

Linux 下 ping 命令实现详解

相信大家一定遇到过上不了网的情形，都知道用个 ping 命令。这不小王就是这样的女孩，老是上不了网，老是找我，我就先 ping 一下，逐步找找问题在哪儿，有的放矢，不至于盲目抓瞎

好，今天就给大家讲讲有关 linux 下 ping 程序的实现，以后谁再找我修网，对不起，该不奉陪，我只为小王，嘿嘿。

先来说说 ping 程序的原理吧，其实挺简单，就是一个主机系统向另外一个主机系统说：I love you(ICMP 报文)，然后那个主机如果相信你或者说想和你通信，和你心知心，那它就把收到的 I love you(ICMP)报文原样返回。好嘛，源主机收到这个回应后，就 happy 了，因为对方是和自己心连心的。如果对方没有收到这个消息，或者对你不感冒，不愿意理你，不回你这个报文，或者说些不知云是云雾是雾的话，对不起啦，感情是两个人的事情哦。

要想深刻了解，需有入目三分的实力，这个 ping 也一样，咱们先来看看它采用的 TCP/IP 协议，我刚说了，它发送的是 ICMP 回显请求，回答的是回显应答报文。谈起这个 ICMP(Internet Control Message,网际控制报文协议)是为网关和目标主机而提供的一种差错控制机制，使它们在遇到差错时能把错误报告给报文源发方。是 IP 层的一个协议。但是由于差错报告在发送给报文源发方时可能也要经过若干子网，因此牵涉到路由选择等问题，所以 ICMP 报文需通过 IP 协议来发送。ICMP 数据报的数据发送前需要两级封装：首先添加 ICMP 报头形成 ICMP 报文，再添加 IP 报头形成 IP 数据报。如下图所示：

IP 报头
ICMP 报头
ICMP 数据报

由于 IP 层协议是一种点对点的协议，而非端对端的协议，它提供无连接的数据报服务，没有端口的概念，因此很少使用 bind()和 connect() 函数，若有使用也只是用于设置 IP 地址。发送数据使用 sendto()函数，接收数据使用 recvfrom()函数。

TCP/IP 的经典大作《TCP/IP 协议详解.卷一》清晰的告诉我，IP 报头格式如下：

0	4	8	12	16	19	24	31
版本	首部长度	服务类型	长度				
认证				标志	段偏移量		
TTL		协议		校验和			
源 IP 地址							
目的 IP 地址							
选项 ...							

详细的，小王那懒的人都知道翻翻上面提到的书，我也就不详细介绍了，我这里给出 linux 中的数据结构实现：

```
struct ip
{
    #if __BYTE_ORDER == __LITTLE_ENDIAN
```

```

    unsigned int ip_hl:4;        /* header length */
    unsigned int ip_v:4;        /* version */
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
    unsigned int ip_v:4;        /* version */
    unsigned int ip_hl:4;        /* header length */
#endif
    u_int8_t ip_tos;            /* type of service */
    u_short ip_len;             /* total length */
    u_short ip_id;              /* identification */
    u_short ip_off;             /* fragment offset field */
#define IP_RF 0x8000           /* reserved fragment flag */
#define IP_DF 0x4000           /* dont fragment flag */
#define IP_MF 0x2000           /* more fragments flag */
#define IP_OFFMASK 0x1fff      /* mask for fragmenting bits */
    u_int8_t ip_ttl;            /* time to live */
    u_int8_t ip_p;              /* protocol */
    u_short ip_sum;             /* checksum */
    struct in_addr ip_src, ip_dst; /* source and dest address */
};

```

别看这多，其实 ping 程序用到的没几个：

(1)IP 报头长度 IHL (Internet Header Length) 以 4 字节为一个单位来记录 IP 报头的长度，是上述 IP 数据结构的 ip_hl 变量。

(2)生存时间 TTL (Time To Live) 以秒为单位，指出 IP 数据报能在网络上停留的最长时间，其值由发送方设定，并在经过路由的每一个节点时减一，当该值为 0 时，数据报将被丢弃，是上述 IP 数据结构的 ip_ttl 变量。ICMP 报文分为两种：查询报文和差错报文。每个 ICMP 报头均包含类型、编码和校验和这三项内容，其余选项则随 ICMP 的功能不同而不同。ICMP 报文格式如下：

8 位类型	8 位代码	16 位校验和
(不同类型和代码有不同的内容)		

Ping 命令只使用众多 ICMP 报文中的两种："请求回送(ICMP_ECHO)和"请求回应'(ICMP_ECHOREPLY)。在 Linux 中定义如下：

```

#define ICMP_ECHO    0
#define ICMP_ECHOREPLY 8

```

在 Linux 中 ICMP 数据结构(<netinet/ip_icmp.h>)定义如下：

```

struct icmp
{
    u_int8_t icmp_type; /* type of message, see below */
    u_int8_t icmp_code; /* type sub code */
    u_int16_t icmp_cksum; /* ones complement checksum of struct */
    union

```

```

{
    u_char ih_pptr;      /* ICMP_PARAMPROB */
    struct in_addr ih_gwaddr; /* gateway address */
    struct ih_idseq      /* echo datagram */
    {
        u_int16_t icd_id;
        u_int16_t icd_seq;
    } ih_idseq;
    u_int32_t ih_void;

    /* ICMP_UNREACH_NEEDFRAG -- Path MTU Discovery (RFC1191) */
    struct ih_pmtu
    {
        u_int16_t ipm_void;
        u_int16_t ipm_nextmtu;
    } ih_pmtu;

    struct ih_rtradv
    {
        u_int8_t irt_num_addrs;
        u_int8_t irt_wpa;
        u_int16_t irt_lifetime;
    } ih_rtradv;
} icmp_hun;
#define icmp_pptr    icmp_hun.ih_pptr
#define icmp_gwaddr  icmp_hun.ih_gwaddr
#define icmp_id      icmp_hun.ih_idseq.icd_id
#define icmp_seq      icmp_hun.ih_idseq.icd_seq
#define icmp_void     icmp_hun.ih_void
#define icmp_pmvoid   icmp_hun.ih_pmtu.ipm_void
#define icmp_nextmtu   icmp_hun.ih_pmtu.ipm_nextmtu
#define icmp_num_addrs icmp_hun.ih_rtradv.irt_num_addrs
#define icmp_wpa       icmp_hun.ih_rtradv.irt_wpa
#define icmp_lifetime  icmp_hun.ih_rtradv.irt_lifetime
union
{
    struct
    {
        u_int32_t its_otime;
        u_int32_t its_rtime;
        u_int32_t its_ttime;
    } id_ts;
    struct
    {

```

```

    struct ip idi_ip;
    /* options and then 64 bits of data */
} id_ip;
struct icmp_ra_addr id_radv;
u_int32_t id_mask;
u_int8_t id_data[1];
} icmp_dun;
#define icmp_otime icmp_dun.id_ts.its_otime
#define icmp_rtime icmp_dun.id_ts.its_rtime
#define icmp_ttime icmp_dun.id_ts.its_ttime
#define icmp_ip icmp_dun.id_ip.idi_ip
#define icmp_radv icmp_dun.id_radv
#define icmp_mask icmp_dun.id_mask
#define icmp_data icmp_dun.id_data
};

```

Ping 命令中需要显示的信息，包括 icmp_seq 和 ttl 都已有实现的办法，但还缺 rtt 往返时间。为了实现这一功能，可利用 ICMP 数据报携带一个时间戳。使用以下函数生成时间戳：

```

#include
int gettimeofday(struct timeval *tp,void *tzp)
其中 timeval 结构如下：
    struct timeval{
        long tv_sec;
        long tv_usec;
    }

```

在发送和接收报文时由 gettimeofday 分别生成两个 timeval 结构，两者之差即为往返时间，即 ICMP 报文发送与接收的时间差，而 timeval 结构由 ICMP 数据报携带，

tzp 指针表示时区，一般都不使用，赋 NULL 值。系统自带的 ping 命令当它接送完所有 ICMP 报文后，会对所有发送和所有接收的 ICMP 报文进行统计，从而计算 ICMP 报文丢失的比率。为达此目的，定义两个全局变量：接收计数器和发送计数器，用于记录 ICMP 报文接受和发送数目。丢失数目=发送总数-接收总数，丢失比率=丢失数目/发送总数。现给出模拟 Ping 程序功能的代码如下：

```

☐☐必要的头文件,宏定义和函数说明#include <stdio.h>
#include <signal.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <setjmp.h>

```

```
#include <errno.h>
```

```
#define PACKET_SIZE      4096
```

```
#define MAX_WAIT_TIME    5
```

```
#define MAX_NO_PACKETS   3
```

```
char sendpacket[PACKET_SIZE];
```

```
char recvpacket[PACKET_SIZE];
```

```
int sockfd, datalen=56;
```

```
int nsend=0, nreceived=0;
```

```
struct sockaddr_in dest_addr;
```

```
pid_t pid;
```

```
struct sockaddr_in from;
```

```
struct timeval tvrecv;
```

```
void statistics(int signo);
```

```
unsigned short cal_chksum(unsigned short *addr, int len);
```

```
int pack(int pack_no);
```

```
void send_packet(void);
```

```
void recv_packet(void);
```

```
int unpack(char *buf, int len);
```

```
void tv_sub(struct timeval *out, struct timeval *in);
```

```
void statistics(int signo)
```

```
{    printf("\n-----PING statistics-----\n");
```

```
    printf("%d packets transmitted, %d received , %d%%  lost\n",  
           nsend, nreceived, (nsend-nreceived)/nsend*100);
```

```
    close(sockfd);
```

```
    exit(1);
```

```
}
```

```
/*校验和算法*/
```

```
unsigned short cal_chksum(unsigned short *addr, int len)
```

```
{    int nleft=len;
```

```
    int sum=0;
```

```
    unsigned short *w=addr;
```

```
    unsigned short answer=0;
```

```
/*把 ICMP 报头二进制数据以 2 字节为单位累加起来*/
```

```
while(nleft>1)
```

```
{    sum+=*w++;
```

```
    nleft-=2;
```

```
}
```

```
/*若 ICMP 报头为奇数个字节，会剩下最后一字节。把最后一个字节视为一个 2 字节数据的
```

高

```
//字节, 这个 2 字节数据的低字节为 0, 继续累加*/
if( nleft==1)
{
    *(unsigned char *)(&answer)=*(unsigned char *)w;
    sum+=answer;
}
sum=(sum>>16)+(sum&0xffff);
sum+=(sum>>16);
answer=~sum;
return answer;
}

/*设置 ICMP 报头*/
int pack(int pack_no)
{
    int i,packsize;
    struct icmp *icmp;
    struct timeval *tval;

    icmp=(struct icmp*)sendpacket;
    icmp->icmp_type=ICMP_ECHO;
    icmp->icmp_code=0;
    icmp->icmp_cksum=0;
    icmp->icmp_seq=pack_no;
    icmp->icmp_id=pid;
    packsize=8+datalen;
    tval= (struct timeval *)icmp->icmp_data;
    gettimeofday(tval,NULL);    /*记录发送时间*/
    icmp->icmp_cksum=cal_chksum( (unsigned short *)icmp,packsize); /*校验算法*/
    return packsize;
}

/*发送三个 ICMP 报文*/
void send_packet()
{
    int packsize;
    while( nsend<MAX_NO_PACKETS) //发送 MAX_NO_PACKETS 个 ICMP 报文
    {
        nsend++;
        packsize=pack(nsend); /*设置 ICMP 报头*/
        //sendpacket 为要发送的内容, 由 pack()函数设定, dest_addr 是目的地址,
        if( sendto(sockfd,sendpacket,packsize,0,
                    (struct sockaddr *)&dest_addr,sizeof(dest_addr) )<0 )
        {
            perror("sendto error");
            continue;
        }
        sleep(1); /*每隔一秒发送一个 ICMP 报文*/
    }
}
```

```

    }
}

/*接收所有 ICMP 报文*/
void recv_packet()
{
    int n,fromlen;
    extern int errno;

    signal(SIGALRM,statistics);
    fromlen=sizeof(from);
    while( nreceived<nsend)
    {
        //alarm()用来设置信号 SIGALRM 在经过参数 seconds 指定的秒数后传送给目前的
        进程

        alarm(MAX_WAIT_TIME);
        if( (n=recvfrom(sockfd,recvpacket,sizeof(recvpacket),0,
                        (struct sockaddr *)&from,&fromlen)) <0)
        {
            if(errno==EINTR) continue;
            perror("recvfrom error");
            continue;
        }
        gettimeofday(&tvrecv,NULL); /*记录接收时间*/
        if(unpack(recvpacket,n)==-1)continue;
        nreceived++;
    }
}

/*剥去 ICMP 报头*/
int unpack(char *buf,int len)
{
    int i,iphdrln;
    struct ip *ip;
    struct icmp *icmp;
    struct timeval *tvsend;
    double rtt;

    ip=(struct ip *)buf;
    //求 ip 报头长度,即 ip 报头的长度标志乘 4,头长度指明头中包含的 4 字节字的个数。可接受
    //的最小值是 5, 最大值是 15
    iphdrln=ip->ip_hl<<2;
    icmp=(struct icmp *)(buf+iphdrln); /*越过 ip 报头,指向 ICMP 报头*/
    len-=iphdrln; /*ICMP 报头及 ICMP 数据报的总长度*/
    if( len<8) /*小于 ICMP 报头长度则不合理*/
    {
        printf("ICMP packets\'s length is less than 8\n");
        return -1;
    }
}

```

```

    }
    /*确保所接收的是我所发的的 ICMP 的回应*/
    if( (icmp->icmp_type==ICMP_ECHOREPLY) && (icmp->icmp_id==pid) )
    {
        tvsend=(struct timeval *)icmp->icmp_data;
        tv_sub(&tvrecv,tvsend); /*接收和发送的时间差*/
        rtt=tvrecv.tv_sec*1000+tvrecv.tv_usec/1000; /*以毫秒为单位计算 rtt*/
        /*显示相关信息*/
        printf("%d byte from %s: icmp_seq=%u ttl=%d rtt=%.3f ms\n",
                len,inet_ntoa(from.sin_addr),icmp->icmp_seq,ip->ip_ttl,rtt);
    }
    else    return -1;
}

int main(int argc,char *argv[])
{
    struct hostent *host;
    struct protoent *protocol;
    unsigned long inaddr=0l;
    int waittime=MAX_WAIT_TIME;    //define MAX_WAIT_TIME    5
    int size=50*1024;

    if(argc<2)
    {
        printf("usage:%s hostname/IP address\n",argv[0]);
        exit(1);
    }
    //getprotobyname("icmp")返回对应于给定协议名的包含名字和协议号的 protoent 结构指针。
    if( (protocol=getprotobyname("icmp"))==NULL)
    {
        perror("getprotobyname");
        exit(1);
    }
    /*生成使用 ICMP 的原始套接字,这种套接字只有 root 才能生成*/
    if( (sockfd=socket(AF_INET,SOCK_RAW,protocol->p_proto) )<0)
    {
        perror("socket error");
        exit(1);
    }
    /* 回收 root 权限,设置当前用户权限*/
    setuid(getuid());
    /*扩大套接字接收缓冲区到 50K 这样做主要为了减小接收缓冲区溢出的
    的可能性,若无意中 ping 一个广播地址或多播地址,将会引来大量应答*/
    setsockopt(sockfd,SOL_SOCKET,SO_RCVBUF,&size,sizeof(size) );
    bzero(&dest_addr,sizeof(dest_addr));
    dest_addr.sin_family=AF_INET;

    /*判断是主机名还是 ip 地址*/
    if( inaddr=inet_addr(argv[1])==INADDR_NONE)

```



```

{
    if((host=gethostbyname(argv[1]))==NULL) /*是主机名*/
    {
        perror("gethostbyname error");
        exit(1);
    }
    memcpy( (char *)&dest_addr.sin_addr,host->h_addr,host->h_length);
}
else /*是 ip 地址*/
    dest_addr.sin_addr.s_addr = inet_addr(argv[1]);
/*获取 main 的进程 id,用于设置 ICMP 的标志符*/
pid=getpid();
printf("PING %s(%s): %d bytes data in ICMP packets.\n",argv[1],
        inet_ntoa(dest_addr.sin_addr),datalen);
send_packet(); /*发送所有 ICMP 报文*/
recv_packet(); /*接收所有 ICMP 报文*/
statistics(SIGALRM); /*进行统计*/
return 0;
}
/*两个 timeval 结构相减*/
void tv_sub(struct timeval *out,struct timeval *in)
{
    if( (out->tv_usec-=in->tv_usec)<0)
    {
        --out->tv_sec;
        out->tv_usec+=1000000;
    }
    out->tv_sec-=in->tv_sec;
}

```

好了，编译，运行。按照正常的 ping 程序运行(当然了，这个程序还是很简单，目前为止还不支持环回地址 127.0.0.1)。结果告诉我，小王微微的笑了(调皮的很，还不想让我看见)，我也就开心了..