**LwIP之UDP协议实现**

2017年10月13日 14:22:30 [有时需要偏执狂](https://me.csdn.net/u010710458) 阅读数：1654 标签： [LwIP](http://so.csdn.net/so/search/s.do?q=LwIP&t=blog)[UDP](http://so.csdn.net/so/search/s.do?q=UDP&t=blog) 更多

个人分类： [LwIP源代码分析](https://blog.csdn.net/u010710458/article/category/7200171)

版权声明：本文为博主原创文章，未经博主允许随意转载。 https://blog.csdn.net/u010710458/article/details/78225625

[UDP理论](http://blog.csdn.net/u010710458/article/details/77652679)

**UDP控制块**

每一个UDP连接都对应一个UDP控制块，UDP协议的实现就是对这些控制块结构成员进行操作。为什么需要控制块链表？为了让协议栈可以实现多个连接，可以多个网络进程同时进行。最后这些控制块通过链表连接在一起。其中链接属性为外部的udp\_pcbs是一个全局变量，指向控制块变量首地址。即这是一个指针变量，其数值是链表首地址。

extern struct udp\_pcb \*udp\_pcbs;*//链表首地址，声明为全局变量*

*//标识控制块的状态信息*

#define UDP\_FLAGS\_NOCHKSUM 0x01U

*//不进行校验和计算*

#define UDP\_FLAGS\_UDPLITE 0x02U

#define UDP\_FLAGS\_CONNECTED 0x04U

*//已经连接（UDP不需要握手协议所以连接成功仅仅是内部已经完整记录了双方IP地址和端口）*

#define UDP\_FLAGS\_MULTICAST\_LOOP 0x08U

*/\*\* Function prototype for udp pcb receive callback functions*

*\* addr and port are in same byte order as in the pcb*

*\* The callback is responsible for freeing the pbuf*

*\* if it's not used any more.*

*\**

*\* ATTENTION: Be aware that 'addr' points into the pbuf 'p' so freeing this pbuf*

*\* makes 'addr' invalid, too.*

*\**

*\* @param arg user supplied argument (udp\_pcb.recv\_arg)*

*\* @param pcb the udp\_pcb which received data*

*\* @param p the packet buffer that was received*

*\* @param addr the remote IP address from which the packet was received*

*\* @param port the remote port from which the packet was received*

*\*/*

typedef void (\*udp\_recv\_fn)(void \*arg, struct udp\_pcb \*pcb, struct pbuf \*p,

ip\_addr\_t \*addr, u16\_t port);*//定义回调函数类型，typedef就这样使用，不然定义一个函数指针会很复杂。*

*// Processing Control Block 编程就是操作控制块*

struct udp\_pcb {

*/\* Common members of all PCB types \*/*

IP\_PCB; *//宏定义IP控制块字段，使用到里面的本地IP地址和远端IP地址*

*/\* Protocol specific PCB members \*/*

struct udp\_pcb \*next;*//将控制块构成链表*

u8\_t flags;*//状态信息*

*/\*\* ports are in host byte order \*/*

u16\_t local\_port, remote\_port;

*/\**

*本地和远端端口，UDP收到报文，会遍历链表,检查本地端口号和报文中目的的端口，匹配成 功，那么调用对应控制块里面的函数指针处理报文。这样完成报文最终递交给用户程序。*

*\*/*

#if LWIP\_IGMP

*/\*\* outgoing network interface for multicast packets \*/*

ip\_addr\_t multicast\_ip;

#endif /\* LWIP\_IGMP \*/

#if LWIP\_UDPLITE

*/\*\* used for UDP\_LITE only \*/*

u16\_t chksum\_len\_rx, chksum\_len\_tx;

#endif /\* LWIP\_UDPLITE \*/

*/\*\* receive callback function \*/*

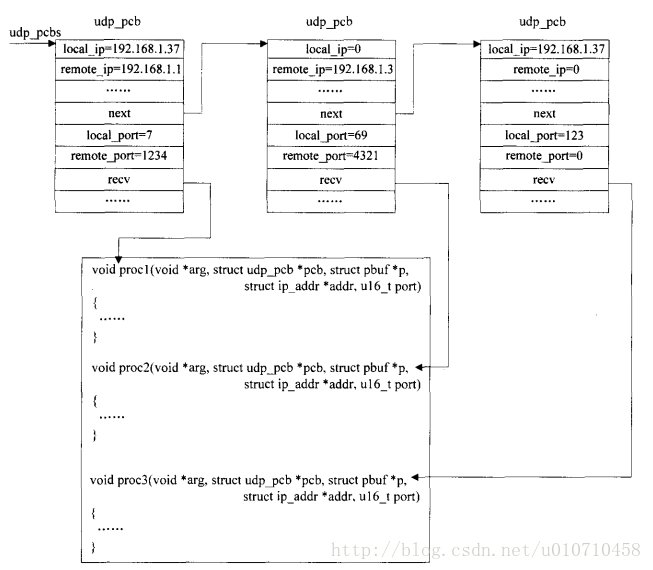
udp\_recv\_fn recv;*//回调函数，这个变量类型通过typedef定义了。匹配成功进行报文处理函数，最终报文向应用程序提交了数据*

*/\*\* user-supplied argument for the recv callback \*/*

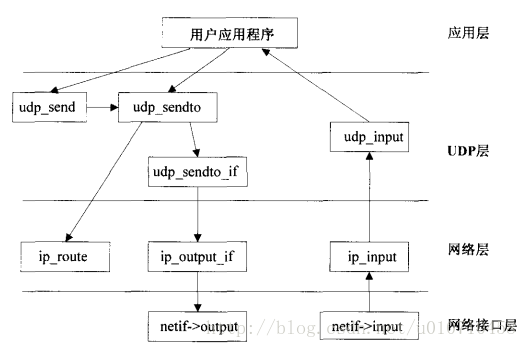
void \*recv\_arg; *//回调函数中，用户可以用过第一个参数传递一些信息，最后保存在结构体这里，这个用户参数有什么作用？*

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59

   
当有多个udp用户进程时候就会产生多个udp控制块。用户进程发送数据包，根据目标IP和端口通过IP层发送出去；用户进程接收数据包，通过数据包里面的目标IP和端口选择udp控制块里面对应的回调函数。   
总体简单操作就是遍历控制块链表。   
其中用户通过回调函数的方式被协议栈调用，这个就叫做raw/callback API。这个大型开源项目里面使用很多，框架将底层接口留给用户实现，上层牛逼程序员将框架给你搭建起来。用得最多的就是结构体里面放入函数指针。

**UDP控制块操作函数**

UDP协议操作函数树状图。   


*/\**

*从内存池申请一个控制块存储区域，返回动态分配内存首地址。*

*在建立链接之前必须动态分配控制块存储*

*\*/*

struct udp\_pcb \*

udp\_new(void)*//内存池申请一个控制块，存储接下来这个连接需要存储的信息。*

{

struct udp\_pcb \*pcb;

pcb = (struct udp\_pcb \*)memp\_malloc(MEMP\_UDP\_PCB);*//初始化指针变量*

*/\* could allocate UDP PCB? \*/*

if (pcb != NULL) {

*/\* UDP Lite: by initializing to all zeroes, chksum\_len is set to 0*

*\* which means checksum is generated over the whole datagram per default*

*\* (recommended as default by RFC 3828). \*/*

*/\* initialize PCB to all zeroes \*/*

memset(pcb, 0, sizeof(struct udp\_pcb));*//将结构体全部清0*

pcb->ttl = UDP\_TTL;*//设置udp数据报有效时间*

}

return pcb;

}

*/\*\**

*\* 绑定本地IP和端口号。也就是初始化控制块结构里面相应的成员，参数说明已经很清楚了。*

*\* ipaddr=IP\_ADDR\_ANY 指定任意网卡本地IP*

*\* port=0表示自动分配一个端口号*

*\**

*\* @param pcb： UDP PCB to be bound with a local address ipaddr and port.*

*\* @param ipaddr： local IP address to bind with. Use IP\_ADDR\_ANY to*

*\* bind to all local interfaces.*

*\* @param port： local UDP port to bind with. Use 0 to automatically bind*

*\* to a random port between UDP\_LOCAL\_PORT\_RANGE\_START and*

*\* UDP\_LOCAL\_PORT\_RANGE\_END.*

*\**

*\* ipaddr & port are expected to be in the same byte order as in the pcb.*

*\**

*\* @return： lwIP error code.*

*\* - ERR\_OK. Successful. No error occured.*

*\* - ERR\_USE. The specified ipaddr and port are already bound to by*

*\* another UDP PCB.*

*\**

*\*/*

err\_t

udp\_bind(struct udp\_pcb \*pcb, ip\_addr\_t \*ipaddr, u16\_t port)*//给UDP控制块绑定本地IP和端口，可以给定，也可以自动分配*

{

u8\_t rebind; *//表明控制块是否已经在链表中*

struct udp\_pcb \*ipcb;*//缓存变量，缓存UDP控制块首地址*

LWIP\_DEBUGF(UDP\_DEBUG | LWIP\_DBG\_TRACE, ("udp\_bind(ipaddr = "));

ip\_addr\_debug\_print(UDP\_DEBUG, ipaddr);

LWIP\_DEBUGF(UDP\_DEBUG | LWIP\_DBG\_TRACE, (", port = %"U16\_F")\n", port));

rebind = 0;

*/\* Check for double bind and rebind of the same pcb \*/*

for (ipcb = udp\_pcbs; ipcb != NULL; ipcb = ipcb->next) {

*/\* is this UDP PCB already on active list? \*/*

if (pcb == ipcb) {

*/\* pcb may occur at most once in active list \*/*

LWIP\_ASSERT("rebind == 0", rebind == 0);

*/\**

*断定rebind == 0是否成立，成立则返回，不成立，则打印调试信息，这种做法写库函数时候有用。*

*\*/*

*/\* pcb already in list, just rebind \*/*

rebind = 1;*//表示pcb已经绑定过。这里只需要重新设置IP和端口即可。*

}

*/\* By default, we don't allow to bind to a port that any other udp*

*PCB is alread bound to, unless \*all\* PCBs with that port have tha*

*REUSEADDR flag set. \*/*

#if SO\_REUSE

else if (!ip\_get\_option(pcb, SOF\_REUSEADDR) &&

!ip\_get\_option(ipcb, SOF\_REUSEADDR)) {

#else /\* SO\_REUSE \*/

*/\* port matches that of PCB in list and REUSEADDR not set -> reject \*/*

else {

#endif /\* SO\_REUSE \*/

if ((ipcb->local\_port == port) &&

*/\* IP address matches, or one is IP\_ADDR\_ANY? \*/*

(ip\_addr\_isany(&(ipcb->local\_ip)) ||

ip\_addr\_isany(ipaddr) ||

ip\_addr\_cmp(&(ipcb->local\_ip), ipaddr))) {

*/\* other PCB already binds to this local IP and port \*/*

LWIP\_DEBUGF(UDP\_DEBUG,

("udp\_bind: local port %"U16\_F" already bound by another pcb\n", port));

return ERR\_USE;

}

}

}

ip\_addr\_set(&pcb->local\_ip, ipaddr);*//设置控制块IP*

*/\* no port specified? \*/*

if (port == 0) {

port = udp\_new\_port();*//0，自动分配一个临时port*

if (port == 0) {

*/\* no more ports available in local range \*/*

LWIP\_DEBUGF(UDP\_DEBUG, ("udp\_bind: out of free UDP ports\n"));

return ERR\_USE;

}

}

pcb->local\_port = port;*//绑定port*

snmp\_insert\_udpidx\_tree(pcb);

*/\* pcb not active yet? \*/*

if (rebind == 0) {*//控制块不在链表中，将其加入链表首部*

*/\* place the PCB on the active list if not already there \*/*

pcb->next = udp\_pcbs;

udp\_pcbs = pcb;*//链表首地址*

}

LWIP\_DEBUGF(UDP\_DEBUG | LWIP\_DBG\_TRACE | LWIP\_DBG\_STATE,

("udp\_bind: bound to %"U16\_F".%"U16\_F".%"U16\_F".%"U16\_F", port %"U16\_F"\n",

ip4\_addr1\_16(&pcb->local\_ip), ip4\_addr2\_16(&pcb->local\_ip),

ip4\_addr3\_16(&pcb->local\_ip), ip4\_addr4\_16(&pcb->local\_ip),

pcb->local\_port));

return ERR\_OK;

}

*/\*\**

*\* 为UDP绑定一个远端IP和端口号，两个都绑定，UDP才处于链接状态。*

*\**

*\* This will associate the UDP PCB with the remote address.*

*\**

*\* @param pcb： UDP PCB to be connected with remote address ipaddr and port.*

*\* @param ipaddr： remote IP address to connect with.*

*\* @param port： remote UDP port to connect with.*

*\**

*\* @return ：lwIP error code*

*\**

*\* ipaddr & port are expected to be in the same byte order as in the pcb.*

*\**

*\* The udp pcb is bound to a random local port if not already bound.*

*\**

*\* @see udp\_disconnect()*

*\*/*

err\_t*//为UDP绑定一个远端IP和端口号，两个都绑定，UDP才处于链接状态，数据才可以发送出去*

*//要操作的控制块，远端IP和端口号*

udp\_connect(struct udp\_pcb \*pcb, ip\_addr\_t \*ipaddr, u16\_t port)

{

struct udp\_pcb \*ipcb;

if (pcb->local\_port == 0) {*//如何本地IP没绑定，则调用绑定本地IP*

err\_t err = udp\_bind(pcb, &pcb->local\_ip, pcb->local\_port);

if (err != ERR\_OK) {

return err;

}

}

ip\_addr\_set(&pcb->remote\_ip, ipaddr);*//设置远端IP*

pcb->remote\_port = port;*//设置远端端口好*

pcb->flags |= UDP\_FLAGS\_CONNECTED;*//标记连接成功*

*/\*\* TODO: this functionality belongs in upper layers \*/*

#ifdef LWIP\_UDP\_TODO

*/\* Nail down local IP for netconn\_addr()/getsockname() \*/*

if (ip\_addr\_isany(&pcb->local\_ip) && !ip\_addr\_isany(&pcb->remote\_ip)) {

struct netif \*netif;

if ((netif = ip\_route(&(pcb->remote\_ip))) == NULL) {

LWIP\_DEBUGF(UDP\_DEBUG, ("udp\_connect: No route to 0x%lx\n", pcb->remote\_ip.addr));

UDP\_STATS\_INC(udp.rterr);

return ERR\_RTE;

}

*/\*\* TODO: this will bind the udp pcb locally, to the interface which*

*is used to route output packets to the remote address. However, we*

*might want to accept incoming packets on any interface! \*/*

pcb->local\_ip = netif->ip\_addr;

} else if (ip\_addr\_isany(&pcb->remote\_ip)) {

pcb->local\_ip.addr = 0;

}

#endif

LWIP\_DEBUGF(UDP\_DEBUG | LWIP\_DBG\_TRACE | LWIP\_DBG\_STATE,

("udp\_connect: connected to %"U16\_F".%"U16\_F".%"U16\_F".%"U16\_F",port %"U16\_F"\n",

ip4\_addr1\_16(&pcb->local\_ip), ip4\_addr2\_16(&pcb->local\_ip),

ip4\_addr3\_16(&pcb->local\_ip), ip4\_addr4\_16(&pcb->local\_ip),

pcb->local\_port));

*/\* Insert UDP PCB into the list of active UDP PCBs. \*/*

for (ipcb = udp\_pcbs; ipcb != NULL; ipcb = ipcb->next) {*//遍历链表，查找是否是控制块*

if (pcb == ipcb) {

*/\* already on the list, just return \*/*

return ERR\_OK;*//返回OK*

}

}

*/\* PCB not yet on the list, add PCB now \*/*

pcb->next = udp\_pcbs;*//不是再加入链表，所以单独申请内存块，然后直接绑定，可以自动填充后加入链表*

udp\_pcbs = pcb;

return ERR\_OK;*//返回正确*

}

*/\*\**

*\* Disconnect a UDP PCB也就是清除远端IP和端口。要使用这个，那么就需要重新connect一次绑定一个远端IP和地址。*

*\**

*\* @param pcb the udp pcb to disconnect.*

*\*/*

void*//清除远端IP和端口，并设置为非连接*

udp\_disconnect(struct udp\_pcb \*pcb)

{

*/\* reset remote address association \*/*

ip\_addr\_set\_any(&pcb->remote\_ip);*//清IP*

pcb->remote\_port = 0;*//清端口*

*/\* mark PCB as unconnected \*/*

pcb->flags &= ~UDP\_FLAGS\_CONNECTED;*//设置为非连接状态*

}

void*//从链表中删除一个UDP控制块，也就是清楚对应的内存池空间*

udp\_remove(struct udp\_pcb \*pcb)

{

struct udp\_pcb \*pcb2;

snmp\_delete\_udpidx\_tree(pcb);

*/\* pcb to be removed is first in list? \*/*

if (udp\_pcbs == pcb) {

*/\* make list start at 2nd pcb \*/*

udp\_pcbs = udp\_pcbs->next;

*/\* pcb not 1st in list \*/*

} else {

for (pcb2 = udp\_pcbs; pcb2 != NULL; pcb2 = pcb2->next) {

*/\* find pcb in udp\_pcbs list \*/*

if (pcb2->next != NULL && pcb2->next == pcb) {

*/\* remove pcb from list \*/*

pcb2->next = pcb->next;

}

}

}

memp\_free(MEMP\_UDP\_PCB, pcb);

}

*/\*\**

*\* Set a receive callback for a UDP PCB，设置UDP接收数据回调函数。*

*\**

*\* This callback will be called when receiving a datagram for the pcb.*

*\**

*\* @param pcb the pcb for wich to set the recv callback*

*\* @param recv function pointer of the callback function*

*\* @param recv\_arg additional argument to pass to the callback function*

*\*/*

void*//注册回调函数，最重要的地方*

udp\_recv(struct udp\_pcb \*pcb, udp\_recv\_fn recv, void \*recv\_arg)

{

*/\* remember recv() callback and user data \*/*

pcb->recv = recv;*//用户自定义数据报处理函数*

pcb->recv\_arg = recv\_arg;*//用户自定义数据*

}

err\_t*//发送报文，通过控制块，发送对应的报文*

udp\_send(struct udp\_pcb \*pcb, struct pbuf \*p)

{

*/\* send to the packet using remote ip and port stored in the pcb \*/*

return udp\_sendto(pcb, p, &pcb->remote\_ip, pcb->remote\_port);*//用户数据填充pbuf，然后udp取出数据区域填充，然后送给ip层*

}

void

udp\_input(struct pbuf \*p, struct netif \*inp);

*//udp报文接收函数，这函数是被IP层调用，用于处理报文并根据端口号交给对应的用户进程（这里指的对应的回调函数）。*

*/\**

*与报文中目的端口号和源端口号都匹配且处于连接状态的控制块将被视作最佳匹配的控制块，若没有这样的控制块，则与目的端口号相匹配的未连接控制块将被视作查找结构。使用UDP传输、处理数据的关键在于定义用户自定义的数据处理函数，当UDP匹配到某个控制块时，将回调用户注册的处理函数，用户程序应该负责报文pbuf的删除工作。如果找不到匹配的端口号，而该数据报确实是发送给本地的，则一个端口不可达报文会被返回给源主机(这点是为了告诉发送主机，这点完全可以利用wireshark捕捉到)。*

*\*/*

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 153
* 154
* 155
* 156
* 157
* 158
* 159
* 160
* 161
* 162
* 163
* 164
* 165
* 166
* 167
* 168
* 169
* 170
* 171
* 172
* 173
* 174
* 175
* 176
* 177
* 178
* 179
* 180
* 181
* 182
* 183
* 184
* 185
* 186
* 187
* 188
* 189
* 190
* 191
* 192
* 193
* 194
* 195
* 196
* 197
* 198
* 199
* 200
* 201
* 202
* 203
* 204
* 205
* 206
* 207
* 208
* 209
* 210
* 211
* 212
* 213
* 214
* 215
* 216
* 217
* 218
* 219
* 220
* 221
* 222
* 223
* 224
* 225
* 226
* 227
* 228
* 229
* 230
* 231
* 232
* 233
* 234
* 235
* 236
* 237
* 238
* 239
* 240
* 241
* 242
* 243
* 244
* 245
* 246
* 247
* 248
* 249
* 250
* 251
* 252
* 253
* 254
* 255
* 256
* 257
* 258
* 259
* 260
* 261
* 262
* 263

**UDP简单操作例程**

协议栈已经实现了，其实自己写好了网络驱动，以及调用上述函数，非常简单。

void udp\_demo\_init(void)

{

err\_t err;

struct udp\_pcb \*udppcb;*//*

struct ip\_addr rmtipaddr;

udppcb = udp\_new();*//初始化指针变量并指向新分配的控制块内存*

if(udppcb){*//*

IP4\_ADDR(&rmtipaddr,10,13,3,214);*//将点分10进制IP转为4字节变量，因为IP控制块里面通过32位存储IP地址。*

err=udp\_connect(udppcb,&rmtipaddr,UDP\_REMOTE\_PORT);*//设置远端IP和端口*

err=udp\_bind(udppcb,IP\_ADDR\_ANY,UDP\_LOCAL\_PORT);*//设置本地IP和端口，其中IP\_ADDR\_ANY表明使用DHCP网卡地址，在发送函数里会判断这个变量，选择网卡地址*

udp\_recv(udppcb , udp\_demo\_recv , NULL);*//注册控制块回调函数，端口收到数据，回调处理函数将被调用。*

}

}

*//这里进行简单处理，也就是将接受到的数据发送出去。*

void udp\_demo\_recv(void \*arg,struct udp\_pcb \*upcb,struct pbuf \*p,struct ip\_addr \*addr,u16\_t port)

{

struct ip\_addr my\_ipaddr;

unsigned char \*temp = (unsigned char \*)addr;*//保存远端IP。*

IP4\_ADDR(&my\_ipaddr , temp[0] , temp[1] , temp[2] , temp[3]);*//*

udp\_send(upcb , p );*//通过控制块发送数据，已经绑定了远端IP。*

pbuf\_free(p);*//发送出去，用户空间清除接收数据帧pbuf。*

}

void udp\_demo\_senddata(struct udp\_pcb \*upcb)

{

struct pbuf \*ptr;

ptr=pbuf\_alloc(PBUF\_TRANSPORT,strlen((char\*)tcp\_demo\_sendbuf),PBUF\_POOL); *//*

if(ptr)

{

ptr->payload=(void\*)tcp\_demo\_sendbuf;

udp\_send(upcb,ptr); *//*

pbuf\_free(ptr);*//*

}

}

*//*

void udp\_demo\_connection\_close(struct udp\_pcb \*upcb)

{

udp\_disconnect(upcb);

udp\_remove(upcb); *//清理内存块空间*

}

*//上述在STM32上面实现，进行回环测试。通过pc网口调试助手发送数据，STM32相应对应数据。*

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40

至此UDP协议讲解完毕。UDP协议是一个比较简单的传输层协议，不涉及复杂的握手已经确认机制，所以实现起来也比较简单。