Socket0 is in IPRAW mode, the IP protocol field is 6, and socket1 is in TCP mode. In TCP mode, the protocol field of the IP packet is also 6, and this causes the data for socket1 communication to be intercepted by socket0 and the data cannot be received.

The library supports the reassembled IP, but the length of the maximum reassembled packet cannot be greater than the length of the receive buffer.

4.6 UDP client

Procedures to create a UDP socket:

① Set the source port.

② Set the target port.

③ Set the target IP address.

④ Set the start address and the length of the receive buffer.

⑤ Set the protocol type to PROTO_TYPE_UDP.

⑥ Call the WCHNET_SocketCreat function, and pass the above settings to it.

WCHNET_SocketCreat will find a free information list in the socket information list, and copy the above configuration to it. If no free list is found, an error will be returned. If it is created successfully, a success will be returned, and the free information table will be output to the application layer.

UDP message structure:

Target MAC	Source MAC	Type	IP header	UDP header	UDP data	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	8 Bytes	Max 1472 Bytes	4 Bytes

UDP is a simple and unreliable transport layer protocol for data messages. The transfer speed is fast, but the data cannot be guaranteed to reach the target. The application layer must ensure the reliability and stability of the transfer.

The application layer can call WCHNET_SocketSend to send data. There is no limit to the length of the data to be sent. The library loops internally and send the data automatically in turn. The maximum length of a packet that can be sent by UDP is 1472 bytes. If the length of the data stream written by the application layer is greater than 1472 bytes, the library will package the data stream into several UDP packets for sending. An error code will be returned immediately if it fails.

When the library receives the UDP data packet, the UDP data packet is copied to the receive buffer and a SINT_STAT_RECV interrupt is generated. After the application layer receives this interrupt, call WCHNET_SocketRecvLen to get the effective length of the current socket buffer. According to the length, the application layer calls WCHNET_SocketRecv to read the data in the socket receive buffer. The application can read all the data at one time, or read it in multiple times. Since flow control cannot be performed in UDP mode, it is recommended that the application layer read all data immediately after querying the receive data interrupt, to avoid being overwritten by subsequent data.

4.7 UDP server

UDP server can receive the local port address sent by any IP address.

Procedures to create a UDP socket:

① Set the source port.

② Set the target IP address. The target address is 255.255.255.255.

③ Set the start address and the length of the receive buffer.

④ Set the protocol type to PROTO_TYPE_UDP.

⑤ Set the entry address of the receive callback function.

⑥ Call the WCHNET_SocketCreat function, and pass the above settings to it.

In UDP server mode, WCHNET adds the AppCallBack pointer in SocketInf, to distinguish the source IP and source port of the data packet. After UDP receives the data, it notifies the application layer of the source IP and source port of the data packet through AppCallBack. In the AppCallBack function, the application layer should read all the data, and WCHNET initializes all related variables of the receive buffer after calling back the AppCallBack.

If data is not received through callback, please be sure to initialize AppCallBack in SocketInf to 0.

4.8 TCP client

Procedures to create a TCP client socket:

① Set the source port.

② Set the target port.

③ Set the target IP address.

④ Set the start address and the length of the receive buffer.

⑤ Set the protocol type to PROTO_TYPE_TCP.

⑥ Call the WCHNET_SocketCreat function, and pass the above settings to it.

⑦ Call the WCHNET_SocketConnect function, and TCP initiates connection.

WCHNET_SocketCreat will find a free information list in the socket information list, and copy the above configuration to it. If no free list is found, an error will be returned. If it is created successfully, a success will be returned, and the free information table will be output to the application layer.

After calling WCHNET_SocketConnect, the library actively initiates a connection request to the remote end. After connected successfully, a SINT_STAT_CONNECT connection interrupt is generated. If the remote end is not online or other exceptions occur, the library automatically retries. Both the number of retries and the retry period can be set in the application layer. If it is still unsuccessful, the library automatically closes the socket, and a SINT_STAT_TIM_OUT timeout interrupt is generated. Only after the connection interrupt is generated, the application layer can use this socket to send/receive data.

TCP message structure:

Target MAC	Source MAC	Type	IP header	TCP header	TCP data	CRC32
6 Bytes	6 Bytes	2 Bytes	20 Bytes	20 Bytes	Max. 1460Bytes	4 Bytes

TCP provides reliable byte stream service for connection.

Unack Segment: TCP messages that are not sent successfully.

Two sending methods in WCHNET TCP mode:
1: Copy. Copy the user's data to Mem_Heap_Memory for sending. The total length of the data is not limited. If the length is greater than WCHNET_TCP_MSS, WCHNET divides the data into several TCP packets with the size of WCHNET_TCP_MSS for sending. It is generally used when the number of sockets is small and the amount of data to be sent is relatively small. The application layer only needs to call the WCHNET_SocketSend function.
2: Non-copy. Send directly using the user buffer. The maximum length of data is WCHNET_TCP_MSS. It is generally used in the case of a large number of sockets, a large amount of data to be sent, and strict requirements on RAM.

Note when using the non-copy method:
Call WCHNET_SocketSend(sockeid,tcpdata,&len) to send, len must not be greater than TCP_MSS, tcpdata cannot be a buffer allocated locally or in the stack, and the application layer can no longer use the tcpdata buffer after calling WCHNET_SocketSend until WCHNET notifies the application layer that the data segment of this buffer is successfully sent.

WCHNET notifies the application layer that the data segment is successfully sent through AppCallBack. The prototype of AppCallBack is as follows:

  void (*pSockRecv)(**struct** _SCOK_INF *, uint32_t, uint16_t, uint8_t *, uint32_t);

In TCP mode, AppCallBack is used to notify the application layer of the number of socket Unack Segments, socinf is the socket information, and len is the number of Unack Segments. After the application layer gets the number, call WCHNET_QueryUnack to get the information of these messages. If tcpdata is not sent successfully, WCHNET writes tcpdata (buffer address) into addrlist. For details about WCHNET_QueryUnack, please refer to Section 3.25. For the configuration of the sending method, please refer to Section 4.2.

In TCP mode, if the data transmission fails, a SINT_STAT_TIM_OUT interrupt will be generated, and the application layer should close the socket.

When the library receives the TCP data packet, the TCP data packet is copied to the receive buffer and a SINT_STAT_RECV interrupt is generated. After the application layer receives this interrupt, call WCHNET_SocketRecvLen to get the effective length of the current socket buffer. According to the length, the application layer calls WCHNET_SocketRecv to read the data in the socket receive buffer. The application can read all the data at one time, or read it in multiple times. In TCP mode, every time the application layer calls WCHNET_SocketRecv, the library will copy the received data to the receive buffer of the application layer, and then notify the remote end of the current window size. For details about WCHNET_SocketRecv, please refer to Section 3.18.

4.9 TCP server

Procedures to create a TCP server socket:

① Set the source port.

② Set the protocol type to PROTO_TYPE_TCP.

③ Call the WCHNET_SocketCreat function, and pass the above settings to it.

④ Call the WCHNET_SocketListen function, and TCP starts listening.

Through the above procedures, a listening socket can be established. This socket only listens for client connections and itself does not send/receive data, so there is no need to set a receive buffer.

If a client connects successfully, the listening socket will find a free list from the socket information list. If no free list is found, it disconnects. If a free list is found, it initializes this list and write the information such as target IP, source port and target port into this list, and generate a SINT_STAT_CONNECT connection interrupt. After the application layer software receives this interrupt, it should immediately call WCHNET_ModifyRecvBuf to allocate a receive buffer for this connection. If the application software establishes multiple servers, it can determine which server is connected by querying the source port in the socket information list.

For the data structure, the process of sending data and receiving data, please refer to the TCP client mode.