

```

/*
 *
 *
 * Using the InterNet Control Message Protocol (ICMP) "ECHO" facility,
 * measure round-trip-delays and packet loss across network paths.
 *
 * Author -
 *      Mike Muuss
 *      U. S. Army Ballistic Research Laboratory
 *      December, 1983
 * Modified at Uc Berkeley
 *
 * Changed argument to inet_ntoa() to be struct in_addr instead of u_long
 * DFM BRL 1992
 *
 * Status -
 *      Public Domain.  Distribution Unlimited.
 *
 * Bugs -
 *      More statistics could always be gathered.
 *      This program has to run SUID to ROOT to access the ICMP socket.
 */

#include <stdio.h>
#include <errno.h>
#include <sys/time.h>

#include <sys/param.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/file.h>

#include <netinet/in_sysm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>

#define MAXWAIT      10      /* max time to wait for response, sec. */
#define MAXPACKET    4096    /* max packet size */
#define VERBOSE      1      /* verbose flag */
#define QUIET        2      /* quiet flag */
#define FLOOD        4      /* flooding flag */
#ifdef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 64
#endif

u_char  packet[MAXPACKET];
int      i, pingflags, options;
extern  int  errno;

int s;          /* Socket file descriptor */
struct hostent *hp; /* Pointer to host info */
struct timezone tz; /* leftover */

struct sockaddr whereto; /* Who to ping */
int datalen; /* How much data */

char usage[] =
"Usage: ping [-dfqrv] host [packetsize [count [preload]]]n";

char *hostname;
char hnamebuf[MAXHOSTNAMELEN];

int npackets;

int preload = 0; /* number of packets to "preload" */
int ntransmitted = 0; /* sequence # for outbound packets = #sent */
int ident;

int nreceived = 0; /* # of packets we got back */

```

```

int timing = 0;
int tmin = 999999999;
int tmax = 0;
int tsum = 0; /* sum of all times, for doing average */
int finish(), catcher();
char *inet_ntoa();

/*
 *          M A I N
 */
main(argc, argv)
char *argv[];
{
    struct sockaddr_in from;
    char **av = argv;
    struct sockaddr_in *to = (struct sockaddr_in *) &where;
    int on = 1;
    struct protoent *proto;

    argc--, av++;
    while (argc > 0 && *av[0] == '-') {
        while (*++av[0]) switch (*av[0]) {
            case 'd':
                options |= SO_DEBUG;
                break;
            case 'r':
                options |= SO_DONTROUTE;
                break;
            case 'v':
                pingflags |= VERBOSE;
                break;
            case 'q':
                pingflags |= QUIET;
                break;
            case 'f':
                pingflags |= FLOOD;
                break;
        }
        argc--, av++;
    }
    if(argc < 1 || argc > 4) {
        printf(usage);
        exit(1);
    }

    bzero((char *)&where, sizeof(struct sockaddr));
    to->sin_family = AF_INET;
    to->sin_addr.s_addr = inet_addr(av[0]);
    if(to->sin_addr.s_addr != (unsigned)-1) {
        strcpy(hnamebuf, av[0]);
        hostname = hnamebuf;
    } else {
        hp = gethostbyname(av[0]);
        if (hp) {
            to->sin_family = hp->h_addrtype;
            bcopy(hp->h_addr, (caddr_t)&to->sin_addr, hp->h_length);
            hostname = hp->h_name;
        } else {
            printf("%s: unknown host %sn", argv[0], av[0]);
            exit(1);
        }
    }

    if( argc >= 2 )
        datalen = atoi( av[1] );
    else
        datalen = 64-8;
    if (datalen > MAXPACKET) {
        fprintf(stderr, "ping: packet size too largen");
        exit(1);
    }
}

```

```

if (datalen >= sizeof(struct timeval)) /* can we time 'em? */
    timing = 1;

if (argc >= 3)
    npackets = atoi(av[2]);

if (argc == 4)
    preload = atoi(av[3]);

ident = getpid() & 0xFFFF;

if ((proto = getprotobyname("icmp")) == NULL) {
    fprintf(stderr, "icmp: unknown protocol\n");
    exit(10);
}

if ((s = socket(AF_INET, SOCK_RAW, proto->p_proto)) < 0) {
    perror("ping: socket");
    exit(5);
}

if (options & SO_DEBUG) {
    if(pingflags & VERBOSE)
        printf("...debug on.n");
    setsockopt(s, SOL_SOCKET, SO_DEBUG, &on, sizeof(on));
}

if (options & SO_DONTROUTE) {
    if(pingflags & VERBOSE)
        printf("...no routing.n");
    setsockopt(s, SOL_SOCKET, SO_DONTROUTE, &on, sizeof(on));
}

if(to->sin_family == AF_INET) {
    printf("PING %s (%s): %d data bytesn", hostname,
        inet_ntoa(to->sin_addr), datalen); /* DFM */
} else {
    printf("PING %s: %d data bytesn", hostname, datalen );
}

setlinebuf( stdout );

signal( SIGINT, finish );
signal(SIGALRM, catcher);

/* fire off them quickies */
for(i=0; i < preload; i++)
    pinger();

if(!(pingflags & FLOOD))
    catcher(); /* start things going */

for (;;) {
    int len = sizeof (packet);
    int fromlen = sizeof (from);
    int cc;
    struct timeval timeout;
    int fdmask = 1 << s;

    timeout.tv_sec = 0;
    timeout.tv_usec = 10000;

    if(pingflags & FLOOD) {
        pinger();
        if( select(32, &fdmask, 0, 0, &timeout) == 0)
            continue;
    }

    if ( (cc=recvfrom(s, packet, len, 0, &from, &fromlen)) < 0) {
        if( errno == EINTR )
            continue;
        perror("ping: recvfrom");
        continue;
    }

    pr_pack( packet, cc, &from );
}

```

```

        if (npackets && nreceived >= npackets)
            finish();
    }
    /*NOTREACHED*/
}

/*
 *
 *          C A T C H E R
 *
 * This routine causes another PING to be transmitted, and then
 * schedules another SIGALRM for 1 second from now.
 *
 * Bug -
 * Our sense of time will slowly skew (ie, packets will not be launched
 * exactly at 1-second intervals). This does not affect the quality
 * of the delay and loss statistics.
 */
catcher()
{
    int waittime;

    pinger();
    if (npackets == 0 || ntransmitted < npackets)
        alarm(1);
    else {
        if (nreceived) {
            waittime = 2 * tmax / 1000;
            if (waittime == 0)
                waittime = 1;
        } else
            waittime = MAXWAIT;
        signal(SIGALRM, finish);
        alarm(waittime);
    }
}

/*
 *
 *          P I N G E R
 *
 * Compose and transmit an ICMP ECHO REQUEST packet. The IP packet
 * will be added on by the kernel. The ID field is our UNIX process ID,
 * and the sequence number is an ascending integer. The first 8 bytes
 * of the data portion are used to hold a UNIX "timeval" struct in VAX
 * byte-order, to compute the round-trip time.
 */
pinger()
{
    static u_char outpack[MAXPACKET];
    register struct icmp *icp = (struct icmp *) outpack;
    int i, cc;
    register struct timeval *tp = (struct timeval *) &outpack[8];
    register u_char *datap = &outpack[8+sizeof(struct timeval)];

    icp->icmp_type = ICMP_ECHO;
    icp->icmp_code = 0;
    icp->icmp_cksum = 0;
    icp->icmp_seq = ntransmitted++;
    icp->icmp_id = ident;          /* ID */

    cc = datalen+8;                /* skips ICMP portion */

    if (timing)
        gettimeofday( tp, &tz );

    for( i=8; i<datalen; i++)      /* skip 8 for time */
        *datap++ = i;

    /* Compute ICMP checksum here */
    icp->icmp_cksum = in_cksum( icp, cc );

    /* cc = sendto(s, msg, len, flags, to, tolen) */

```

```
i = sendto( s, outpack, cc, 0, &where, sizeof(struct sockaddr) );
```

```
if( i < 0 || i != cc ) {
    if( i < 0 ) perror("sendto");
    printf("ping: wrote %s %d chars, ret=%dn",
        hostname, cc, i );
    fflush(stdout);
}
if(pingflags == FLOOD) {
    putchar('.');
    fflush(stdout);
}
}
```

```
/*
 *
 * P R _ T Y P E
 *
 * Convert an ICMP "type" field to a printable string.
 */
```

```
char *
pr_type( t )
register int t;
{
    static char *ttab[] = {
        "Echo Reply",
        "ICMP 1",
        "ICMP 2",
        "Dest Unreachable",
        "Source Quench",
        "Redirect",
        "ICMP 6",
        "ICMP 7",
        "Echo",
        "ICMP 9",
        "ICMP 10",
        "Time Exceeded",
        "Parameter Problem",
        "Timestamp",
        "Timestamp Reply",
        "Info Request",
        "Info Reply"
    };

    if( t < 0 || t > 16 )
        return("OUT-OF-RANGE");

    return(ttab[t]);
}
```

```
/*
 *
 * P R _ P A C K
 *
 * Print out the packet, if it came from us. This logic is necessary
 * because ALL readers of the ICMP socket get a copy of ALL ICMP packets
 * which arrive ('tis only fair). This permits multiple copies of this
 * program to be run without having intermingled output (or statistics!).
 */
```

```
pr_pack( buf, cc, from )
char *buf;
int cc;
struct sockaddr_in *from;
{
    struct ip *ip;
    register struct icmp *icp;
    register long *lp = (long *) packet;
    register int i;
    struct timeval tv;
    struct timeval *tp;
    int hlen, triptime;

    from->sin_addr.s_addr = ntohl( from->sin_addr.s_addr );
```

```

gettimeofday( &tv, &tz );

ip = (struct ip *) buf;
hlen = ip->ip_hl << 2;
if (cc < hlen + ICMP_MINLEN) {
    if (pingflags & VERBOSE)
        printf("packet too short (%d bytes) from %sn", cc,
            inet_ntoa(ntohl(from->sin_addr))); /* DFM */
    return;
}
cc -= hlen;
icp = (struct icmp *) (buf + hlen);
if( (!(pingflags & QUIET)) && icp->icmp_type != ICMP_ECHOREPLY ) {
    printf("%d bytes from %s: icmp_type=%d (%s) icmp_code=%dn",
        cc, inet_ntoa(ntohl(from->sin_addr)),
        icp->icmp_type, pr_type(icp->icmp_type), icp->icmp_code); /*DFM*/
    if (pingflags & VERBOSE) {
        for( i=0; i<12; i++)
            printf("x%2.2x: x%8.8xn", i*sizeof(long),
                *lp++);
    }
    return;
}
if( icp->icmp_id != ident )
    return; /* 'Twas not our ECHO */

if (timing) {
    tp = (struct timeval *)&icp->icmp_data[0];
    tvsub( &tv, tp );
    triptime = tv.tv_sec*1000+(tv.tv_usec/1000);
    tsum += triptime;
    if( triptime < tmin )
        tmin = triptime;
    if( triptime > tmax )
        tmax = triptime;
}

if(!(pingflags & QUIET)) {
    if(pingflags != FLOOD) {
        printf("%d bytes from %s: icmp_seq=%d", cc,
            inet_ntoa(from->sin_addr),
            icp->icmp_seq ); /* DFM */
        if (timing)
            printf(" time=%d msn", triptime );
        else
            putchar('n');
    } else {
        putchar('b');
        fflush(stdout);
    }
}
nreceived++;
}

```

```

/*
 *
 *
 * Checksum routine for Internet Protocol family headers (C Version)
 *
 */
in_cksum(addr, len)
u_short *addr;
int len;
{
    register int nleft = len;
    register u_short *w = addr;
    register u_short answer;
    register int sum = 0;

    /*

```

```

* Our algorithm is simple, using a 32 bit accumulator (sum),
* we add sequential 16 bit words to it, and at the end, fold
* back all the carry bits from the top 16 bits into the lower
* 16 bits.
*/
while( nleft > 1 ) {
    sum += *w++;
    nleft -= 2;
}

/* mop up an odd byte, if necessary */
if( nleft == 1 ) {
    u_short u = 0;

    *(u_char *)(&u) = *(u_char *)w ;
    sum += u;
}

/*
 * add back carry outs from top 16 bits to low 16 bits
 */
sum = (sum >> 16) + (sum & 0xffff); /* add hi 16 to low 16 */
sum += (sum >> 16); /* add carry */
answer = ~sum; /* truncate to 16 bits */
return (answer);
}

```

```

/*
 *
 * T V S U B
 *
 * Subtract 2 timeval structs: out = out - in.
 *
 * Out is assumed to be >= in.
 */

```

```

tvsub( out, in )
register struct timeval *out, *in;
{
    if( (out->tv_usec -= in->tv_usec) < 0 ) {
        out->tv_sec--;
        out->tv_usec += 1000000;
    }
    out->tv_sec -= in->tv_sec;
}

```

```

/*
 *
 * F I N I S H
 *
 * Print out statistics, and give up.
 * Heavily buffered STDIO is used here, so that all the statistics
 * will be written with 1 sys-write call. This is nice when more
 * than one copy of the program is running on a terminal; it prevents
 * the statistics output from becomming intermingled.
 */

```

```

finish()
{
    putchar('n');
    fflush(stdout);
    printf("n----%s PING Statistics----n", hostname );
    printf("%d packets transmitted, ", ntransmitted );
    printf("%d packets received, ", nreceived );
    if (ntransmitted)
        if( nreceived > ntransmitted)
            printf("-- somebody's printing up packets!");
        else
            printf("%d%% packet loss",
                (int) (((ntransmitted-nreceived)*100) /
                    ntransmitted));
    printf("n");
    if (nreceived && timing)
        printf("round-trip (ms)  min/avg/max = %d/%d/%dn",
            tmin,

```

```
        tsum / nreceived,  
        tmax );  
fflush(stdout);  
exit(0);  
}
```