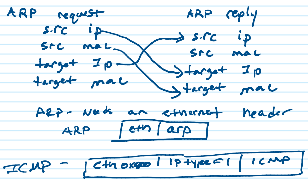
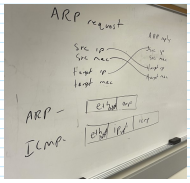
Consolidated Notes

Friday, October 28, 2022 6:05 PM





Internet Control Message Protocol (ICMP)

ICMP will have the following layout:
0

Eth type = 0x800	IP protocol = 1	ICMP type #

ICMP type #:
0 = echo reply (used to ping)
8 = echo request (used to ping)

The ARP header (bytes 12-??) must follow the Ethernet (bytes 0 - 11)
We must parse all the data to see what we have. We will take bytes 0-11 and copy them into a Eth header struct.

Then take bytes 12 to 42(for example) and copy that into an ARP struct.

ARP request have to broadcast everywhere

Eth	**ARP**
Source [source mac]	Operation (request or reply) set based on our needs
Destination [ff:ff:ff:ff:ff:ff] (Broadcast mac address)	Fixed values for every ARP request/reply
Type = 0x806 (ARP Packet)	Hardware type = 1(Ethernet)
	Hardware length = 6 bytes
	Protocol Type = 0x800
	Protocol Length = 4 bytes (for an IP address)
	Source Protocol Address [source IP]
	Source Hardware Address [source mac]
	Target Protocol Address [target IP]
	Target Hardware Address [00:00:00:00:00:00]

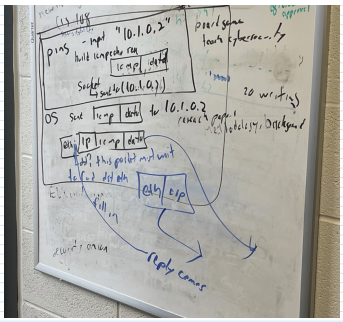
Eth all ff
ARP all 00

0 - 11 bytes 12 - ?? bytes

ARP Reply has to be unicast to who made the request

Eth	**ARP**
Source [target mac]	Operation (request or reply) set based on our needs
Destination [Source mac] (unicast)	Fixed values for every ARP request/reply
Type = 0x806 (ARP Packet)	Hardware type = 1(Ethernet)
	Hardware length = 6 bytes
	Protocol Type = 0x800
	Protocol Length = 4 bytes (for an IP address)
	Source Protocol Address [target IP]
	Source Hardware Address [target mac]
	Target Protocol Address [source IP]
	Target Hardware Address [source mac]

If the requested address does not exit we the requester will not receive an ARP response.
The owner of the address is the only one that can respond.



0 - 11 bytes 12 - ?? bytes

Ignore all other packet types. No other packets should occur

In this event we must send an ICMP "Host unreachable error message"

Or if the packet had too many hops send an ICMP "Time exceeded
If someone pings your router you must respond

Reply

Eth	**IP**	**ICMP Type**
Type = 0x800 (IP Packet)	IP Protocol =1	Type = 0
source mac	if its not 1 avoid it	checksum
dest mac		id
	ip source	sequence
	ip dest	64 bits of data
	source mac	
	dest mac	
	Protocol one means that ICMP is next	

A response to a ping is an ICMP echo response packet

Eth	**IP**	**ICMP Type**
Type = 0x800 (IP Packet)	IP Protocol =1	Type = 8
	if its not 1 avoid it	checksum
Request		id
	ip source	sequence
	ip dest	64 bits of data
	source mac	
	dest mac	
	Protocol one means that ICMP is next	

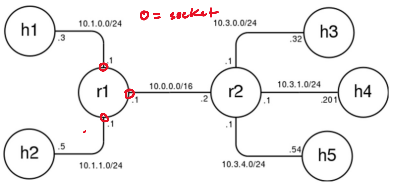
A ping is an ICMP echo request packet

Note for obtaining mac address
Iterate through if config
getif addresses stuff

Not sure if we need this.

Ethernet header	IP header	ICMP header	user data	Ethernet CRC

https://erg.abdn.ac.uk/users/gorry/course/inet-pages/icmp.html



Criteria	Points
Router accepts packets on packet sockets	5
Builds correct ARP response (including ethernet header)	10
Builds correct ICMP echo reply (including ethernet and ip headers)	10
Correctly uses packet socket to send responses as appropriate	5
All of the above work on all router interfaces	5

ping sends icmp echo request
and waits for a echo reply
ARP requests checks for who has the ip you are pinging
if the pinged ip does not exsit do nothing

if the ip exists

ARP http://www.tcpipguide.com/free/t_ARPMessageFormat.htm

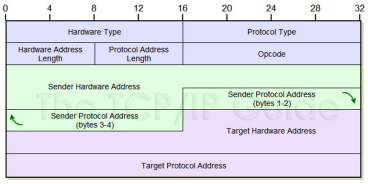


Table 42: Address Resolution Protocol (ARP) Message Format

Field Name	Size (bytes)	Description
HRD	2	***Hardware Type:*** This field specifies the type of hardware used for the local network transmitting the ARP message; thus, it also identifies the type of addressing used. Some of the most common values for this field: *HRD Value* / *Hardware Type* 1 — Ethernet (10 Mb) 6 — IEEE 802 Networks 7 — ARCNET 15 — Frame Relay 16 — Asynchronous Transfer Mode (ATM) 17 — HDLC 18 — Fibre Channel 19 — Asynchronous Transfer Mode (ATM) 20 — Serial Line
PRO	2	***Protocol Type:*** This field is the complement of the *Hardware Type* field, specifying the type of layer three addresses used in the message. For IPv4 addresses, this value is 2048 (0800 hex), which corresponds to the EtherType code for the Internet Protocol.
HLN	1	***Hardware Address Length:*** Specifies how long hardware addresses are in this message. For Ethernet or other networks using IEEE 802 MAC addresses, the value is 6.
PLN	1	***Protocol Address Length:*** Again, the complement of the preceding field; specifies how long protocol (layer three) addresses are in this message. For IP(v4) addresses this value is of course 4.
OP	2	***Opcode:*** This field specifies the nature of the ARP message being sent. The first two values (1 and 2) are used for regular ARP. Numerous other values are also defined to support other protocols that use the ARP frame format, such as RARP, some of which are more widely used than others: *Opcode* / *ARP Message Type* 1 — ARP Request 2 — ARP Reply 3 — RARP Request 4 — RARP Reply 5 — DRARP Request 6 — DRARP Reply 7 — DRARP Error 8 — InARP Request 9 — InARP Reply
SHA	(Variable, equals value in *HLN* field)	***Sender Hardware Address:*** The hardware (layer two) address of the device sending this message (which is the IP datagram source device on a request, and the IP datagram destination on a reply, as discussed in the topic on ARP operation).
SPA	(Variable, equals value in *PLN* field)	***Sender Protocol Address:*** The IP address of the device sending this message.
THA	(Variable, equals value in *HLN* field)	***Target Hardware Address:*** The hardware (layer two) address of the device this message is being sent to. This is the IP datagram destination device on a request, and the IP datagram source on a reply)
TPA	(Variable, equals value in *PLN* field)	***Target Protocol Address:*** The IP address of the device this message is being sent to.

ICMP - http://www.tcpipguide.com/free/t_ICMPv4EchoRequestandEchoReplyMessages-2.htm

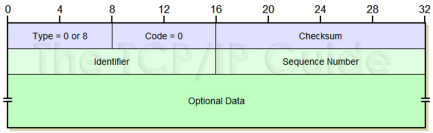


Table 96: ICMPv4 *Echo* and *Echo Reply* Message Format

Field Name	Size (bytes)	Description
Type	1	***Type:*** Identifies the ICMP message type. For *Echo* messages the value is 8, for *Echo Reply* messages the value is 0.
Code	1	***Code:*** Not used for *Echo* and *Echo Reply* messages; set to 0.
Checksum	2	***Checksum:*** 16-bit checksum field for the ICMP header, as described in the topic on the ICMP common message format.
Identifier	2	***Identifier:*** An identification field that can be used to help in matching *Echo* and *Echo Reply* messages.
Sequence Number	2	***Sequence Number:*** A sequence number to help in matching *Echo* and *Echo Reply* messages.
Optional Data	Variable	***Optional Data:*** Additional data to be sent along with the message (not specified.)

IP - http://www.tcpipguide.com/free/t_IPDatagramGeneralFormat.htm

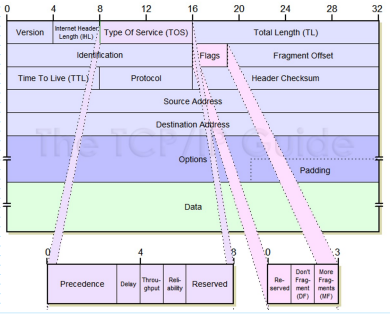
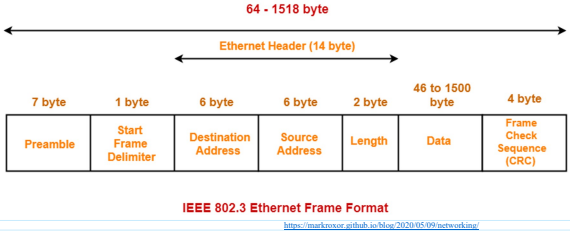


Table 56: Internet Protocol Version 4 (IPv4) Datagram Format

Field Name	Size (bytes)	Description
Version	1/2 (4 bits)	***Version:*** Identifies the version of IP used to generate the datagram. For IPv4, this is of course the number 4. The purpose of this field is to ensure compatibility between devices that may be running different versions of IP. In general, a device running an older version of IP will reject datagrams created by newer implementations, under the assumption that the older version may not be able to interpret the newer datagram correctly.
IHL	1/2 (4 bits)	***Internet Header Length (IHL):*** Specifies the length of the IP header, in 32-bit words. This includes the length of any options fields and padding. The normal value of this field when no options are used is 5 (5 32-bit words = 5*4 = 20 bytes). Contrast to the longer *Total Length* field below.
TOS	1	***Type Of Service (TOS):*** A field designed to carry information to provide quality of service features, such as prioritized delivery, for IP datagrams. It was never widely used as originally defined, and its meaning has been subsequently redefined for use by a technique called *Differentiated Services (DS)*. See below for more information.
TL	2	***Total Length (TL):*** Specifies the total length of the IP datagram, in bytes. Since this field is 16 bits wide, the maximum length of an IP datagram is 65,535 bytes, though most are much smaller.
Identification	2	***Identification:*** This field contains a 16-bit value that is common to each of the fragments belonging to a particular message; for datagrams originally sent unfragmented it is still filled in, so it can be used if the datagram must be fragmented by a router during delivery. This field is used by the recipient to reassemble messages without accidentally mixing fragments from different messages. This is needed because fragments may arrive from multiple messages mixed together, since IP datagrams can be received out of order from any device. See the discussion of IP message fragmentation.
Flags	3/8 (3 bits)	***Flags:*** Three control flags, two of which are used to manage fragmentation (as described in the topic on fragmentation), and one that is reserved. **Subfield Name / Size (bytes) / Description** ***Reserved*** — 1/8 (1 bit) — ***Reserved:*** Not used. ***DF*** — 1/8 (1 bit) — ***Don't Fragment:*** When set to 1, specifies that the datagram should not be fragmented. Since the fragmentation process is generally "invisible" to higher layers, most protocols don't care about this and don't set this flag. It is, however, used for testing the maximum transmission unit (MTU) of a link. ***MF*** — 1/8 (1 bit) — ***More Fragments:*** When set to 0, indicates the last fragment in a message; when set to 1, indicates that more fragments are yet to come in the fragmented message. If no fragmentation is used for a message, then of course there is only one 'fragment' (the whole message), and this flag is 0. If fragmentation is used, all fragments but the last set this flag to 1 so the recipient knows when all fragments have been sent.
Fragment Offset	1 5/8 (13 bits)	***Fragment Offset:*** When fragmentation of a message occurs, this field specifies the offset, or position, in the overall message where the data in this fragment goes. It is specified in units of 8 bytes (64 bits). The first fragment has an offset of 0. Again, see the discussion of fragmentation for a description of how the field is used.
TTL	1	***Time To Live (TTL):*** Short version: Specifies how long the datagram is allowed to "live" on the network, in terms of router hops. Each router decrements the value of the TTL field (reduces it by one) prior to transmitting it. If the TTL field drops to zero, the datagram is assumed to have taken too long a route and is discarded. See below for the longer explanation of *TTL*.
Protocol	1	***Protocol:*** Identifies the higher-layer protocol (generally either a transport layer protocol or encapsulated network layer protocol) carried in the datagram. The values of this field were originally defined by the IETF 'Assigned Numbers' standard, RFC 1700, and are now maintained by the Internet Assigned Numbers Authority (IANA): **Value (Hexadecimal) / Value (Decimal) / Protocol** 00 / 0 / Reserved 01 / 1 / ICMP 02 / 2 / IGMP 03 / 3 / GGP 04 / 4 / IP-in-IP Encapsulation 06 / 6 / TCP 08 / 8 / EGP 11 / 17 / UDP 32 / 50 / Encapsulating Security Payload (ESP) Extension Header 33 / 51 / Authentication Header (AH) Extension Header Note that the last two entries are used when IPSec inserts additional headers into the datagram: the AH or ESP headers.
Header Checksum	2	***Header Checksum:*** A checksum computed over the header to provide basic protection against corruption in transmission. This is not the more complex CRC code typically used by data link layer technologies such as Ethernet; it's just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. The data is not checksummed, only the header. At each hop the device receiving the datagram does the same checksum calculation and on a mismatch, discards the datagram as damaged.
Source Address	4	***Source Address:*** The 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—it is always the device that originally sent the datagram.
Destination Address	4	***Destination Address:*** The 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.
Options	Variable	***Options:*** One or more of several types of options may be included after the standard headers in certain IP datagrams. I discuss them in the topic that follows this one.
Padding	Variable	***Padding:*** If one or more options are included, and the number of bits used for them is not a multiple of 32, enough zero bits are added to "pad out" the header to a multiple of 32 bits (4 bytes).
Data	Variable	***Data:*** The data to be transmitted in the datagram, either an entire higher-layer message or a fragment of one.



IEEE 802.3 Ethernet Frame Format

https://markroxor.github.io/blog/2020/05/09/networking/

https://www.gta.ufrj.br/ensino/eel878/sockets/inet_ntoaman.html

Library Document

Ethernet
https://sites.uclouvain.be/SystInfo/usr/include/net/ethernet.h.html

```
/* 10Mb/s ethernet header */
struct ether_header
{
  u_int8_t  ether_dhost[ETH_ALEN];      /* destination eth addr   */
  u_int8_t  ether_shost[ETH_ALEN];      /* source ether addr      */
  u_int16_t ether_type;                 /* packet type ID field   */
} __attribute__ ((__packed__));
```

From <https://sites.uclouvain.be/SystInfo/usr/include/net/ethernet.h.html>

ARP + Ethernet
https://sites.uclouvain.be/SystInfo/usr/include/netinet/if_ether.h.html

```
struct      ether_arp {
    struct      arphdr ea_hdr;          /* fixed-size header */
    u_int8_t arp_sha[ETH_ALEN];     /* sender hardware address */
    u_int8_t arp_spa[4];            /* sender protocol address */
    u_int8_t arp_tha[ETH_ALEN];     /* target hardware address */
    u_int8_t arp_tpa[4];            /* target protocol address */
};
```

From <https://sites.uclouvain.be/SystInfo/usr/include/netinet/if_ether.h.html>

ARP struct that's used in the ARP+Eth header
https://sites.uclouvain.be/SystInfo/usr/include/net/if_arp.h.html

```
struct arphdr
  {
    unsigned short int ar_hrd;          /* Format of hardware address.  */
    unsigned short int ar_pro;          /* Format of protocol address.  */
    unsigned char ar_hln;               /* Length of hardware address.  */
    unsigned char ar_pln;               /* Length of protocol address.  */
    unsigned short int ar_op;           /* ARP opcode (command).  */

                                        We don't need to use this at all.
#if 0
    /* Ethernet looks like this : This bit is variable sized
       however...  */
    unsigned char __ar_sha[ETH_ALEN];   /* Sender hardware address.  */
    unsigned char __ar_sip[4];          /* Sender IP address.  */
    unsigned char __ar_tha[ETH_ALEN];   /* Target hardware address.  */
    unsigned char __ar_tip[4];          /* Target IP address.  */
#endif
  };
```

From <https://sites.uclouvain.be/SystInfo/usr/include/net/if_arp.h.html>

ICMP
https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip_icmp.h.html

```
struct icmphdr
{
  u_int8_t type;                /* message type */
  u_int8_t code;                /* type sub-code */
  u_int16_t checksum;
  union
  {
    struct
    {
      u_int16_t     id;             We don't need to use this. This is already factored
      u_int16_t     sequence;       in while using this struct
    } echo;                     /* echo datagram */
    u_int32_t       gateway;    /* gateway address */
    struct
    {
      u_int16_t     __unused;
      u_int16_t     mtu;
    } frag;                     /* path mtu discovery */
  } un;
};
```

From <https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip_icmp.h.html>

IP
https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip.h.html

```
struct iphdr                    We don't have to change this. The system knows what to do.
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error     "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
  };
```

From <https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip.h.html>

```
char packet[1500]
uint16_t * p = (uint16_t*)&packet[12];
unit32_t sum = 0;
for (i=0; i<?,i++){
sum += p[i]
}
sum = sum&0xffff + sum >> 16; //Folding
the chechsum
sum = ~sum //(invert)
```

read the table into the file and do what you want
router reads the file output and will then decide how to route packets