Raw Sockets and ICMP

Srinidhi Varadarajan

Topics
- Raw sockets
- Internet Control Message Protocol (ICMP)
- Code Examples
  - Ping
  - Traceroute

Raw Sockets
- Usually, sockets are used to build applications on top of a transport protocol
  - Stream sockets (TCP)
  - Datagram sockets (UDP)
- Some applications need to access a lower layer protocol
  - Control protocols built on IP rather than UDP or TCP, such as ICMP and IGMP
  - Experimental transport protocols
- A “raw” socket allows direct access to IP
  - Used to build applications on top of the network layer

Creating a Raw Socket
- Standard socket() call used to create a raw socket
  - Family is AF_INET, as for TCP or UDP
  - Socket type is SOCK_RAW instead of SOCK_STREAM or SOCK_DGRAM
  - Socket protocol needs to be specified, e.g., IPPROTO_ICMP (often left at 0 for UDP or TCP sockets)

socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)

Socket Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream socket</td>
<td>SOCK_STREAM 1</td>
</tr>
<tr>
<td>Datagram socket</td>
<td>SOCK_DGRAM 2</td>
</tr>
<tr>
<td>Raw protocol interface</td>
<td>SOCK_RAW 3</td>
</tr>
<tr>
<td>Reliably delivered message</td>
<td>SOCK_RDM 4</td>
</tr>
<tr>
<td>Sequenced packet stream</td>
<td>SOCK_SEQPACKET 5</td>
</tr>
</tbody>
</table>

Protocols
- Protocol values
  - Used to define the Protocol field in the IP header
Internet Control Message Protocol

- ICMP defined in RFC 792
- ICMP messages
  - Query network node(s) for information
  - Report error conditions
- ICMP messages are carried as IP datagrams
  - ICMP “uses” or is “above” IP
- ICMP messages usually processed by IP, UDP, or TCP
  - IP, TCP, and UDP “use” or are above ICMP

ICMP in the TCP/IP Suite

ICMP Message Format (1)

- ICMP messages are encapsulated in IP datagrams
  - IP-level routing use to move ICMP messages through a network
  - IP provides multiplexing/demultiplexing based on protocol number (IPPROTO_ICMP = 1)

<table>
<thead>
<tr>
<th>IP Header</th>
<th>ICMP Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bytes</td>
<td>variable length</td>
</tr>
</tbody>
</table>

ICMP Message Format (2)

- `TYPE`: Type of ICMP message
- `CODE`: Used by some types to indicate a specific condition
- `CHECKSUM`: Checksum over full message
- Contents depend on `TYPE` and `CODE`

Example ICMP Message Types

- Queries
  - `TYPE = 8`: Echo request
  - `TYPE = 0`: Echo reply
  - `TYPE = 13`: Time stamp request
  - `TYPE = 14`: Time stamp reply
- Errors
  - `TYPE = 3`: Destination unreachable
    - `CODE = 0`: Network unreachable
    - `CODE = 1`: Host unreachable
    - `CODE = 2`: Protocol unreachable
    - `CODE = 3`: Port unreachable
  - `TYPE = 11`: Time exceeded
    - `CODE = 0`: Time-to-live equals 0 in transit

Error Example: Port Unreachable

- Port unreachable error occurs when a receiving host receives a packet with an unknown (inactive) port number
- IP datagram is valid -- reaches addressed host
- UDP datagram contains a port that is not in use (e.g. 8000 and no application has a socket bound to an address with that port)
- UDP replies with an ICMP “Destination Unreachable/Port Unreachable” message
  - `TYPE = 3`, `CODE = 3`
ICMP Error Messages

- ICMP error messages include header and first 8 bytes of offending IP datagram
  - All of IP header
  - Destination address, protocol number, etc.
  - For UDP, all of UDP header including source and destination port numbers
- ICMP message for port unreachable

```
+----------------+----------------+----------------+----------------+
| IP Header      | ICMP Header     | Offending IP   |
| 20             | 8              | Header         |
|                | 20             | 8              |
+----------------+----------------+----------------+
```

Ping Example

- “Ping” utility
  - Tests whether or not a host is reachable
  - Provides a round-trip time
  - Written by Mike Muuss in 1983 to diagnose network problems
- Operation
  - ICMP echo request (TYPE = 8) sent to host
  - Host replies with ICMP echo reply (TYPE = 0)
- Client-server roles
  - Host sending echo request is the client
  - Host sending echo reply is the server
  - Server usually implemented in TCP/IP code

Ping Algorithm

1) Initialize echo request
2) Send echo request
3) Wait for echo reply (or time out)
4) Receive reply
5) Report results
6) Go back to 1 until complete

Echo Request/Reply Format (1)

```
+-------+-------+-------+
| TYPE  | CODE  | CHECKSUM |
| 0     | 0     | 0      |
+-------+-------+-------+
| IDENTIFIER |   |   |
| 0     | 0     | 0      |
+-------+-------+-------+
| SEQUENCE NUMBER |   |
| 0     | 0     | 0      |
+-------+-------+-------+
```

- IDENTIFIER: Means to identify sending instance of “ping”
  - Process id in UNIX
- SEQUENCE NUMBER: Means to identify lost or misordered replies

Echo Request/Reply Format (2)

```
typedef struct tagICMPHDR
{
  u_char Type; // Type
  u_char Code; // Code
  u_short Checksum; // Checksum
  u_short ID; // Identification
  u_short Seq; // Sequence
} ICMPHDR, *PICMPHDR;
```

```
 typedef struct tagECHOREQUEST
{
  ICMPHDR icmpHdr; // Header int dwTime; // Time char cData[REQ_DATASIZE]; // Fill data } ECHOREQUEST, *PECHOREQUEST;
```

Echo Request

- Echo request will include
  - Common request/reply header
  - Time stamp (32 bits)
  - Filler data (REQ_DATASIZE bytes)

```
typedef struct tagECHOREQUEST
{
  ICMPHDR icmpHdr; // Header int dwTime; // Time char cData[REQ_DATASIZE]; // Fill data } ECHOREQUEST, *PECHOREQUEST;
```

```
static ECHOREQUEST echo_req;
```
Initializing the Echo Request

```c
// Fill in some data to send
memset(echo_req.cData, ' ', REQ_DATASIZE);
```

```
// Save tick count when sent (milliseconds)
echo_req.dwTime = gettime(...);
```

```
// Put data in packet and compute checksum
echo_req.icmpHdr.Checksum = in_cksum(...);
```

Waiting for Echo Reply

```
readfds.fd_count = 1; // set size
readfds.fd_array[0] = raw; // socket set
timeout.tv_sec = 10; // timeout (s)
timeout.tv_usec = 0; // timeout (us)
if((rc = select(1, &readfds, NULL, NULL, &timeout)) == SOCKET_ERROR)
  errexit("select() failed %d
", perror());
```

Echo Reply

- **Raw socket returns IP header**
- **Received datagram contains**
  - IP header
  - ICMP echo request/reply header
  - Echo request message
  - Potentially, additional fill data

```c
typedef struct tagECHOREPLY
{
    IPHDR ipHdr;
    ECHOREQUEST echoRequest;
    char cFiller[256];
} ECHOREPLY, *PECHOREPLY;
```

IP Header (1)

### IP Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Header Length</td>
<td>4</td>
</tr>
<tr>
<td>Total Length</td>
<td>20</td>
</tr>
<tr>
<td>Source IP Address</td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td></td>
</tr>
</tbody>
</table>

IP Header (2)

```c
typedef struct tagIPHDR
{
    u_char VIHL;       // Ver, Hdr length
    u_char TOS;        // Type of service
    short TotLen;      // Total length
    short ID;          // Identification
    short FlagOff;     // Flags, Frag off
    u_char TTL;        // Time-to-live
    u_char Protocol;   // Protocol
    u_short Checksum;  // Checksum
    struct in_addr iaSrc; // Source IP addr
    struct in_addr iaDst; // Dest IP addr
} IPHDR, *PIP HDR;
```

Extracting Results from Reply

- **Ping client can extract IP, ICMP, and echo information from the received datagram**

```c
ECHOREPLY echo_reply;
```

```
type = echo_reply.echoRequest.icmpHdr.Type;
ttl = echo_reply.ipHdr.TTL;
```
### Traceroute Example

- **Traceroute**
  - Reports the route used by an IP datagram from source to destination
  - Provides a round-trip time
  - Written by Van Jacobson as a network diagnostic and debugging tool

- **Operation**
  - Sends ICMP or other datagram toward destination
  - IP time-to-live (TTL) value is controlled to limit extent
  - Intermediate nodes return ICMP time exceeded error -- includes router address

### IP TTL Value

- **IP Time-To-Live Value**: Maximum number of routers through which the datagram may pass
  - Decremented at each router
  - May be decremented once per second
  - Decremented at least once per router
  - Used to prevent looping in the network

- **Basis for Traceroute**

### Traceroute Operation

```
source  router  router  dest
TTL=1 time exceeded
TTL=2 time exceeded
TTL=3 port unreachable
```

- IP packets sent by source (traceroute)
- ICMP packets returned by routers and host

### Traceroute Algorithm

1. Set TTL value to 1
2. Initialize echo request
3. Send echo request
4. Wait for echo reply or time exceeded error (or time out)
5. Receive reply
6. Report results
7. If echo reply, then done; else increment TTL and return to 2

May want to do echo multiple times per TTL

### Setting the TTL Value

- Need to control the IP TTL value
- Raw socket with ICMP does not let us write IP header values
- Use `setsockopt()` to set TTL value

```c
setsockopt(raw, IPPROTO_IP, IP_TTL, (char *) &ttl, sizeof(ttl))
```

- `int on = 1;
  setsockopt(raw, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on))`

### Basic Traceroute Loop

```c
ttl = 0;
do {
    ++ttl;
    if(setsockopt(raw, IPPROTO_IP, IP_TTL, (char *) &ttl, sizeof(ttl)))
        errexit("setsockopt() failed: %d
        perror()");
    done = PingTarget(raw, target_addr);
} while (!done && ttl < MAX_TTL);
```
Potential “Bells and Whistles”

- Multiple pings for each TTL value to better assess round-trip time
- Modify amount of data sent in echo request
- Calculate link delay and other statistics
  - $\text{Delay}[i] = \text{RTT}[i] - \text{RTT}[i-1]$
- Look up intermediate host names using gethostbyaddr()
- Graphical features

ICMP, Ping, Traceroute Reference


You should now be able to …

- Describe the use of ICMP for queries and replies
- Analyze ICMP message format
- Analyze the operation of Ping and Traceroute applications
- Analyze, design, and implement network applications using raw sockets