Experiment 8

Implementation of ‘ping’ Service

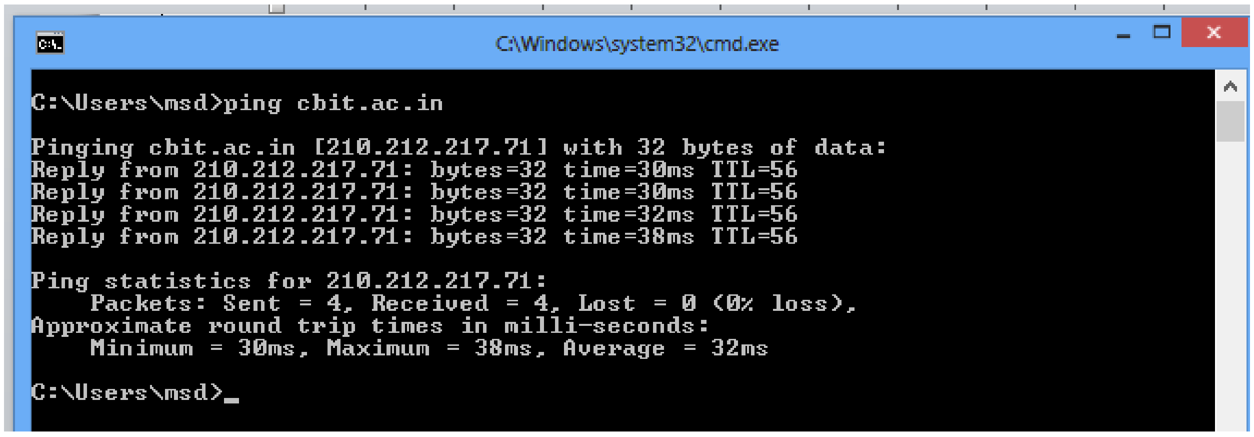
AIM: To implement ‘ping’ service

TOOLS USED: Debian Linux platform, socket API, GCC compiler, TCP/IP stack

DESCRIPTION: **Ping** is network diagnosis tool or utility program used to test the reachability of a host. It runs on top of IP (Internet Protocol). It measure the Round-Trip Time (RTT) for messages sent from the originating host to a destination computer and back. The name comes from active sonar terminology that sends a pulse sound and listens for the echo to detect objects under water; however, the acronym for “PING” meaning "Packet InterNet Groper" has been in use since early days in computing for testing and measuring networks and the Internet.

Ping makes use of ICMP (Internet Message Control Protocol). If sends ICMP **echo request** packets to the target host and waits for the ICMP **echo reply**. It measures the round-trip time from transmission to reception, reporting errors and packet loss. The results of the test usually include a statistical summary of the response packets received, including the minimum, maximum, the mean round-trip times, and usually standard deviation of the mean.

The command-line options for the ping utility and its output vary depending on implementation. Options may include the size of the payload, count of tests, limits for the number of hops (TTL) that probes traverse, and interval between the requests. Many systems provide a companion utility ping6, for similar testing on Internet Protocol version 6 (IPv6) networks. The Figure 7.1 is a screenshot is the output of ‘ping cbit.ac.in’ command.

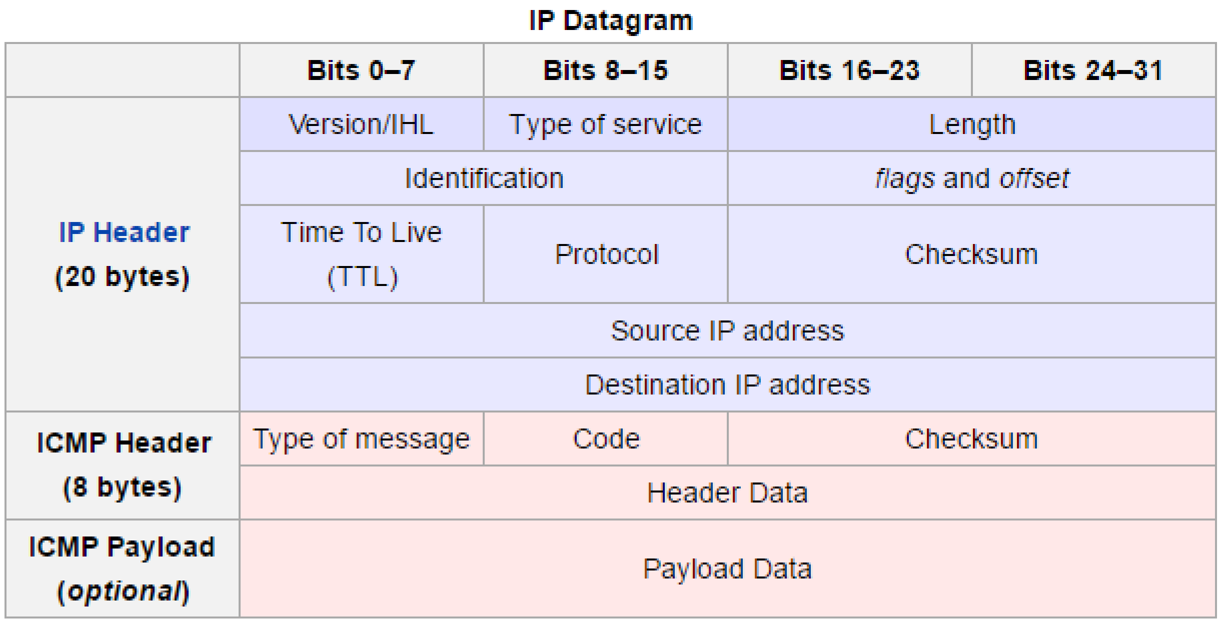


```
C:\Users\msd>ping cbit.ac.in

Pinging cbit.ac.in [210.212.217.71] with 32 bytes of data:
Reply from 210.212.217.71: bytes=32 time=30ms TTL=56
Reply from 210.212.217.71: bytes=32 time=30ms TTL=56
Reply from 210.212.217.71: bytes=32 time=32ms TTL=56
Reply from 210.212.217.71: bytes=32 time=38ms TTL=56

Ping statistics for 210.212.217.71:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 30ms, Maximum = 38ms, Average = 32ms

C:\Users\msd>_
```

Figure 7.1: Sample screenshot of ping command usage

The ping utility was written by Mike Muuss in December 1983 as a tool to troubleshoot problems in an IP network. He was inspired by a remark by David Mills on using ICMP echo packets for IP network diagnosis and measurements. In case of error, the target host or an intermediate router sends back an ICMP error message.

Message format: The ‘ping’ routine uses ICMP protocol and runs on top of IP protocol. Figure 7.2 shows the format of ICMP message. It is a 32-byte packet. The blue portion is IP header format and the pink portion is the ICMP header format.



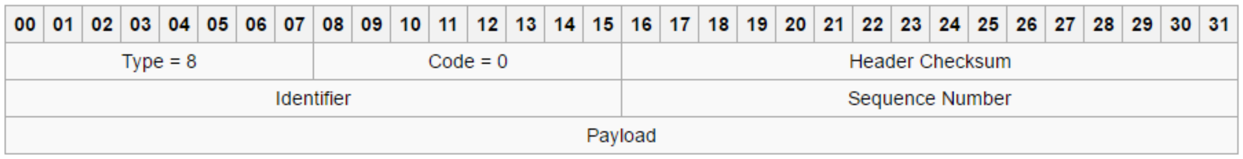
IP Datagram

	Bits 0–7	**Bits 8–15**	**Bits 16–23**	**Bits 24–31**
IP Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

Figure 7.2: ICMP message format

- In IP header, the protocol should be set to 1.
- In the ICMP header, the fields include
  - Type of ICMP message (8-bits)
  - Code (8-bits)
  - Checksum (16-bits) calculated with the ICMP part of the packet. It is the 16-bit 1’s complement of the sum of the ICMP message starting with the Type field.
  - Header Data (32-bits) field, which in this case (**ICMP echo request** and **reply**), will be composed of **identifier** (16-bits) and **sequence number** (16-bits)
  - ICMP Payload: Payload for different kind of answers; can be an arbitrary length, left to implementation details. However, the packet including IP and ICMP headers must be less than the Maximum Transmission Unit (MTU) of the network or risk being fragmented

Echo Request: The echo request (ping) is an ICMP message, as shown in the following Figure: 7.3

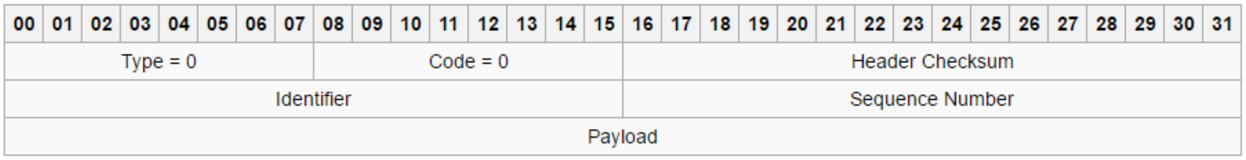


00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type = 8								Code = 0								Header Checksum															
Identifier																Sequence Number															
Payload																															

Figure 7.3: Echo request message

Where type=8, code=0. The identifier and sequence numbers are used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number with that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.

Echo reply: Is an ICMP message generated in response to an echo request; it is mandatory for all hosts and routers, and must include the exact payload received in the request. The format of Echo reply is shown in Figure 7.4, where the Type and code must be set 0. The identifier and sequence numbers can be used by the client to determine which echo requests are associated with the echo replies.



00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type = 0								Code = 0								Header Checksum															
Identifier																Sequence Number															
Payload																															

Figure 7.4: Echo reply

Payload: The payload of the packet is generally filled with ASCII characters, as the output of the tcpdump utility shows:

```
16:24:47.966461 IP (tos 0x0, ttl 128, id 15103, offset 0, flags [none],
proto: ICMP (1), length: 60) 192.168.146.22 > 192.168.144.5: ICMP echo
request,
id 1, seq 38, length 40
       0x0000:  4500 003c 3aff 0000 8001 5c55 c0a8 9216  E..<:.....\U....
       0x0010:  c0a8 9005 0800 4d35 0001 0026 6162 6364  ......M5...&abcd
       0x0020:  6566 6768 696a 6b6c 6d6e 6f70 7172 7374  efghijklmnopqrst
       0x0030:  7576 7761 6263 6465 6667 6869            uvwabcdefghi
```

The payload includes a timestamp of when the message was sent and a sequence number. This allows ping to compute the round trip time in a stateless manner without needing to record when packets were sent.

Security considerations:

The flood ping option exists of many implementations, sending requests as fast as possible in an attempt to determine the response of the network under high-load conditions. That option is restricted to users having administrative privileges, but may be used in **denial-of-service** attacks to induce a ping flood, in which the attacker attempts to overwhelm the victim with ICMP echo requests.

Ping has been considered as a security risk as merely acknowledging a host's presence turns it into a potential target.[9] For these reasons, many systems provide means to disable the reply,[10][11] despite the fact that RFC 1122 mandates hosts to always send a reply.

Host discovery, scanning or ping sweep is a feature of network scanning tools such as nmap, working by utilizing ICMP echo packets.

Algorithms: This can be implemented using two functions. The first function named **listener()** receives a packet and display the information available in the packet. The second function named **ping()** creates an ICMP packet and sends.

Ping listener:

1. Start
2. Create a raw socket by specifying the protocol as IPPROTO_ICMP
3. Receive a packet from the socket by specifying the ping server address
4. Display the reply status (i.e. ping information of error)
5. Repeat steps 3 and 4
6. Stop

Ping:

1. Start
2. Create a raw socket by specifying the protocol as IPPROTO_ICMP
3. Using ***setsockopt()*** system call set the TTL value
4. Set the socket into non-blocking mode by using the ***fcntl()*** system call
5. Create an ICMP packet and fill the fields with the required information
6. Send the filled-in packet
7. Repeat steps 5 and 6
8. stop

Code:

```
/* myping.c
 *
 * Copyright (c) 2000 Sean Walton and Macmillan Publishers.  Use may be in
 * whole or in part in accordance to the General Public License (GPL).
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. */

/* myping.c                                                      */
/*   Use the ICMP protocol to request "echo" from destination.*/
```

```
#include <fcntl.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

#define PACKETSIZE     64
struct packet
{
      struct icmphdr hdr;
      char msg[PACKETSIZE-sizeof(struct icmphdr)];
};

int pid=-1;
struct protoent *proto=NULL;

/*--------------------------------------------------------------------*/
/*--- checksum - standard 1s complement checksum                   ---*/
/*--------------------------------------------------------------------*/
unsigned short checksum(void *b, int len)
{     unsigned short *buf = b;
      unsigned int sum=0;
      unsigned short result;

      for ( sum = 0; len > 1; len -= 2 )
            sum += *buf++;
      if ( len == 1 )
            sum += *(unsigned char*)buf;
      sum = (sum >> 16) + (sum & 0xFFFF);
      sum += (sum >> 16);
      result = ~sum;
      return result;
}

/*--------------------------------------------------------------------*/
/*--- display - present echo info                                  ---*/
/*--------------------------------------------------------------------*/
void display(void *buf, int bytes)
{     int i;
      struct iphdr *ip = buf;
      struct icmphdr *icmp = buf+ip->ihl*4;

      printf("----------------\n");
      for ( i = 0; i < bytes; i++ )
      {
            if ( !(i & 15) )
printf("\n
```

```
X:  ", i);
            printf("1X ", ((unsigned char*)buf)[i]);
      }
```

```
	printf("\n");
	printf("IPv%d: hdr-size=%d pkt-size=%d protocol=%d TTL=%d src=%s ",
		ip->version, ip->ihl*4, ntohs(ip->tot_len), ip->protocol,
		ip->ttl, inet_ntoa(ip->saddr));
	printf("dst=%s\n", inet_ntoa(ip->daddr));
	if ( icmp->un.echo.id == pid )
	{
		printf("ICMP: type[%d/%d] checksum[%d] id[%d] seq[%d]\n",
			icmp->type, icmp->code, ntohs(icmp->checksum),
			icmp->un.echo.id, icmp->un.echo.sequence);
	}
}

/*--------------------------------------------------------------------*/
/*--- listener - separate process to listen for and collect messages--*/
/*--------------------------------------------------------------------*/
void listener(void)
{	int sd;
	struct sockaddr_in addr;
	unsigned char buf[1024];

	sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
	if ( sd < 0 )
	{
		perror("socket");
		exit(0);
	}
	for (;;)
	{	int bytes, len=sizeof(addr);

		bzero(buf, sizeof(buf));
		bytes = recvfrom(sd, buf, sizeof(buf), 0, (struct
sockaddr*)&addr, &len);
		if ( bytes > 0 )
			display(buf, bytes);
		else
			perror("recvfrom");
	}
	exit(0);
}

/*--------------------------------------------------------------------*/
/*--- ping - Create message and send it.                           ---*/
/*--------------------------------------------------------------------*/
void ping(struct sockaddr_in *addr)
{	const int val=255;
	int i, sd, cnt=1;
	struct packet pckt;
	struct sockaddr_in r_addr;

	sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
	if ( sd < 0 )
	{
		perror("socket");
```

```
            return;
        }
        if ( setsockopt(sd, SOL_IP, IP_TTL, &val, sizeof(val)) != 0)
            perror("Set TTL option");
        if ( fcntl(sd, F_SETFL, O_NONBLOCK) != 0 )
            perror("Request nonblocking I/O");
        for (;;)
        {   int len=sizeof(r_addr);

            printf("Msg #%d\n", cnt);
            if ( recvfrom(sd, &pckt, sizeof(pckt), 0, (struct 
sockaddr*)&r_addr, &len) > 0 )
                printf("***Got message!***\n");
            bzero(&pckt, sizeof(pckt));
            pckt.hdr.type = ICMP_ECHO;
            pckt.hdr.un.echo.id = pid;
            for ( i = 0; i < sizeof(pckt.msg)-1; i++ )
                pckt.msg[i] = i+'0';
            pckt.msg[i] = 0;
            pckt.hdr.un.echo.sequence = cnt++;
            pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));
            if ( sendto(sd, &pckt, sizeof(pckt), 0, (struct 
sockaddr*)addr, sizeof(*addr)) <= 0 )
                perror("sendto");
            sleep(1);
        }
}

/*--------------------------------------------------------------------*/
/*--- main - look up host and start ping processes.                ---*/
/*--------------------------------------------------------------------*/
int main(int count, char *strings[])
{   struct hostent *hname;
    struct sockaddr_in addr;

    if ( count != 2 )
    {
        printf("usage: %s <addr>\n", strings[0]);
        exit(0);
    }
    if ( count > 1 )
    {
        pid = getpid();
        proto = getprotobyname("ICMP");
        hname = gethostbyname(strings[1]);
        bzero(&addr, sizeof(addr));
        addr.sin_family = hname->h_addrtype;
        addr.sin_port = 0;
        addr.sin_addr.s_addr = *(long*)hname->h_addr;
        if ( fork() == 0 )
            listener();
        else
            ping(&addr);
        wait(0);
```

```
	}
	else
		printf("usage: myping <hostname>\n");
	return 0;
}
```

RESULTS AND DISCUSSION:

- run the above program and give your observations
- implement the above program by writing the ping client and server separately
- refine the above code to include options like in actual ping program

CONCLUSIONS:

1. Understood about ping service
2. Successfully implemented and tested with several options
3. This can be extended to simulate the actual ping program.

References:

1. https://en.wikipedia.org/wiki/Ping_(networking_utility)
2. https://www.cs.utah.edu/~swalton/listings/sockets/programs/part4/chap18/myping.c