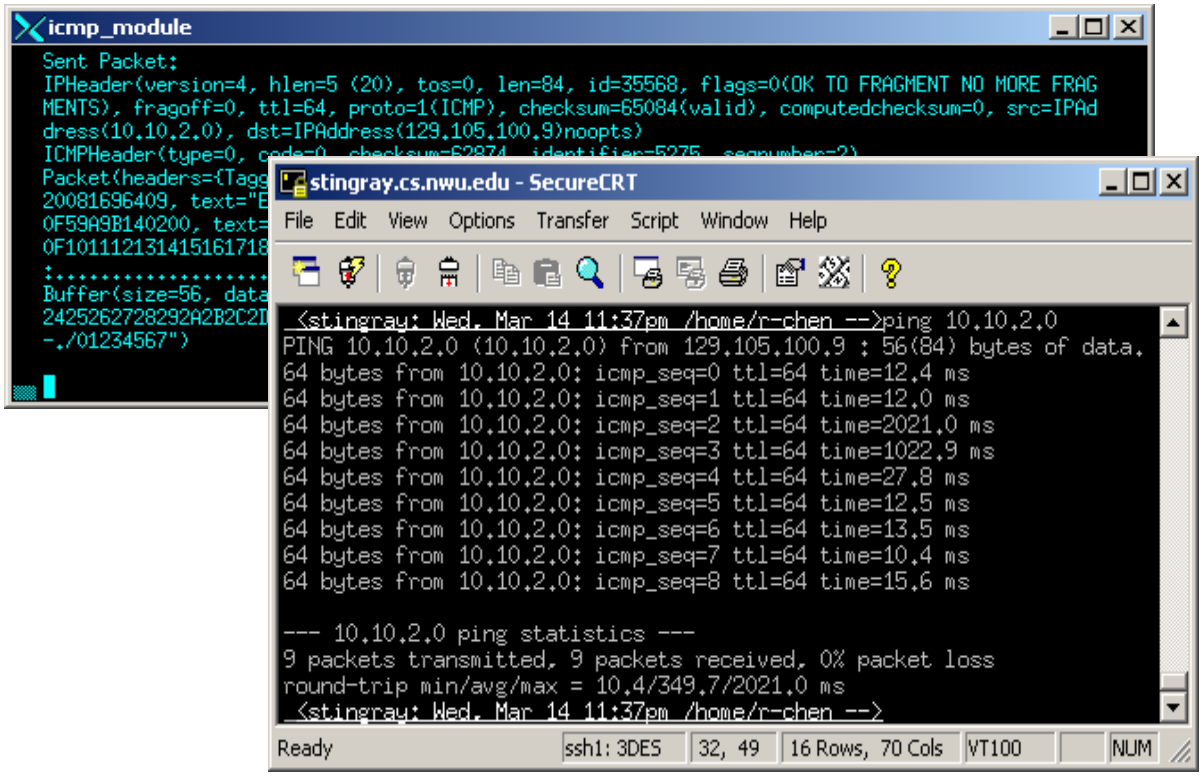
Minet: ICMP Module

Richard S. Chen



Professor Peter Dinda

March 15, 2001

Table of Contents

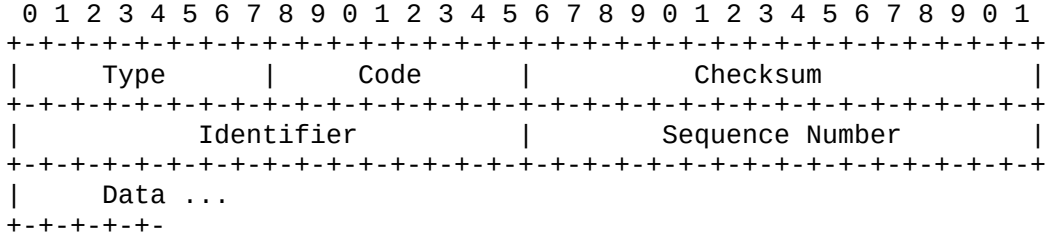
Minet Internet Control Message Protocol Interface

Class ICMPHeader:
ICMPHeader is inherited from the Header class, and is similar in structure to the other header structures.

The constructors are defined as:

- ICMPHeader();
- ICMPHeader(const ICMPHeader &rhs);
- ICMPHeader(const Header &rhs);
- ICMPHeader(const Buffer &rhs);
- ICMPHeader(const char *buf, const unsigned len);

Nearly all ICMP packets share this header structure:



The ICMPHeader class includes functions to access and set the above fields. In addition, accessor functions specific to a particular ICMP message type are also included:

Message Type	Function
ICMP Error Messages (type 3,4,5,11,12)	*ICMP protocol specifies that the IP Header along with the first 64 bits of the datagram be returned in an error message. Functions are available to access and set this data in the payload.*
Redirect Error (type 5)	*Instead of the Identifier and Sequence Number fields, the Redirect error message contains a Gateway Address in that location. Functions are available to access and set the gateway address.*
Parameter Problem Error (type 12)	*Instead of the Identifier and Sequence Number fields, the Parameter Problem error message contains a "pointer" which specifies the octet of the data that generated the error. Functions are available to access and set this pointer field.*
Timestamp Request/ Reply (type 13,14)	*Functions are available to access and set the timestamp information within the payload. A function is also available to retrieve the current time in milliseconds.*
Address Mask Request/ Reply (type 17, 18)	*Functions are available to access and set the address mask reply included within the payload.*

Class icmp_packet:
icmp_packet is inherited from the Packet class. The purpose of this class is to allow for the easy creation and manipulation of ICMP packets. In order to allow for this ease of use, the bulk of the code is implemented directly within the constructor.

The general constructors are defined as:

- icmp_packet();
- icmp_packet(const Packet &rhs);
- icmp_packet(const RawEthernetPacket &rhs);
- icmp_packet(const Packet &original,
  const IPAddress &source, const IPAddress &destination,
  const unsigned char &type, const unsigned char &code,
  const unsigned short &identifier,
  const unsigned short &seqnumber,
  Buffer &payload);

In addition to the general constructors, customized constructors are available to create ICMP request and reply messages. The simplest of these constructors take only a destination IP address and an ICMP message type:

- icmp_packet(const IPAddress &destination,
  const unsigned char &type, const unsigned char &code,
  const unsigned short &identifier,
  const unsigned short &seqnumber);
- icmp_packet(const IPAddress &destination,
  const unsigned char &type, const unsigned char &code);
- icmp_packet(const IPAddress &destination,
  const unsigned char &type);

There are also customized constructors available to create ICMP error messages. ICMP protocol requires that error packets carry the original IP header along with 64 bits of data from the original datagram (this is to allow the sending host to identify the packet that generated the error). The error message constructors take the packet in question as a parameter:

- icmp_packet(const IPAddress &destination,
  const unsigned char &type, const unsigned char &code,
  const unsigned short &identifier,
  const unsigned short &seqnumber,
  const Packet &p);
- icmp_packet(const IPAddress &destination,
  const unsigned char &type, const unsigned char &code,
  const Packet &p);

As mentioned above, much of the functionality of the icmp_packet wrapper class is coded directly within the constructor. However, a function “respond” is also available to handle ICMP requests.

There are two particular variants of this function: one takes a Packet, and the other takes a RawEthernetPacket. The RawEthernetPacket variation simply calls the other function with a Packet created from the raw data. The resulting packet is of the Packet type; thus there would still need to be code to attach an EthernetHeader and convert it back into the RawEthernetPacket type.

Sample calls are as follows:

ICMP Message Type	Sample Code
Reply/ Request	*icmp_packet request("129.105.100.9", ECHO_REQUEST);* *MinetSend(ipmux, request);*
Error	*icmp_packet error("129.105.100.9",* *DESTINATION_UNREACHABLE,* *PROTOCOL_UNREACHABLE, p);* *MinetSend(ip, error);*
Response	*// respond to packet* *icmp_packet response;* *response.respond(p);* *if (response.requires_reply())* *MinetSend(ipmux, response);* *else* *// sent to application layer*

ICMP Request/ Reply Message Types:

ICMP Type: 8 (Echo Request)
ICMP Type: 13 (Timestamp Request)
ICMP Type: 17 (Address Mask Request)
Receive:

Minet is able to receive and automatically respond to all ICMP Request message types. The code to do this is currently implemented in the ICMP Module; it is also possible to implement it within the IP Module.

Send:

There is currently no application layer interface to send ICMP Request messages. Instead, requests are sent by manually creating the packet using the icmp_packet constructor.

ICMP Type: 0 (Echo Reply)
ICMP Type: 13 (Timestamp Reply)
ICMP Type: 18 (Address Mask Reply)
Receive:

Minet is able to receive all ICMP Reply message types. All such replies are forwarded up to the Sock Module, where the appropriate information is displayed.

Send:

Minet is able to automatically respond to all request message types by sending an appropriate reply message. The code that deals with this is in the ICMP Module, although it is also possible to implement it within the IP Module.

ICMP Type: 15 (Information Request)
ICMP Type: 16 (Information Reply)
Obsolete:

RFC 1122 considers these message types obsolete; they have not been implemented in Minet.

ICMP Error Message Types:

ICMP Type: 3 (Destination Unreachable)

Receive:

Minet is able to receive all Destination Unreachable errors. All such error messages are forwarded up to the Sock Module, with the IP header and 64 bits of the original datagram.

Send:

RFC specification requires hosts to be able to generate Protocol Unreachable (code 2), and Port Unreachable (code 3) errors (the other codes are sent by gateways).

Minet is able to automatically generate a Protocol Unreachable error (implemented in the IP Mux Module). Minet is currently unable to automatically generate a Port Unreachable error.

ICMP Type: 4 (Source Quench)

Receive:

Minet is able to receive all Source Quench errors. All such error messages are forwarded up to the Sock Module, with the IP header and 64 bits of the original datagram. Minet is currently unable to reduce the rate by which Minet sends out packets.

Send:

RFC indicates that host implementation of sending Source Quench errors is optional.

Minet supports this error type only as far as being able to manually create Source Quench error messages.

ICMP Type: 5 (Redirect)

Receive:

Minet is able to receive all Redirect errors. All such error messages are forwarded up to the Sock Module, with the IP header and 64 bits of the original datagram. Minet is currently unable to update the route table.

Send:

RFC specification indicates that hosts should not send Redirect ICMP error messages.

Minet supports this error type only as far as being able to manually create Redirect error messages.

ICMP Type: 11 (Time Exceeded)

Receive:

Minet is able to receive all Time Exceeded errors. All such error messages are forwarded up to the Sock Module, with the IP header and 64 bits of the original datagram.

Send:

Minet is able to automatically generate a Time to Live Exceeded Zero code (implemented in the IP Mux Module). Minet is currently unable to automatically generate a Fragment Reassembly Exceeded error.

ICMP Type: 12 (Parameter Problem)

Receive:

Minet is able to receive all Parameter Problem errors. All such error messages are forwarded up to the Sock Module, with the IP header and 64 bits of the original datagram.

Send:

Minet is able to automatically generate an IP Header Bad error if the IP checksum is incorrect (implemented in the IP Module).

Discussion

The Minet ICMP implementation is fully functional in sending, receiving, and processing ICMP Request and Reply messages.

Minet is also fully functional in sending and receiving ICMP Error messages. There are still a few functionalities that need to be implemented in order for Minet to be fully capable of processing received error messages:

Error Type:	Description:
Destination Unreachable (Port Unreachable)	*Minet is currently able to generate Protocol Unreachable errors, although Port Unreachable errors are still not implemented. Implementation should be relatively straightforward, using a simple if… else statement to check to see if the port is valid.*
Source Quench	*Minet is currently unable to reduce the rate by which Minet sends out packets. This functionality is difficult to implement using the current FIFO communication model. Future communication models may be more intuitive in supporting this functionality.*
Redirect	*Minet is currently able to access all the necessary data in the Redirect packet, but it is unable to update the route table using that data. Implementation should be relatively straightforward.*
Time Exceeded (Fragment Reassembly)	*Minet is currently able to generate Time to Live Equals Zero errors, although Fragment Reassembly Exceeded errors are still not implemented. Implementation should be relatively straightforward, using a simple if…else statement to check to see if the fragment reassembly time has been exceeded.*

The work I have done this quarter implements many aspects of ICMP within Minet. The icmp_packet class is able to properly construct both ICMP Request/Reply message types and ICMP Error message types in nearly all layers of the stack. The ICMP Module is able to respond to all ICMP Request messages, and Minet is able to automatically generate and send ICMP Error messages in many of the error situations. In addition, I have implemented a primitive communication interface between the ICMP Module and the Sock Module.

There are still a few things that can be done on the ICMP Module in future work. The above four functionalities should be implemented to allow Minet to correctly respond to all ICMP error types. In addition, future work can include enhancing the interface between the ICMP Module and the Sock Module, as well as implementing a new interface for ICMP between the Sock Module and the Application.

Appendix

Resources:

- TCP/IP Illustrated, Volume 1
  - Addison-Wesley Professional Computing Series
  - W. Richard Stevens

- RFC 792: Internet Control Message Protocol
  - http://www.freesoft.org/CIE/RFC/792/

- RFC 1122: Requirements for Internet Hosts – Communication Layers
  - http://www.freesoft.org/CIE/RFC/1122/