# 一种基于 uIP 协议栈的多应用层协议支持方案的设计

周绍文

（东方电子股份有限公司，山东 烟台 264000）

**摘要：**提出一种基于免费开源 uIP 协议栈的多应用层协议支持方案的设计。在充分利用 uIP 协议栈的基础上，通过分层设计，自定义接口完成层间调用等设计方法，实现了基于 uIP 协议栈的多应用层协议支持。该设计方案已经应用于公司新一代无操作系统的电力系统二次侧保护设备的通信系统平台中，对完成该保护系统对外基于 TCP/IP 通信要求提供了可靠的支持。

**关键词**：嵌入式应用； uIP；TCP/IP 协议栈；应用层协议；分层设计；接口

## Design of support scheme of multi-application protocols based on uIP stack

ZHOU Shao-wen

(Dongfang Electronics Co.Ltd, Yantai 264000, China)

**Abstract:** This paper introduces a design of support scheme of multi-application protocols based on the free and open source uIP stack. With the layered design, it implements a way of support of multi-application protocols based on uIP stack by the interface call between layers. The design has been applied on the communication plat of a new substation secondary protection equipment without embedded operation system on power system, which supplies the system with a reliable support of communication based on TCP/IP stack.

**Key words:** embedded application; uIP; TCP/IP protocol stack; application protocols; layered design; interface

中图分类号： TM76 文献标识码：A 文章编号： 1674-3415(2010)09-0109-04

## 0 引言

TCP/IP 协议栈在 Internet 上的成功应用，使其已经成为全球事实上的通信标准。广泛应用于自动化领域的嵌入式系统也迫切需要支持 TCP/IP 协议，或者在 TCP/IP 协议栈的基础上实现互联互通，以防成为信息孤岛。随着应用需求的增加和可靠性设计要求的提高，对嵌入式系统多应用层协议的支持也提出了要求。现在的主流嵌入式操作系统，例如 VxWorks, pSOS 等均支持 TCP/IP 协议栈，用户通过适当的网络管理方式可以实现对多应用层协议的支持。但是此类软件销售价格很高，而且有的嵌入式操作系统的 TCP/IP 协议栈甚至需要单独付费。某些嵌入式系统应用，或者考虑产品成本、硬件资源等方面的限制，根本不适合采用嵌入式操作系统。

开源代码 uIP 是瑞典计算机科学学院（Swedish Institute of Computer Science）Adam Dunkels 等人开发的微型 TCP/IP 协议栈（light-weight TCP/IP Stack），具有如下特点：

(1) 良好的文档和源代码注释——几乎每一行代码都有注释；

(2) 代码非常少；

(3) 占用非常少的内存；

(4) 支持 ARP,SLIP, IP, UDP,ICMP(ping)和 TCP 协议；

(5) 提供一套实例程序：web 服务器，web 客户端，电子邮件发送程序（SMTP 客户器），Telnet 服务器，DNS 主机名解析程序；

(6) 同时活动的 TCP 连接数没有限制，在编译时候可以设置；

(7) 可免费用于商业和非商业用途；

(8) TCP 和 IP 协议遵循 RFC 标准，包括流控制，片断分割和重传超时估算。

uIP 源码可以从网址 http://www.sics.se/□adam/uip/下载，目前版本为 1.0。全部源码仅 50 左右个文件（包括各种示例程序文件）、数千行代码，对于 TCP/IP 初学者和需要基于 TCP/IP 协议栈开发产品的相关人员，研究或应用都比较容易。

根据工程需要，本文介绍一种基于 uIP 协议栈的多应用层协议支持方案的设计。

## 1 通信系统层次

通过研究 uIP 发现，基于 uIP 的应用层协议是由 uIP 驱动，根据 uIP 设置的标志完成数据发送或接收的。根据 uIP 的这一特点，设计如下基于 uIP 协议栈的通信系统层次。

表 1 通信系统层次

Tab.1 Layers of communtcation system

| 应用层协议 |
| --- |
| 抽象层 |
| uIP |

应用层协议完成基于 TCP/IP 协议栈的标准或自定义协议，例如电力系统行业通信国际标准 IEC60870-5-104, 中国南方电网继电保护故障信息系统通信与接口规范等，该层根据具体的通信协议要求完成数据分析、数据组织等，由 uIP 层通过抽象层驱动。该层的具体实现根据具体的通信协议标准不同而不同，限于篇幅，本文不对该层的具体实现做详述。

抽象层隔离应用层协议与 uIP 层。该层目的是简化应用层协议编程，使应用层专注于具体的应用层协议细节处理，不必关心其下层（主要是指 uIP 标志）的具体实现。该层负责解析 uIP 标志进而驱动应用层协议具体过程，既对应用层协议抽象 uIP 功能，也反过来对 uIP 抽象应用层协议功能。抽象层的实现需要仔细研究 uIP 各标志用途，以驱动应用层协议的适当操作。同时，抽象层也对系统提供统一的 uIP 调度接口，该接口内封装 uIP 的各项操作。

uIP层，完成 uIP 初始化、以太网物理层的初始化与驱动管理，TCP/IP 协议栈基本功能，包括 ARP，SLIP，IP，UDP，ICMP(ping)和 TCP 协议等。完成以太网物理层相关工作：以太网芯片初始化，驱动，中断管理，数据的物理接收与发送等功能。建议以太网物理层通过中断收发数据，以提高系统效率，本文中不对 uIP 的以太网物理层做详细描述。该层通过抽象层对系统提供统一的调度接口完成 uIP 接收数据、分析数据、发送数据、驱动应用层协议接收、发送数据，以及 uIP 的定时调度。

关于 uIP 的以太网物理层的实现，可以参阅本人的另一篇论述《一种基于 uIP 协议栈多网卡支持方案的设计》。

## 2 通信流程

通过研究 uIP 源码提供的调度机制可知：uIP 可以通过接收数据和定时两种机制驱动；

通过 1 通信系统层次的分析可知：应用层协议收发数据是通过 uIP 驱动抽象层进而驱动应用层协议实现的。

因此，具体的通信流程可分析如下：

(1) 系统通过抽象层提供的统一接口调度 uIP。

(2) uIP 读取以太网数据包到 uIP 自身缓存。

(3) uIP 解析该数据包，有需要 uIP 发送响应数据的，例如 ARP 报文、ICMP 报文等，uIP 直接组织数据，调用物理层发送数据接口，完成发送，此部分数据分析过程不必应用层协议参与。

(4) uIP 检测到应用层协议数据时，根据以太网数据包头定位到相应连接，调用抽象层接口宏 UIP_APPCALL。该接口根据 uIP 相应标志调度应用层协议相应接口，完成应用层协议接收、分析、发送数据等。应用层协议接收数据时需要注意的是，uIP 层仅保证是完整的 IP 报文，而不负责该报文在应用层协议中帧的完整性，故应用层续帧机制由应用层协议自己完成(所有基于 TCP/IP 协议栈的应用层协议均应注意这点)；应用层发送数据时需要注意的是，要确定 uIP 缓存中是否有数据（根据分析 uIP 源码，uIP 调用上层接口后，uIP 缓存中的数据应被上层读走，但是需要防错）。另需注意：应用层协议的发送也是在该接口中调用实现的。

(5) uIP 检测到应用层有需要发送的数据，调用物理层发送数据接口完成数据发送。

系统通过抽象层提供的统一接口调度 uIP 层时，除了完成上述收发数据过程外，还处理 uIP 各定时器，例如 TCP 各定时器，检测各连接的各项超时等。

总结该流程分析，物理层的一帧以太网数据包被 uIP 读取后，如果有应用层协议数据，会直接到达应用层协议处理，续帧功能由应用层协议负责。

## 3 抽象层接口

通过上述分析，抽象层需要就其上层——应用层协议和其下层——uIP 层分别提供接口。接口分析如下：

### 3.1 对应用层协议接口

(1) 抽象层需要调用应用层协议收发接口完成数据分析处理与组织发送。如何在这层获得相应应用层协议的接口，与系统关于应用层协议的设计思路紧密结合，本文介绍一种注册机制以实现对多应用层协议的支持。采用注册机制，可以在增加或修改应用层协议时，不必修改或尽量少的修改抽象层及其下层代码。多应用层协议注册机制结合 uIP 示

例如下：

利用 uIP 提供的关于 TCP 连接的结构 uip_conn 的定义中公开给上层的配置接口 uip_tcp_appstate_t，存储应用层协议信息，当需要创建 UDP 连接时，与此类似，不再赘述。示例如下：

```
struct protocolApp_state{
    void
(*RxdMonitor)(unsigned char *buf,unsigned int len);/*应用层协议接收数据接口，注意，在该接口中完成数据从 uIP 到协议层拷贝，解帧、续帧功能*/
    void
(*TxdMonitor)(bool retry=false);/*发送数据，注意在这里要仔细分析数据是否需要重发*/
    unsigned short port;/*端口号*/
……
};
typedef struct protocolApp_state uip_tcp_appstate_t;
```

提示，基于 uIP 的通信系统设计需要认真关注该结构的定义，许多有关连接的应用层协议相关的自定义内容都可以在这里定义，以扩充 uIP 连接的功能。

根据上述声明定义需要 uIP 协议栈的有关应用层协议的全局变量，存储各协议的上述信息，供抽象层使用：

```
protocolApp_state g_protocolState[]
{
    {/*A 应用层协议*/

    },
    {/*B 应用层协议*/

    },
    {/*C 应用层协议*/

    }
    ……
}
#define
MAXPROTOCOLNUM
(sizeof(g_protocolState)/sizeof(g_protocolState[0]))
```

各连接对应应用层协议收发接口的配置实现方法：uIP 接收数据后，检测到应用层数据时，会调用抽象层提供的上层接口宏 UIP_APPCALL，同时，uIP 自身缓存是全局性，可以被抽象层分析。故，抽象层可以分析uIP接收到的数据TCP/IP头得知该连接的端口号等网络信息，进而可以根据应用层协议配置 g_protocolState，定位该连接的应用层协议收发数据接口。抽象层也可以根据 uIP 已经定位好的当前连接 uip_conn 信息定位应用层协议。

(2) 抽象层提供供系统调度机制使用的统一通信处理接口，例如：

void _CommunicateProcess(void)，可以仿 uIP 源代码提供的调度机制（在文件 main.c 中）实现。在该接口中调用 uIP 的相关操作，访问连接，驱动应用层协议，进而处理收发数据。

### 3.2 对 uIP 接口

供 uIP 调用的上层接口，已经由 uIP 规定了，即 UIP_APPCALL 宏，注意，该接口仅处理当前连接。示例如下：

```
#define UIP_APPCALL  _abstract_appcall_
```

关于_ abstract _appcall_( )接口的定义，需要与系统关于协议的设计紧密结合，结合上述例子，该接口定义示例如下：

```
void _abstract_appcall(void)
{
 if(uip_conn)
 {
      for(c=0;c<(MAXPROTOCOLNUM);c++)
      {
          /*此处可以分析 uIP 缓存 TCP/IP 头部内容，或者结合已经定位的连接 uip_conn 信息和应用层协议信息定位该连接的应用层协议收发接口。*/

      }
      if(uip_flags&UIP_ACKDATA)/*协议层可以发送新数据*/
      {
          uip_conn->appstate.TxdMonit
       or();
      }
      else
      if(uip_flags&UIP_NEWDATA)/*协议层接收数据，并可以发送新数据*/
      {
          uip_conn->appstate.RxdMonit
       or(uip_appdata,uip_len);
          if(uip_slen == 0)
              uip_conn->appstate.TxdMonit
      or();
      }
      else
      if(uip_flags&UIP_REXMIT)/*要求协议层重发*/
      {
          uip_conn->appstate.TxdMonit
       or(true);
      }
      else
      if(uip_flags&UIP_POLL)/*协议层可以发送新数据*/
      {
```

```
            ......
            }
            else
             if(uip_flags&UIP_CLOSE)/*连接关闭*/
            {
            ......
            }
            else
             if(uip_flags&UIP_ABORT)/*连接被对端中断，协议层是否需要保存数据重发？协议层需要明确哪些数据需要重发，哪些不需要*/
            {
            ......
            }
            else
             if(uip_flags&UIP_CONNECTED)/*连接首次建立成功，协议层可以发送数据*/
            {
            ......
            }
            else
             if(uip_flags&UIP_TIMEDOUT)/*连接超时关闭*/
            {
            ......
            }
      }
     }
```

## 4 数据确认问题

### 4.1 TCP/IP 协议对数据的延迟确认问题

根据 TCP/IP 资料，RFC 声明的最大确认延时为 500 ms，目前绝大多数实现采用 200 ms 时延。由于可以长达 500 ms 无确认，应用层协议无从得知对端是否正确接收到数据而不能确定是否发送新数据，会导致严重的通信性能问题。这是基于 TCP/IP 协议栈设计应用层协议会普遍遇到的问题。该问题对于 Internet 等商业应用并非不可接受的，但是对于广泛应用于自动化领域、对实时性要求较高的嵌入式系统却是致命的。关于该问题，uIP 提供了一种快速确认的方式，在技术手册 6.13 uIP TCP throughput booster hack 有介绍，需要几个模块（module）的支持。在这里，本人从应用层协议的设计方面探讨一下快速确认的实现方法：

(1) 要求应用层协议是问答式，这样，在应用层协议的响应帧里会携带 TCP/IP 层确认。

(2) 应用层协议不是严格问答式时，则要求协议设计时对数据进行严格分类，确定一级数据、二级数据等，并将各类数据设定严格的优先级，低优先级数据或者二级数据延时发送，高优先级数据或者一级数据要求确认机制。例如，在电力系统通信中，变位遥信、保护动作信息、保护告警信息、SOE 等可以划分为一级数据，变化遥测、电度量、文件传输等可以划分为二级数据。对于同属一级数据的变位遥信、保护动作信息等也需要设定好传输优先级。

(3) 如果应用层协议不要求对方必须确认，或者允许延时确认。这种情况下暂时无较好的处理机制，只能极大程度上依赖 TCP/IP 协议栈的确认机制。这种情况下，个人建议：

方法一：uIP 栈下连续两帧可能需要重发的紧急数据间隔 500 ms 以上，以使应用层协议或者抽象层确定是否需要重发；

方法二：应用层协议/抽象层提供紧急数据缓存队列，以等待对方确认。该队列满时，应用层协议不再发送需要确认的数据。

本人建议方法二更为有效，需要仔细实现。

### 4.2 数据重发问题

根据对 uIP 的学习研究得知，标准 uIP 并不支持数据的重发。uIP 将数据重发工作交由上层处理，在本设计方案中：uIP 判断需要数据重发，将重发标志置位；抽象层检测该重发标志，驱动应用层协议重发数据；应用层协议根据数据级别确定是否需要重发。

## 5 结语

本文根据目前嵌入式应用的特点，考虑研发进度、成本需要、资源限制等各方面的要求，在开源代码 uIP 协议栈基础上，为需要 TCP/IP 协议栈的嵌入式应用在软件方面提供了一种支持多应用层协议的设计。该设计方案已经应用于公司新一代无操作系统的电力系统二次侧保护设备的通信系统平台中，对完成该保护系统对外基于 TCP/IP 通信要求提供了可靠的支持。虽然本方案已经在电力系统二次侧设备的通信系统中实现，但是，不局限于电力系统应用，可广泛应用于各类需要基于 TCP/IP 协议栈的通信系统的嵌入式应用中。

## 参考文献

[1] Adam Dunkels. Full TCP/IP for 8-Bit Architectures[Z].

[2] Richard Stevens W. TCP/IP Illustrated Volume 1: Protocols[Z].

收稿日期：2009-06-02； 修回日期：2009-08-10

作者简介：

周绍文（1978-），男，工程师，本科，目前主要从事电力系统通信、嵌入式软件平台构建等工作。E-mail:neuzsw@sohu.com