Software Engineering and Applications 软件工程与应用, 2021, 10(3), 284-294
Published Online June 2021 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2021.103032

Hans汉斯

基于RT_Thread中W5500的驱动配置、应用与优化

贺松杰，张　燕，李　苗

郑州科技学院，河南　郑州
Email: 18625515230@163.com

收稿日期：2021年4月7日；录用日期：2021年6月9日；发布日期：2021年6月16日

摘　要

W5500是工业级以太网芯片，本文介绍了W5500芯片的特点，通信方式，结合Rt-Thread Studio对RT-Thread操作系统下的W5500驱动结构进行分析，并在STM32F103VET6搭载RT-Thread操作系统，在RT-Thread Studio环境中开发W5500网络设备，并详细介绍在该系统下添加W5500驱动。通过实验验证W5500在RT-Thread中存在的驱动漏洞问题，并进行驱动优化，理论结果可行，实际验证结果可行。

关键词

RT-Thread，W5500，RT-Thread Studio，驱动漏洞，驱动优化

Drive Configuration, Application and Optimization of W5500 Based on RT_Thread

SongJie He, Yan Zhang, Miao Li

Zhengzhou University of Science and Technology, Zhengzhou Henan
Email: 18625515230@163.com

Received: Apr. 7th, 2021; accepted: Jun. 9th, 2021; published: Jun. 16th, 2021

Abstract

W5500 is an industrial-grade Ethernet chip. This article introduces the characteristics and communication methods of the W5500 chip, combines Rt-Thread Studio to analyze the W5500 drive structure under RT-Thread operating system, equips with RT-Thread operating system in STM32F103VET6, develops W5500 network equipment in RT-Thread Studio environment, and in-

文章引用: 贺松杰, 张燕, 李苗. 基于 RT_Thread 中 W5500 的驱动配置、应用与优化[J]. 软件工程与应用, 2021, 10(3): 284-294. DOI: 10.12677/sea.2021.103032

troduces in detail the addition of W5500 driver under this system. Experiments are conducted to verify the driver vulnerability of W5500 in RT-Thread, and to optimize the driver. The theoretical results are feasible, and the actual verification results are feasible.

Keywords

RT-Thread, W5500, RT-Thread Studio, Driver Vulnerability, Driver Optimization

Copyright © 2021 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

物联网(Internet of Things, IoT)概念广为普及，嵌入式设备联网是大势所趋。RT-Thread 其实就是一个实时支持多任务的物联网系统。物联网操作系统是以操作系统内核为基础，包括文件系统和图形库等较为完整的中间组件，具备低功耗、安全、通信协议支持和云端接入的软件平台。

本文以基于 CORTEX-M3 内核的 STM32F103VET6 为例，通过 SPI 总线驱动 W5500，搭载 RT-Thread 操作系统，详细介绍如何在 RT-Thread Studio 集成开发环境中加载 W5500 驱动，编写应用进行 TCP/IP 通信测试。并介绍了 RT-Thread 中关于 W5500 的以太网驱动存在一个关键的问题：处理连续大量的数据时驱动采用的驱动方式是利用硬件的下降沿来驱动，而 W5500 在遇到连续的事件来不及处理时中断引脚输出持续的低电平，造成 STM32 处理器无法对新的事件进行处理，从而造成网络的假死状态。本文分析出原因并在接受任务时新建定时任务来监听引脚电平，在持续的低电平后会模拟中断事件来发送任务信号量，以解决连续数据引起的网络假死现象。

2. W5500 介绍与硬件连接

2.1. W5500 硬件介绍

W5500 是一款全硬件 TCP/IP 嵌入式以太网控制器，为嵌设计与研发入式系统提供了更加方便的互联网连接方案[1]。内部框架图如下图 1。

W5500 提供了一套标准的 4 线制 SPI 接口，接口最大通信速度可达 80M，本系统主机选用 STM32F103VET6 作为 SPI 主机，SPI 接口速率最大 18M。可以直接连接 W5500 进行通信。W5500 外围电路比较简单，除了需要提供一个 25M 的时钟外，一个标准的网络接口转换器，其它只需要少许电阻电容即可工作。与 stm32 的连接如下图 2 所示。

2.2. 在 RT-Thread Studio 中使能 W5500 及配置相关接口

W5500 提供标准的 4 线制通信接口，加上中断和复位引脚总共占用 6 个 IO 口。W5500 采用 SPI 接口通信。SPI 系统可以直接与各个厂家生产的多种外围器件直接连接，工作在主从模式[2]。SCLK 提供时钟脉冲，MOSI 和 MISO 和 SCLK 的脉冲下进行数据传输。SPI 通信在主机和从机之间分别提供 2 个方向的数据线，因此可以进行全双工的数据传输。CPOL 和 CPHA 代表不同的时钟极性，可以为 0 或者是 1，因此提供 4 种时钟极性。W5500 的 SPI 接口通信支持模式 0 与模式 3，数据都是在上升沿锁存，下降沿输出。如下图 3 所示。

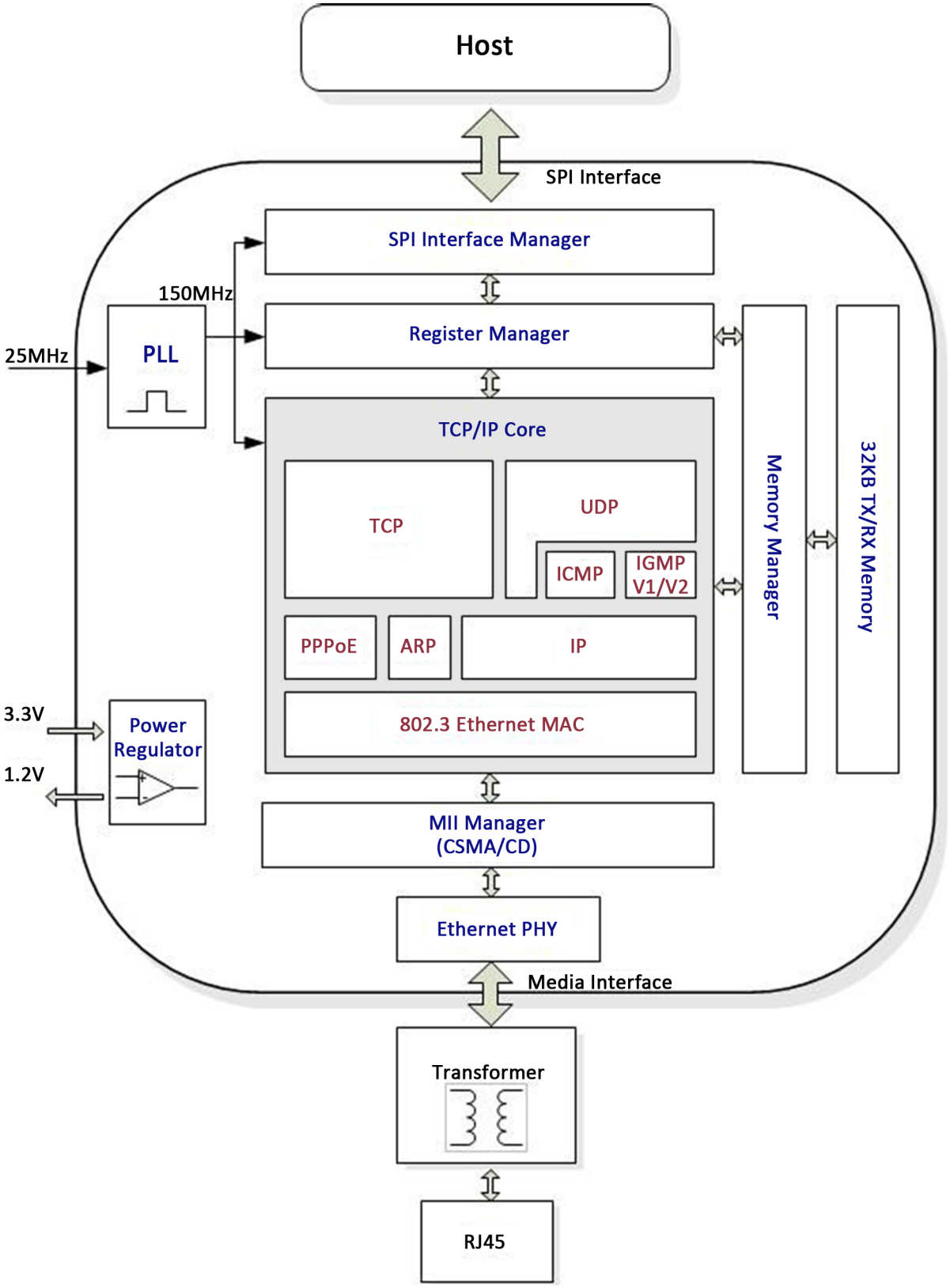

Figure 1. W5500 internal frame diagram
图 1. W5500 内部框架图

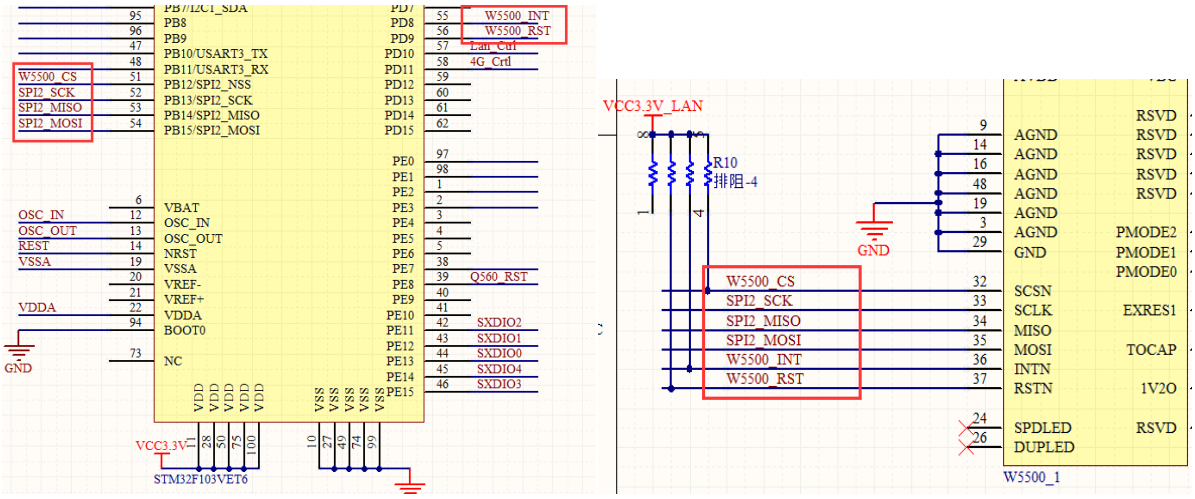

Figure 2. The interface circuit between W5500 and STM32
图 2. W5500 与 STM32 的接口电路

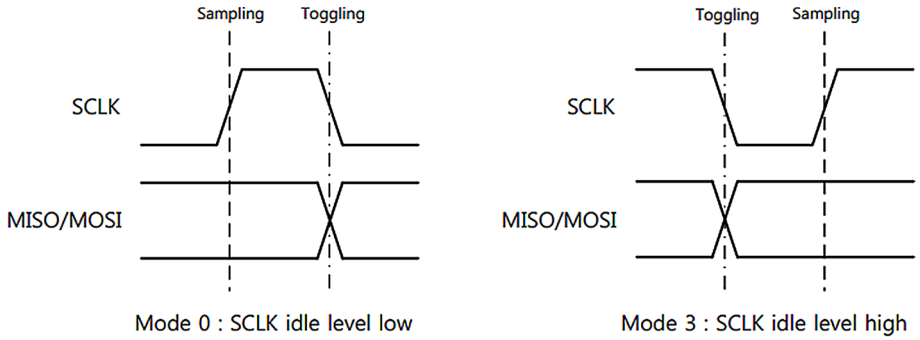

Figure 3. W5500 SPI communication mode
图 3. W5500 的 SPI 通信方式

2.2.1. RT Thread Studio 简介

RT Thread Studio 是一站式的 RT-Thread 开发工具，通过简单的图形化配置系统及丰富的软件包组件资源，让物联开发变得简单。它集成了工作创建与管理、代码编辑、SDK 管理、RT-Thread 配置、构建配置、调试配置、程序下载和调试功能[3]。而且它还支持直接导入常用的 KEIL 工程和 IAR FOR ARM 工程，最新版本还支持 STMCUBEMX，方便一站式配置 ST 旗下的设备，使之前独立的嵌入式设备快速接入互联网[4]。

2.2.2. W5500 的 IO 配置

在 RT-Thread Studio 中使能 W5500，只需要在 RT-Thread Settings 中添加，图形化的界面使对应的配置比较简单。但是 RT-Thread 提供的引脚编号需要和芯片的引脚号区分开来，它们是不同的概念，引脚编号由 PIN 设备驱动的程序定义，不同的 STM32 型号的 IO 口对应不同的编号。我们可以使用函数 GET_PIN (port, pin)来获取我们需要的引脚编号。W500 除 SPI 的 4 根通信总线，还需要 W5500_RST 与 W5500_INT 引脚，它们分别对应 W500 的复位脚与中断脚，作用十分重要。由图 1~2 所示它们分别占用 STM32 的 PD8 与 PD9 脚，所以我们利用宏定义来配置引脚：#define WIZ_RST_PIN GET_PIN(D, 9); #define WIZ_IRQ_PIN GET_PIN (D, 8)。我们也可以在 drv_gpio.c 中找到对应 PORTD 脚的序号，图 4 为 RT_Thread 中 GPIO 的引脚定义。

由图 4 可知 PD8 与 PD9 分别对就的编号是 56、57 号，找到对应引脚之后我们就可以在 RT Thread Studio 中使能对应的功能。

```
#if defined(GPIOD)
    __STM32_PIN(48,  D, 0),
    __STM32_PIN(49,  D, 1),
    __STM32_PIN(50,  D, 2),
    __STM32_PIN(51,  D, 3),
    __STM32_PIN(52,  D, 4),
    __STM32_PIN(53,  D, 5),
    __STM32_PIN(54,  D, 6),
    __STM32_PIN(55,  D, 7),
    __STM32_PIN(56,  D, 8),
    __STM32_PIN(57,  D, 9),
    __STM32_PIN(58,  D, 10),
    __STM32_PIN(59,  D, 11),
    __STM32_PIN(60,  D, 12),
    __STM32_PIN(61,  D, 13),
    __STM32_PIN(62,  D, 14),
    __STM32_PIN(63,  D, 15),
```

Figure 4. PORTD pin definition in RT-Thread
图 4. RT-Thread 中 PORTD 引脚定义

2.2.3. SPI 配置

RT_Thread 使用虚拟的 SPI 总线来描述挂载在 SPI 上的不同设备。例如 SPI10 代表 SPI1 总线上的 0 号设备，由于我们使用的 STM32F103VET6 的 SPI2 功能，并且 SPI2 总线上只有一个设备，因此在配置 W5500 时我们在 spi device name 一栏填入 spi20 即可。且我们打开 W5500 的 DHCP 功能和 DNS 功能，以便可以动态获取 IP 地址与进行域名解析。最终配置完成如下图 5 所示。

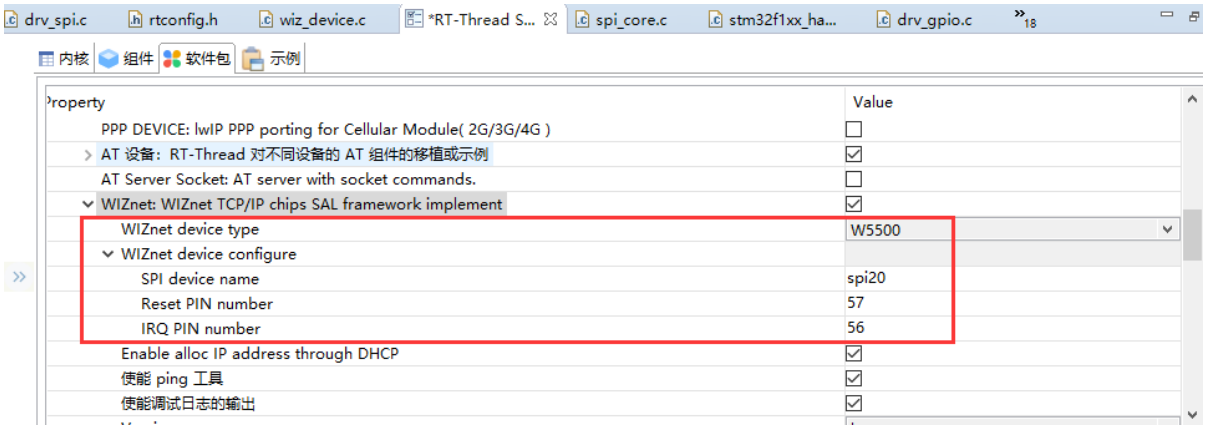

Figure 5. RT-Thread Studio W5500 configuration
图 5. RT-Thread Studio W5500 配置

2.3. 初始化 SPI 并挂载 SPI2 总线

STM32F103VET 的 SPI2 来和 W5500 需要使用通信，在 W5500 的初始化时首先会进行 SPI 的挂载，所以我们需要在 RT-Thread Studio 中初始化 STM32F103VET6 的 SPI2 口，集成环境中已经集成了 STM32CubeMX，开发人员也可以 STM32CubeMX 对芯片进行相应的设置生成 stm32f1xx_hal_msp.c 文件，里面有对系统的一些初始化函数，例如对 SPI 的初始化函数：void HAL_SPI_MspInit (SPI_HandleTypeDef* hspi)；在 W5500 调用 SPI2 进行初始化时，会自动找到该函数进行初始化。

在外面 SPI 和引脚配置之后，我们又加载了 SPI2 的初始化函数，进行下载和调试后通过串口打印出的信息仍提示：You should attach spi20 into SPI bus firstly。说明 SPI2 总线没有被正常的挂载，系统并没有在配置在完 W5500 自己挂载 SPI2 总线，所以我们设计了把 SPI2 的总线挂载子程序：

```
int w5500_spi_attach(void){
spi_cs.GPIO_Pin = GPIO_PIN_12;
spi_cs.GPIOx = GPIOB;
rt_pin_mode(GET_PIN(B,12), PIN_MODE_OUTPUT);
rt_spi_bus_attach_device(&spi_dev_wiz,WIZ_SPI_DEVICE,
                         "spi2",(void*)&spi_cs);}
```

子程序对 SPI 总线中 CS 引脚做了初始化操作之后，通过 INIT_DEVICE_EXPORT(w5500_spi_attach) 把该函数挂载到 RT-Thread 的系统初始化链表中，再进行下载和调试发现 W5500 工作正常，连接串口提示：[I/wiz] RT-Thread WIZnet package (V2.0.0) initialize success。初始化完成之后我们需要编写应用来测试通信。

3. SAL 层系统标准接口

在编写应用测试通信之前，我们要了解在编写应用的一些函数。W5500 使用 SAL 函数进行应用层的编写，SAL 层的应用大大提高了网络应用的兼容性问题，降低了嵌入式网络设备的开发难度与周期。提供 Socket 层面 TLS 加密传输特性，更好的保护嵌入式网络的数据安全，防止数据泄露。

3.1. SAL 层网络框架

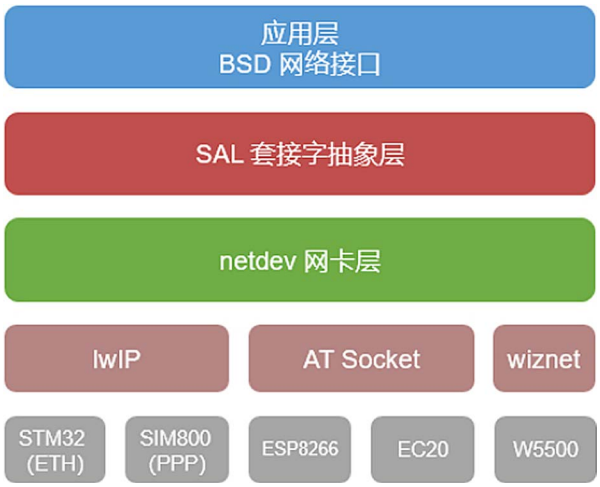

Figure 6. SAL layer network framework
图 6. SAL 层网络框架

如图 6 的框架示意图，最顶层的为网络应用层，提供一套标准 BSD Socket api 接口函数，如常用的 socket、connect 等函数。往下一层即为 SAL 套接字抽象层，通过它 RT-Thread 系统能够适配下层不同的网络协议栈，并提供给上层统一的网络编程接口，方便不同的网络协议栈接入。SAL 层为上层应用提供 accept、send、recv 等函数。

再往下则为 netdev 网上层，解决多网卡情况设备网络连接和网络管理相关问题。通过网卡层方便用户统一管理各个网卡信息及网络连接状态，并且可调用统一的网上调试命令接口。

第四层即为协议栈层，该层包括了常用的 TCP/IP 协议栈，例如轻量级的 Lwip 协议栈，以及用作 AT 指令进行相关操作的 AT Socket 网络功能协议栈，再有就是 W5500 上用的 wiznet 网络协议栈，这些协议栈能直接和硬件进行接触，完成从网络层到传输层的数据转化。

统一标准的的 RT-Thread 网络接口 BSD Socket API，让不同的网络硬件对应用层变得透明，同样的应用不同的硬件只需要对硬件层做一些改动即可完成硬件升级。

3.2. 在 RT-Thread Studio 中打开 SAL

使能 SAL 功能方法列举以下两种，1、只需要在 rtconfig.h 文件中添加
`#define` RT_USING_SAL

1、rt-thread studio 提供了图形化配置界面，在 rt-thread studio 中使能只需要首先点开工程中的 RT-Thread Settings，找到套接字抽象层，点击使能套接字抽象层即可(如图 7)：

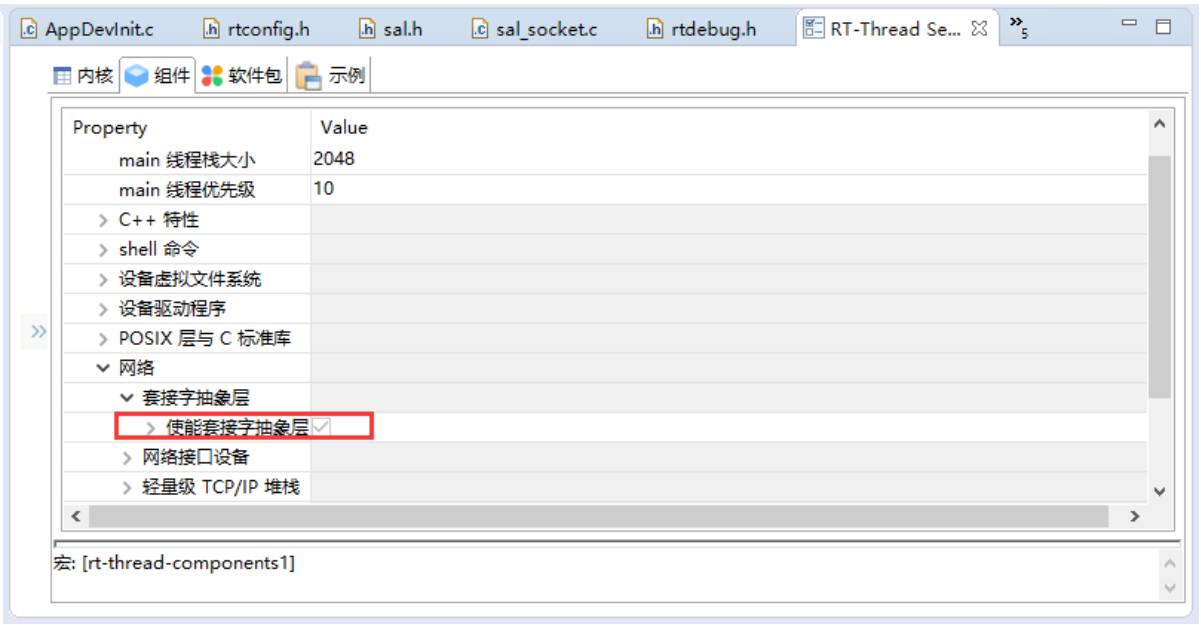

Figure 7. SAL layer enable setting
图 7. SAL 层使能设置

3.3. SAL 层 BSD Socket API 接口函数

下面我们列举几个需要在应用中用到的 API 接口函数。

3.3.1. 连接初始化和关闭函数

- 创建套接字：int socket (int domain,int type,int protocol)。此函数调用成功后会返回对应的 int 型套接字描述符。如果小于 0 则说明没有空闲的 socket。domain 参数为对应的协议类型，目前支持 IPV4 的 AF_INET 和 IPV6 的 AF_INET6。第二个参数为协议类型，分别支持 SOCK_STREAM(流套接字)、SOCK_DGRAM(数据报套接字)、SOCK_RAW(原始套接字)。
- 绑定套接字：int bind (int s, const struct sockaddr *name, socklen_t namelen)此函数的返回值小于 0 则说明绑定失败，其它情况都说明绑定成功。第 1 个参数 s 为 socket 的描述符。第 2 个参数为 sockaddr 的结构体指针 name。第 3 个参数为代表 sockaddr 的结构体长度。
- 关闭套接字函数：int closesocket(int s)。此函数仅有一个参数，即为套接字描述符。用于关闭指定的

socket 连接。

3.3.2. 设备端连接函数

连接函数：int connect (int s, const struct sockaddr *name, socklen_t namelen)用于主动建立一个 socket 连接。当设备工作在 TCP 客户端模式和 UDP 模式时，需要先执行此函数进行连接。

3.3.3. UDP 通信函数

UDP 为无连接的通信函数，和 TCP 通信不同，在通信前不会主动建立连接，所以数据在发送时需要把指定目的地址，接收数据时获取源地址。

UDP 发送函数：int sendto (int s, const void *dataptr, size_t size, int flags, const struct sockaddr *to, socklen_t tolen)；UDP 通信为无连接的通信，在发送数据前无需提前建立连接。函数的第一个参数 s 为 socket 描述符。第二个参数 dataprt 和第三个参数 size 分别是要发送的数据指针与长度。flags 通常为 0。to 为目标地址的 sock 结构体指针。最后一个参数 tolen 为目标地址 to 的指针长度。

UDP 接收函数：int recvfrom (int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)；第一个参数 s 为 socket 描述符。第二个参数 mem 和 len 分别为接收指针和接收长度。第四个参数 flags 通常为 0。from 为 sockaddr 的结构体指针。fromlen 为接收地址的长度。

UDP 接收函数同样会阻塞进程，直到接收到数据进程转为就绪态。

4. 应用层网络能力测试

在应用层编写的时候首先我们要确定目的主机的 IP 地址和端口，然后我们开始进行一些必要的网络层初始化，然后才能开始进行通信。首先进行应用层的初始化，我们采用 UDP 进行通信，目的端口选为 2000，相关的初始化如下：

4.1. UDP 通信初始化

UDP (User Data Protocol，用户数据报协议)是一种无连接的不可靠的通信，因为在传输数据之前源端和终端不建立连接，当它想传送数据时就简单地去抓取数据，并尽可能地把它扔在网络上。因为不需要建立连接和维护连接状态，所以可以很方便的实现一台服务机向多台客户机的广播通信。而且网络开销小，吞吐量仅受应用层生成数据的速率，带宽以及源终端的主机性能的限制。

在 rt-thread 中初始化 UDP 的流程：

1) 获取网卡对象对象信息：netdev = netdev_get_by_name ("W5500")；

2) 申请 socket 并设定为 UDP 通信：NetSockFd = socket (AF_INET, SOCK_DGRAM, 0))；

3) 初始化客户端信息包括本网卡的 IP 和通信端口，然后进行绑定，绑定代码：

Bind (NetSockFd, (struct sockaddr *)&client_addr, sizeof (struct sockaddr))

绑定完之后即可进行通信。

4.2. UDP 通信测试

我们进行简单的 UDP 通信，只需要保证 PC 端和设备在同一个局域网下，就可以进行广播通信，我们已经设定本机通信端口为 3000，目的端口为 2000。打开网络调试助手，设定通信方式为 UDP，目标地址设为 255.255.255.255，目标端口设定为 3000。进行广播查找设备。此时我们做了一套简单的通信协议，只有符合协议的内容才会被处理并返回信息。广播查找协议内容如下图 8。

设置完网络助手并开启定时发送，设备发送和回复都设置为 HEX 发送与接收，通信界面如下图 9，调试助手在收到消息回复时会把目的 IP 自动设置为接收数据的 IP，此时方便在未知设置的 IP 情况下来

获取对方消息：

终端发送的广播数据格式（目标端口一定为 3000）

帧头	目标设备类型	帧类型	命令	数据长度	数据	校验和	数据 ID
55 AA	01	07	01（可忽略	02	端口号 + 状态	1 字节	02

主机向终端回复应答帧，终端记下主机的 IP 和端口

应答帧格式如下：

帧头	目标设备类型	帧类型	命令	数据长度	数据	校验和	数据 ID
55 AA	02	07	01（可忽略	06	180805115308	1 字节	02

Figure 8. Application layer test protocol
图 8. 应用层测试协议

Figure 9. UDP communication interface
图 9. UDP 通信界面

4.3. 问题发现、分析及优化

4.3.1. 问题发现

当我们在客户端设置自动发送时长为 10 ms，在长时间的大量数据通信时，W5500 会出现网络死机现象，数据发送正常，但是收不到设备端(W5500)的回复。在进行多次测试后发现这个问题的会多次重现。在使用进行万用表对相关引脚进行测试之后发现问题。

4.3.2. 问题分析

总结问题原因：RT-Thread 中 W5500 接收数据过程如下图 10 所示。

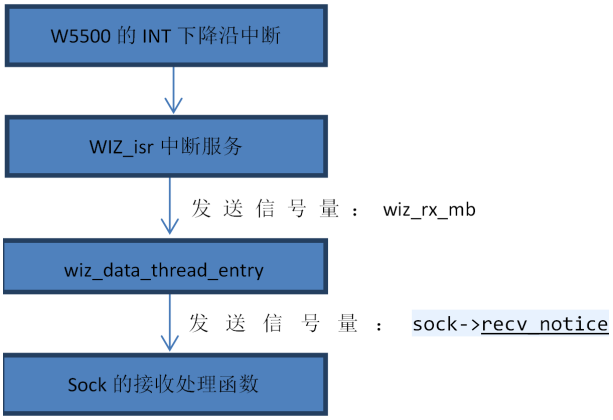

Figure 10. W5500 receiving data flow chart
图 10. W5500 接收数据流程图

如图 10 所示，W5500 在收到数据或者发送成功后会产生中断，STM32F103VET6 是根据 W5500 产生的下降沿中断来发送信号量。并来执行 W5500 的数据处理任务。但是当 W5500 的数据过多，而 STM32F103VET6 来不及处理时 W5500 的 INT 脚一直输出低电平，造成 STM32 无法处理新的数据，原因是 STM32F103VET6 的中断边沿触发只支持上升沿和下降沿中断。所以一旦 W5500 的接收发送或者其他事件产生中断时来不及处理时就会产生持续的低电平，造成 STM32F103VET6 无法处理新的数据而产生网络假死状态。

4.3.3. 问题优化

根据问题原因，我们添加中断监控信号量，并添加中断监控任务，一旦设备接收到新的数据时首先会进行中断，然后执行数据处理任务，之后会清除对应的中断，中断引脚会恢复高电平。我们在数据处理后引脚变高时向中断监控任务发送一个信号量。中断监控任务收到信号量检测 W5500 中断引脚电平，如果还为低电平，则会假装中断任务向 W5500 的数据处理任务发送一个信号量，让数据处理任务来进行一次处理并清中断的动作。就可以解决因为 W5500 的中断脚持续输出低电平而造成的网络假死状态。监控任务的信号发送在 WIZ.C 中的 static void wiz_data_thread_entry (void *parameter)中添加：rt_sem_release (Sem_NetNoDie)；//释放信号量，告诉监控任务进入了一次中断。监控服务程序不再添加。本实验需要通过长期运行来验证结果，笔者已经验证成功，具体测试方式如章节 3.2 中所示，只需要在软件界面勾选上循环发送，即可通过长时间的发送和响应来验证结果。具体程序处理流程如下图 11。

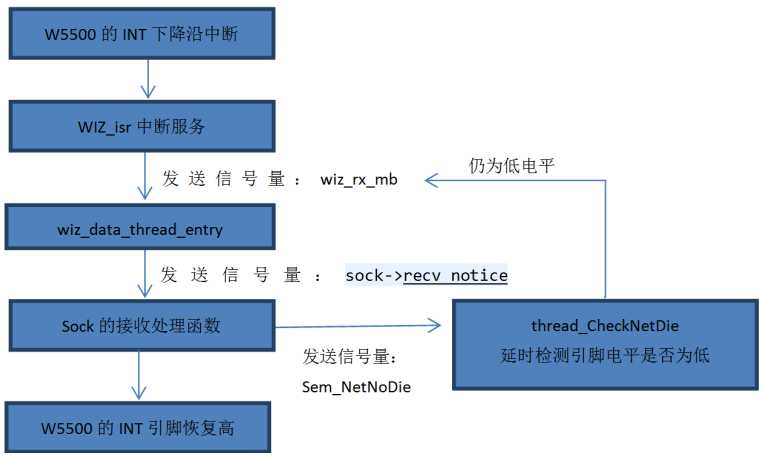

Figure 11. W5500 receiving optimization flowchart
图 11. W5500 接收优化流程图

5. 结语

通过程序连续工作验证，对 RT-Thread 操作系统下 W5500 的优化解决了驱动本身的一些漏洞问题。RT-Thread 的移植和应用都科学合理，移植方便。在物联网发展迅速的今天，本文为开发 RT-Thread 下相关嵌入式设备提供一种方案，对潜在的问题分析优化，有助于用本方案开发嵌入式设备的人员少走一些弯路，更好的推动 RT-Thread 的发展。

参考文献

[1] 刘纯虎, 付斌, 盛庆华. 基于 STM32 的微型 USB-CAN 适配器开发[J]. 计算机测量与控制, 2013, 21(4): 996-999.

[2] 高培, 何栋炜, 李文翔, 等. RT-Thread 的 SPI 总线驱动结构分析、移植及应用[J]. 单片机与嵌入式系统应用, 2017(12): 48-53

[3] RT-Thread 文档中心[DB/OL]. https://www.rt-thread.org/document/site/, 2020.

[4] 张玉鲁. 基于 RT-Thread 的串口转以太网数据分路系统设计[D]: [硕士学位论文]. 海口: 海南大学, 2020.