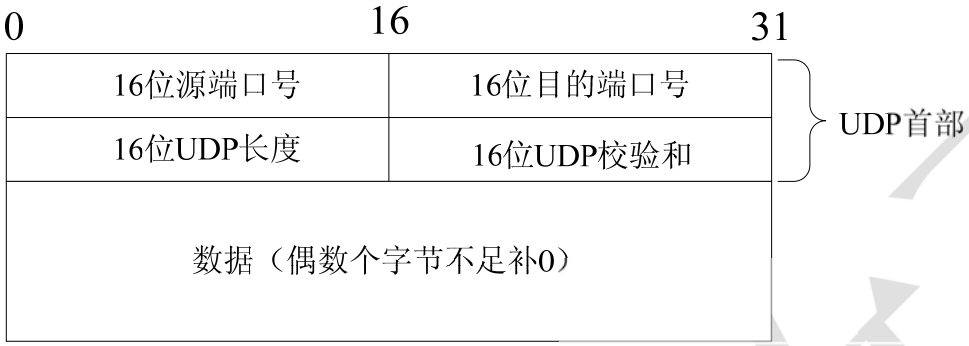
网络编程—附录

附录 A（报头介绍）

UDP 报头



源端口号：发送方端口号
目的端口号：接收方端口号
长度：UDP用户数据报的长度，最小值是8（仅有首部）
校验和：检测UDP用户数据报在传输中是否有错，有错就丢弃

TCP 报头

1. 源端口号：发送方端口号
2. 目的端口号：接收方端口号
3. 序列号：本报文段的数据的第一个字节的序号
4. 确认序号：期望收到对方下一个报文段的第一个数据字节的序号

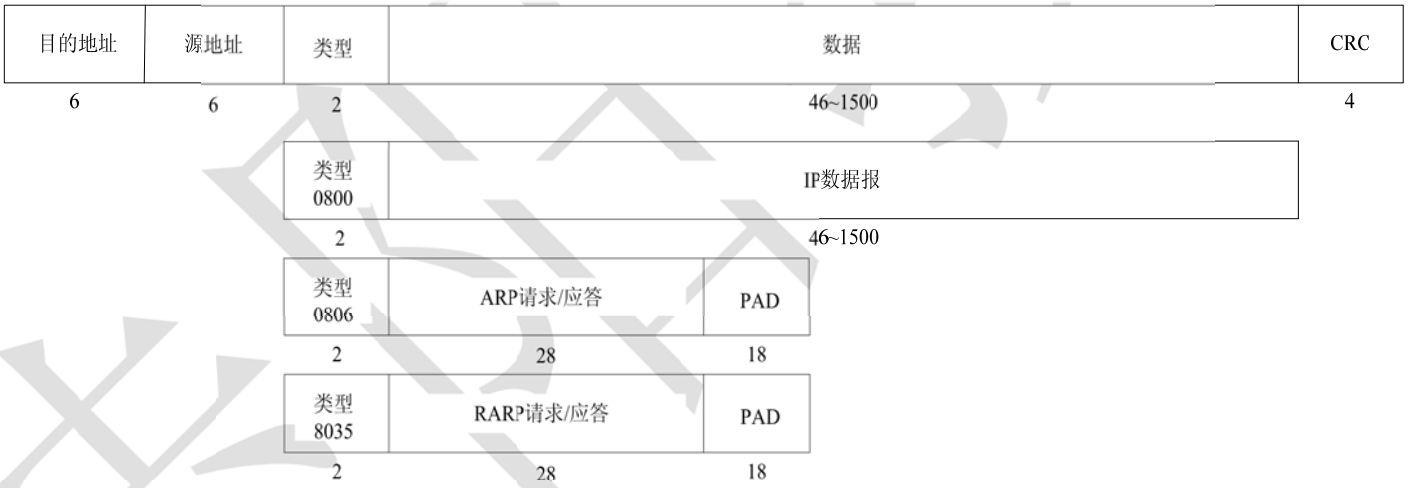
5.首部长度（数据偏移）：TCP报文段的数据起始处距离TCP报文段的起始处有多远，即首部长度。单位：32位，即以4字节为计算单位。
6.保留：占6位，保留为今后使用，目前应置为0
7.紧急URG：此位置1，表明紧急指针字段有效，它告诉系统此报文段中有紧急数据，应尽快传送
8.确认ACK：仅当ACK=1时确认号字段才有效，TCP规定，在连接建立后所有传达的报文段都必须把ACK置1
9.推送PSH：当两个应用进程进行交互式的通信时，有时在一端的应用进程希望在键入一个命令后立即就能够收到对方的响应。在这种情况下，TCP就可以使用推送（push）操作，这时，发送方TCP把PSH置1，并立即创建一个报文段发送出去，接收方收到PSH=1的报文段，就尽快地（即“推送”向前）交付给接收应用进程，而不再等到整个缓存都填满后再向上交付
10.复位RST：用于复位相应的TCP连接
11.同步SYN：仅在三次握手建立TCP连接时有效。当SYN=1而ACK=0时，表明这是一个连接请求报文段，对方若同意建立连接，则应在相应的报文段中使用SYN=1和ACK=1.因此，SYN置1就表示这是一个连接请求或连接接受报文
12.终止FIN：用来释放一个连接。当FIN=1时，表明此报文段的发送方的数据已经发送完毕，并要求释放运输连接。
13.窗口：指发送本报文段的一方的接收窗口（而不是自己的发送窗口）
14.校验和：校验和字段检验的范围包括首部和数据两部分，在计算校验和时需要加上12字节的伪头部
15.紧急指针：仅在URG=1时才有意义，它指出本报文段中的紧急数据的字节数（紧急数据结束后就是普通数据），即指出了紧急数据的末尾在报文中的位置，注意：即使窗口为零时也可发送紧急数据
16.选项：长度可变，最长可达40字节，当没有使用选项时，TCP首部长度是20字节

IP 报头

1.版本：IP协议的版本。通信双方使用过的IP协议的版本必须一致，目前最广泛使用的IP协议版本号为4（即IPv4 ）
2.首部长度：单位是32位（4字节）

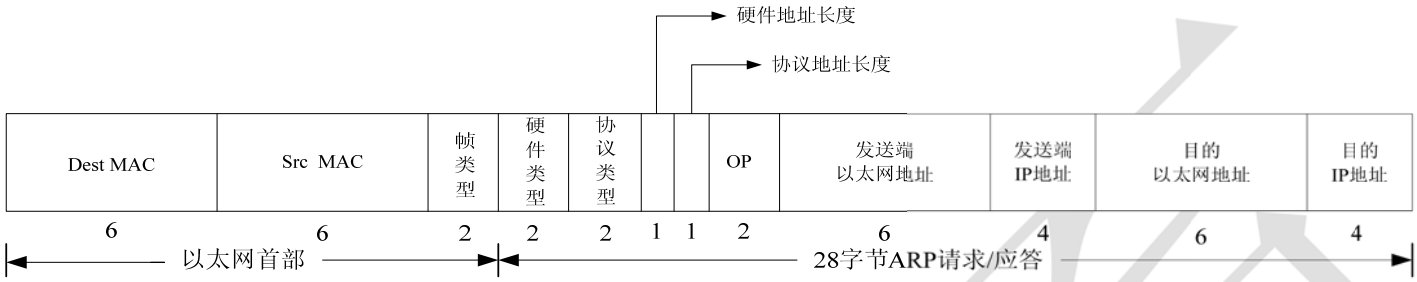
3.服务类型：一般不适用，取值为0
4.总长度：指首部加上数据的总长度，单位为字节
5.标识（identification）：IP软件在存储器中维持一个计数器，每产生一个数据报，计数器就加1，并将此值赋给标识字段
6.标志（flag）：目前只有两位有意义。
✧ 标志字段中的最低位记为MF。MF=1即表示后面“还有分片”的数据报。MF=0表示这已是若干数据报片中的最后一个。
✧ 标志字段中间的一位记为DF，意思是“不能分片”，只有当DF=0时才允许分片
7.片偏移：指出较长的分组在分片后，某片在源分组中的相对位置，也就是说，相对于用户数据段的起点，该片从何处开始。片偏移以8字节为偏移单位。
8.生存时间：TTL，表明是数据报在网络中的寿命，即为“跳数限制”，由发出数据报的源点设置这个字段。路由器在转发数据之前就把TTL值减一，当TTL值减为零时，就丢弃这个数据报。
9.协议：指出此数据报携带的数据时使用何种协议，以便使目的主机的IP层知道应将数据部分上交给哪个处理过程，常用的ICMP(1),IGMP(2),TCP(6),UDP(17),IPv6（41）
10.首部校验和：只校验数据报的首部，不包括数据部分。
11.源地址：发送方IP地址
12.目的地址：接收方IP地址

MAC 头部



1.CRC、PAD 在组包时可以忽略

ARP 头部



1. Dest MAC:目的MAC地址
2. Src MAC：源MAC地址
3. 帧类型：0x0806
4. 硬件类型：1（以太网）
5. 协议类型：0x0800（IP地址）
6. 硬件地址长度：6
7. 协议地址长度：4
8. OP：1（ARP请求），2（ARP应答），3（RARP请求），4（RARP应答）

附录 B（库函数介绍）

Libpcap

pcap_open_live( )

```
1.pcap_t *pcap_open_live(const char *device,int snaplen,int promisc,
                  int to_ms,char *ebuf)
```

功能：打开一个网络结构畸形数据包捕获
返回值：返回一个Libpcap句柄
参数:device：网络接口的名字
snaplen：捕获数据包的长度
promise：1代表混杂模式,其它非混杂模式
to_ms：等待时间
ebuf：存储错误信息

pcap_close( )

```
2.void pcap_close(pcap_t *p)
```

功能：关闭Libpcap操作，并销毁相应的资源
参数：p:需要关闭的Libpcap句柄
返回值：无

pcap_compile( )

```
3.int pcap_compile(pcap_t *p,struct bpf_program *program,
            char *buf,int optimize,bpf_u_int32 mask)
```

功能：编译BPF过滤规则
返回值：成功返回0，失败返回-1
参数：p:Libpcap句柄
program：bpf过滤规则
buf:过滤规则字符串
optimize：优化
mask：掩码

pcap_setfilter( )

4.int pcap_setfilter(pcap *p, struct bpf_program*fp)

功能：设置BPF过滤规则
返回值：成功返回0，失败返回-1
参数 p：Libpcap句柄
fp：BPF过滤规则

pcap_next( )

5.const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)

功能：捕获一个网络数据包
参数 p：Libpcap句柄;
h：数据包头
返回值：捕获的数据包的地址

pcap_loop( )

6.int pcap_loop(pcap_t *p,int cnt, pcap_handler callback,
u_char *user)

功能:循环捕获网络数据包，直到遇到错误或者满足退出条件；每次捕获一个数据包就会调用callback指示的回调函数，所以，可以在回调函数中进行数据包的处理操作
返回值：成功返回0，失败返回负数
参数 p：Libpcap句柄
cnt：指定捕获数据包的个数，如果是-1，会永无休止的捕获;
callback：回调函数
user：向回调函数中传递的参数

typedef void (*pcap_handler)(u_char *, const struct pcap_pkthdr *, const u_char *);

Libnet

libnet_init( )

1.libnet_t *libnet_init(int injection_type, char *device, char *err_buf)
功能：数据包内存初始化及环境建立
参数：injection_type: 构造的类型(LIBNET_LINK,
LIBNET_RAW4, LIBNET_LINK_ADV,LIBNET_RAW4_ADV)
device：网络接口，如"eth0",或IP地址，亦可为NULL(自动查询搜索)
err_buf: 存放出错的信息

返回值：成功：一个libnet句柄； 失败：NULL

libnet_destroy( )

2.void libnet_destroy(libnet_t *l);

功能：释放资源
参数： l: libnet_init返回的句柄
返回值：无

libnet_addr2name4( )

3.char* libnet_addr2name4(u_int32_t in, u_int8_t use_name)

功能：将网络字节序转换成点分十进制数串
参数： in: 网络字节序的ip地址
use_name: LIBNET_RESOLVE、 LIBNET_DONT_RESOLVE
返回值：成功：点分十进制ip地址； 失败：NULL

libnet_name2addr4( )

4.u_int32_t libnet_name2addr4(libnet_t *l, char *host_name, u_int8_t use_name)

功能：将点分十进制数串转换为网络字节序ip地址
参数：l：libnet句柄
host_name：点分十进制数串的地址
use_name: (LIBNET_RESOLVE, LIBNET_DONT_RESOLVE)
返回值：成功：网络字节序ip地址； 失败：-1

libnet_get_ipaddr4( )

5.u_int32_t libnet_get_ipaddr4(libnet_t *l)

功能：获取接口设备ip地址
参数 l: libnet句柄
返回值：成功：网络字节序的ip地址； 失败：-1

libnet_get_hwaddr( )

6.struct libnet_ether_addr* libnet_get_hwaddr(libnet_t *l)

功能：获取接口设备硬件地址
参数： l ：libnet句柄
返回值： 成功：指向MAC地址的指针； 失败：NULL

libnet_build_udp( )

```
7.libnet_ptag_t libnet_build_udp(  u_int16_t sp, u_int16_t dp,
                                u_int16_t len,  u_int16_t sum,u_int8_t *payload,
                                u_int32_t payload_s,libnet_t *l,libnet_ptag_t ptag)
```

功能：构造udp数据包
参数：sp：源端口号；
dp：目的端口号
len：udp包总长度；
sum：校验和，设为0，libnet自动填充
payload：负载，可设置为NULL；
payload_s：负载长度，或为0
l：libnet句柄
ptag：协议标记
返回值：
成功：协议标记
失败：-1

libnet_build_tcp( )

```
8.libnet_ptag_t libnet_build_tcp(
          u_int16_t sp,u_int16_t dp,
          u_int32_t seq,u_int32_t ack,
          u_int8_t control,u_int16_t win
          u_int16_t sum,u_int16_t urg,
          u_int16_t len,u_int8_t *payload,
          u_int32_t payload_s,libnet_t *l,
          libnet_ptag_t ptag)
```

功能：构造tcp数据包
参数：sp：源端口号
dp：目的端口号
seq：序号
ack：ack标记
control：控制标记
win：窗口大小
sum：校验和，设为0，libnet自动填充
urg：紧急指针
len：tcp包长度
payload：负载，可设置为NULL
payload_s：负载长度，或为0
l：libnet句柄
ptag：协议标记
返回值：成功：协议标记； 失败：-1

libnet_build_tcp_options( )

9.libnet_ptag_t libnet_build_tcp_options(u_int8_t *options, u_int32_t options_s,
libnet_t *l, libnet_ptag_t ptag)

功能：构造tcp选项数据包
参数：options：tcp选项字符串
options_s: 选项长度
l：libnet句柄
ptag：协议标记，若为0,建立一个新的协议
返回值：成功：协议标记； 失败：-1

libnet_build_ipv4( )

10.libnet_ptag_t libnet_build_ipv4(
u_int16_t ip_len,u_int8_t tos,
u_int16_t id,u_int16_t flag,
u_int8_t ttl,u_int8_t prot,
u_int16 sum,u_int32_t src,
u_int32_t dst,u_int8_t *payload,
u_int32_t payload_s,
libnet_t *l,libnet_ptag_t ptag)

功能：构造一个 IPv4 数据包
参数：ip_len：ip 包总长
tos：服务类型
id：ip标识
flag：片偏移
ttl：生存时间
prot：上层协议
sum：校验和，设为0，libnet自动填充
src：源ip地址
dst：目的ip地址
payload：负载，可设置为NULL
payload_s：负载长度，或为0
l：libnet句柄
ptag：协议标记
返回值：成功：协议标记； 失败：-1

libnet_build_ipv4_options( )

```
11.libnet_ptag_t libnet_build_ipv4_options(
          u_int8_t *options,u_int32_t options,
          libnet_t *l,libnet_ptag_t ptag)
```

功能：构造IPv4选项数据包
参数：options：tcp选项字符串
options_s: 选项长度
l：libnet句柄
ptag：协议标记，若为0,建立一个新的协议
返回值：成功：协议标记； 失败：-1

libnet_build_arp( )

```
12.libnet_ptag_t libnet_build_arp(
                    u_int16_t hrd, u_int16_t pro,
                    u_int8_t hln, u_int8_t pln,
                   u_int16_t op, u_int8_t *sha,
                   u_int8_t *spa, u_int8_t *tha,
                   u_int8_t *tpa, u_int8_t *payload,
                   u_int32_t payload_s, libnet_t *l,
                   libnet_ptag_t ptag)
```

功能：构造 arp 数据包
参数：hrd：硬件地址格式，ARPHRD_ETHER（以太网）
pro：协议地址格式，ETHERTYPE_IP（IP协议）
hln：硬件地址长度
pln：协议地址长度
op：ARP协议操作类型（1：ARP请求，2：ARP回应，3：RARP请求，4：RARP回应）
sha：发送者硬件地址
spa：发送者协议地址
tha：目标硬件地址
tpa：目标协议地址
payload：负载，可设置为NULL
payload_s：负载长度，或为0
l: libnet句柄
ptag：协议标记
返回值：成功：协议标记； 失败：-1

libnet_build_ethernet( )

```
13.libnet_ptag_t libnet_build_ethernet(
          u_int8_t *dst,u_int8_t *src,
          u_int16_t type,
          u_int8_t *payload,
          u_int32_t payload_s,
          libnet_t *l,libnet_ptag_t ptag)
```

功能：构造一个以太网数据包
参数： dst：目的 mac
src：源mac
type：上层协议类型
payload：负载，即附带的数据
payload_s：负载长度
l：libnet句柄
ptag：协议标记
返回值：成功：协议标记； 失败：-1

附录 C(结构体)

以太网头部

struct ether_header 所在位置:#include <net/ethernet.h> （首选）

```
struct ether_header
{
  u_int8_t  ether_dhost[ETH_ALEN];  /* 目的MAC地址   */
  u_int8_t  ether_shost[ETH_ALEN];  /* 源MAC地址     */
  u_int16_t ether_type;             /* 帧类型        */
};
```

struct ethhdr ; //所在位置:#include <linux/if_ether.h>

```
struct ethhdr
{
  unsigned char h_dest[ETH_ALEN];   /* 目的MAC地址   */
  unsigned char h_source[ETH_ALEN]; /* 源MAC地址     */
  unsigned short int h_proto;       /* 帧类型        */
};
```

ARP 头部

Struct arphdr ; 所在位置 /usr/include/net/if_arp.h #include <net/if_arp.h>

```
struct arphdr
{
    unsigned short int ar_hrd;          /* 硬件类型          */
    unsigned short int ar_pro;          /* 协议类型          */
    unsigned char ar_hln;               /* 硬件地址长度      */
    unsigned char ar_pln;               /* 协议地址长度      */
    unsigned short int ar_op;           /* ARP命令           */
#if 0
    /* Ethernet looks like this : This bit is variable sized
       however...  */
    unsigned char __ar_sha[ETH_ALEN];   /* 发送端以太网地址  */
    unsigned char __ar_sip[4];          /* 发送端IP地址      */
    unsigned char __ar_tha[ETH_ALEN];   /* 目的以太网地址    */
    unsigned char __ar_tip[4];          /* 目的IP地址        */
#endif
};
```

IP 头部

struct iphdr; //所在位置:/usr/include/netinet/ip.h #include <netinet/ip.h>

```
struct iphdr
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;                 /*首部长度       */
    unsigned int version:4;             /*版本           */
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;             /*版本           */
    unsigned int ihl:4;                 /*首部长度       */
#else
# error "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;                       /*服务类型       */
    u_int16_t tot_len;                  /*总长度         */
    u_int16_t id;                       /*标识           */
    u_int16_t frag_off;                 /*标志、片偏移   */
    u_int8_t ttl;                       /*生存时间       */
    u_int8_t protocol;                  /*协议           */
    u_int16_t check;                    /*首部校验和     */
    u_int32_t saddr;                    /*源地址         */
    u_int32_t daddr;                    /*源地址         */
    /*The options start here. */
 };
```

UDP 头部

struct udphdr ; //所在位置:/usr/include/netinet/udp.h #include <netinet/udp.h>

```
struct udphdr
{
  u_int16_t source;                     /*源端口号   */
  u_int16_t dest;                       /*目的端口号*/
  u_int16_t len;                        /*长度       */
  u_int16_t check;                      /*校验和     */
};
```

TCP 头部

struct tcphdr; //所在位置:/usr/include/netinet/tcp.h #include <netinet/tcp.h>

```
struct tcphdr
  {
    u_int16_t source;                   /*源端口号   */
    u_int16_t dest;                     /*目的端口号*/
    u_int32_t seq;                      /*序列号     */
    u_int32_t ack_seq;                  /*确认序号   */
#  if __BYTE_ORDER == __LITTLE_ENDIAN
    u_int16_t res1:4;/**/
    u_int16_t doff:4;                   /*保留: 4    */
    u_int16_t fin:1;                    /*终止FIN    */
    u_int16_t syn:1;                    /*同步SYN    */
    u_int16_t rst:1;                    /*复位RST    */
    u_int16_t psh:1;                    /*推送PSH    */
    u_int16_t ack:1;                    /*确认ACK    */
    u_int16_t urg:1;                    /*紧急URG    */
    u_int16_t res2:2;                   /*保留:      2*/
#  elif __BYTE_ORDER == __BIG_ENDIAN
    u_int16_t doff:4;                   /*首部长度   */
    u_int16_t res1:4;                   /*保留: 4    */
    u_int16_t res2:2;                   /*保留: 2    */
    u_int16_t urg:1;                    /*紧急URG    */
    u_int16_t ack:1;                    /*确认ACK    */
    u_int16_t psh:1;                    /*推送PSH    */
    u_int16_t rst:1;                    /*复位RST    */
    u_int16_t syn:1;                    /*同步SYN    */
    u_int16_t fin:1;                    /*终止FIN    */
#  else
#   error "Adjust your <bits/endian.h> defines"
#  endif
    u_int16_t window;                   /*窗口       */
    u_int16_t check;                    /*校验和     */
    u_int16_t urg_ptr;                  /*紧急指针   */
};
```