QoE for Q68 Ethernet driver notes

Version 0.39

(QoE - QLNET over Ethernet)

SuperBASIC extensions

----

ini_cmd	SuperBASIC Procedure definition list
eth_init	SuperBASIC Procedure ETH_INIT to (Re)Initialize the CP2200 Ethernet controller Assuming that the Ethernet device driver was installed, an attempt is made to initialize the CP2200 again. Accepts an optional parameter for the duplex type. 0 (default) = Auto-negotiation, 1 = Full duplex, 2 = Half duplex
feth_init	SuperBASIC Function FETH_INIT of the above Procedure. Accepts an optional parameter for the duplex type. 0 (default) = Auto-negotiation, 1 = Full duplex, 2 = Half duplex Returns, 0 for OK 'Not Complete' for initialization timed out. 'Transmission error' for Auto-negotiation failed. 'Not Found' for uninitialized driver. 'Bad Parameter'
eth_macs	SuperBASIC Function ETH_MAC$ to return the MAC address of the CP2200 as a dash seperated string. 'Not Found' for uninitialized driver. 'Bad Parameter'
arp_add	SuperBASIC Procedure ARP_ADD to add, or update an arp table entry. MACaddress in the format "aa-bb-cc-dd-ee-ff"
arp_remove	SuperBASIC Procedure ARP_REMOVE to removes one or all ARP table entries from the linked list. Without parameters, removes all entries
arp_list	SuperBASIC Procedure ART_LIST to lists all ARP table entries to #ch or default #1. In the form, IP Address MAC Address
eth_dns	SuperBASIC Procedure ETH_DNS to set the Q68's Domain Name server IP address in the device definition block
eth_setip	SuperBASIC Procedure ETH_SETIP to set the Q68's IP address in the device definition block
eth_subnet	SuperBASIC Procedure ETH_SUBNET to set the Q68's subnet mask in the device definition block
eth_gateway	SuperBASIC Procedure ETH_GATEWAY to set the Q68's default gateway IP address in the device definition block
setip	Routine used by ETH_SETIP, ETH_SUBNET, and ETH_GATEWAY to do the setting of the IP addresses
eth_fdns	SuperBASIC Function ETH_DNS$ to return the set IP address as a dot separated string

eth_fgetip	SuperBASIC Function ETH_GETIP$ to return the set IP address as a dot separated string
eth_fsubnet	SuperBASIC Function ETH_SUBNET$ to return the set IP address as a dot separated string
eth_fgateway	SuperBASIC Function ETH_GATEWAY$ to return the set IP address as a dot separated string
getip	Routine used by ETH_GETIP$, ETH_SUBNET$, and ETH_GATEWAY$ to return the actual strings
eth_netname	SuperBASIC Procedure ETH_NETNAME to set the Q68's network name
eth_fnetname	SuperBASIC Function ETH_NETNAME$ to return the Q68's network name as a string
eth_errno	SuperBASIC Function ETH_ERRNO to return the last driver specific error. This is not the same as a QDOS error, But an error code to indicate the last problem the driver encountered. ETH_ERRNO will clear the error code after it is read.
eth_ping	SuperBASIC Procedure ETH_PING. Sends 4 Pings to the supplied IP Address, Sending the results to the specified channel, or #1.
eth_ipconfig	SuperBASIC Procedure ETH_IPCONFIG. Displays information about the Ethernet controller and driver settings.
get1int	Fetch one Procedure/Function parameter integer and place it on the maths stack
get1lin	Fetch one Procedure/Function parameter long integer and place it on the maths stack
get1str	Fetch one Procedure/Function parameter string and place it on the maths stack
dns_add	SuperBASIC procedure DNS_ADD to add an entry to the DNS cache.
dns_list	SuperBASIC procedure DNS_LIST to list the entries in the DNS cache.
dns_flush	SuperBASIC procedure DNS_FLUSH to remove all entries from the DNS cache. Except for the first one, 'local host'.

Subroutine list
----

add_arp_rec — Adds, or updates a record in the ARP table
Entry
D5.L top four bytes of the MAC address
D6.W bottom two bytes of the MAC address
D7.L IP address
A3 base of device driver definition block
Exit
D0 0, or 'out of memory'

add_dns_rec — Adds a record in the DNS cache table
Entry
A1 pointer to start of a domain name
D1.W length of the domain name
D6.L time to live, zero means forever
D7.L IP address
A3 base of device driver definition block
Exit
D0 0, or 'out of memory'

allocateport — Allocate a free port from the managed table of ports. OPEN_IN and Binding requires a system selected port. This port will be selected from a pool of 256 ports between $D200 to $D300
The allocation is managed from a 32 byte port allocation map, where each bit identifies a port as being free (0) or in use (1)
There is a rotating port number pointer that is incremented each time a port is allocated. So if a port is used, then released, it will not be used again immediately.
Entry
D4 upper word is port supplied to the OPEN routine
A3 base of driver definition block
Exit
D0 error return. Possible errors -
Buffer full, no ports available
D4 lower word, selected port

arp_ip_request
Request a MAC address from a remote computer with the supplied IP address
Entry
D7 IP address of required computer
A1 base of buffer
A3 base of driver definition block
Exit
D0 0 or QDOS error code
buffer full — transmit buffer is not empty, after waiting 1.5 seconds
transmission error — if the last packet was not transmitted successfully
A1 updated pointer to buffer

cache_name Tries to match a domain name to a DNS cache record. Returning an IP address
Entry
D1 length of supplied domain name
A1 pointer to supplied domain name
A3 base of device driver definition block
Exit
A4 IP address if successful, otherwise undefined
D7 date for Time To Live if successful, otherwise undefined
D0 0, or 'Not Found'

check_IP_address
Checks the supplied IP address, If it's not on the local LAN, and a Default gateway has been set, then use the Default gateway IP address
Entry
D0 IP address of required computer
A3 base of driver definition block
Exit
D0 preserved, or the Default gateway IP address

check_mac_address
Scan the ARP table to see if we know the MAC address for the IP address in D0
Entry
D0 IP address of required computer
A3 base of driver definition block
Exit
D0 preserved
zero flag not set if successful, and
D5.L upper part of MAC address
D6.W lower word of MAC address

check_open_valid
Checks to see if the required OPEN command can proceed
Table for determining which OPEN commands are valid. The table takes the form of four bytes for disconnected from network and four bytes for connected to network.
Each four bytes are for OPEN, OPEN_IN, OPEN_NEW, spare (OPEN_OVER)
Return values are 0, invalid parameter, transmission error
or format failed for an undefined open type .
Entry
D7 lower word, is open type
A4 supplied IP address
Exit
D0 0, or an error code

Table for determining which OPEN commands are valid. The table takes the form of four bytes for disconnected from network and four bytes for connected to network.
Each four bytes are for OPEN, OPEN_IN, OPEN_NEW, spare (OPEN_OVER)
Return values are 0, invalid parameter, transmission error
or format failed for an undefined open type

checksetport — If supplied port number is in the range $D200 to $D300, and if already allocated. If so returns an 'in use' error. Otherwise flags port as in use
Entry
D3.W port to allocate
A3 base of driver definition block
Exit
D0 0, or 'In Use' error

cp2200_init — Initialize the CP2200 Ethernet controller.
Entry
D5 Required duplex mode. 0 = Auto-negotiation, 1 = Full duplex, 2 = Half duplex
A3 Assumed start of driver definition
Exit
D0 error return. Possible errors -
Not complete = Self Initialization timed out
Transmission error = Auto-negotiation failed

cp2200_mac — Initializing the MAC of the CP2200
Entry
A1 points at the base address of CP2200 registers
A3 points at the definition of the device driver

cp2200_phlay
Initialize the Physical Layer of the CP2200
Entry
A1 points at the base address of the CP2200
Exit
D0 error return. Possible errors -
Transmission error, Self Initialization timed out

cp2200_WritePacket
Write a packet to the Ethernet controller
Interrupts are disabled during this routine to prevent it being recalled in the middle of writing a packet
Entry
D2.W number of bytes to send
A1 base of buffer
A2 base of CP2200 Ethernet controller
A3 base of device driver definition block
Exit
D0 0 or QDOS error code
buffer full — transmit buffer is not empty, after waiting 1.5 sec
transmission error — if the last packet was not transmitted successfully
D2 preserved
A1 updated pointer to buffer

cp_aneg — Do auto-negotiation of the CP2200 +++++ unfinished ++++
Entry
A1 points at the base address of CP2200 registers
Exit
D0 error return. Possible errors -
Transmission error, Self Initialization timed out

ddlink — Get the assumed start of Q68Net Driver Definition Block in A2, or set the zero flag for not found.

deallocateport — De-allocate a port from the managed table of ports, If in the a pool of 256 ports between $D200 to $D300
Entry
D0.W port to de-allocate
A3 base of driver definition block
Exit
none

do_dns — Send a DNS request to the DNS server
Entry
D1 length of the IP address/Domain name
A0 start of device name
A1 pointer to start of domain name string
A3 base of driver definition block
Exit
A4 IP address
D0 0, or an error code
D7 time to live
all other registers may be trashed

dochn — Convert the S*BASIC channel number to a Channel ID
Entry
D0 S*BASIC Channel number
Exit
D0 0 or Not found
A0 Channel ID

fetchpacket — Look to see if there is a packet waiting in the channels queue to be accessed. If so, check to see if it has yet to read, or has already been read. Linking in the next if needed. Otherwise return 'not complete'. Returns a 'transmission error' on a wrong MAC address.
Must be in supervisor mode, and A6 pointing to the system variables.
Entry
A0 start of channel definition
A3 base of driver definition block
A6 base of system variables
Exit
D0 0, or an error code

get_cdb — Convert a channel ID in A0 to a pointer to the base of the channel definition block.
Entry
A0 Channel ID
Exit
A0 Points to base of channel definition block
D0 Zero, or Channel not open error

get_lang Get the system language, and return as English, French, German, Italian, or Spanish in D0
Exit
D0 001 for English(US)
044 for English(UK)
049 for German
033 for French
034 for Spanish
039 for Italian

init Create a device driver definition block, Initialize the CP2200.
If successful link the block into the system

int_serve Interrupt handler for reading data packets in supervisor mode on entry
Entry
D3 number of 50/60Hz interrupts
A3 base of driver definition block
A6 base of system variables
A7 supervisor stack (64 bytes free)
Exit
everything preserved

is_assign Try to identify the type of packet received, and assign it to a channel, or throw it away
Entry
A0 points to the start of the buffer
A3 base of driver definition block

lang_search Search the language table. Returns a pointer to the start of the required language line in the language table. If language code is not found, it defaults to English
Entry
D0 language code
Exit
A4 points at start of language table entry

local_host_ent Creates a hostent for the 'localhost' domain at the address pointed to by A2
Entry
A2 pointer to buffer for hostent
Exit
none

localWritePacket
Try to write the packet to the local host directing it to the correct open IP channel
Entry
D2.W number of bytes to send
A0 points at the channel definition block
A1 base of buffer
A3 base of driver definition block
Exit
D0 0 or QDOS error code
transmission error if there was a memory problem
D1.W number of bytes sent
D2 preserved
A1 updated pointer to buffer

make_hostent Build a hostent structure from the DNS reply
Entry
A0 base of dummy channel definition block
A2 pointer to the hostent buffer
A3 base of driver definition block
A4 pointer to the start of the DNS response reply
A5 pointer to the start of the DNS queries
D4 is the number of answers in the DNS reply
D7 is the supplied IP address in a GETHOSTBYADDRESS
Exit
none
All registers except A0,A2,A3,A6 are libel to be changed

nd_close Device driver channel close routine

nd_io Device driver I/O routines

nd_open Device driver channel open routines for UDP, TCP and SCK channels
(TCP not fully implemented as yet)

nd_getmac Check to see if the ARP table has been updated with the required MAC address.
If ARP table has not been updated, another ARP request is sent at the half time point, and at the timeout.
Note this is not a subroutine, may not return to caller
Entry
D0 operation
A3 base of driver definition block
Exit
D0 preserved, or an error code

nd_gen_trans_csum
Generate a Transport layer checksum for the block of data pointed to by A1, Length D2.W
Also creates the transport layer header in the channel definition block ready for sending
Entry
A0 base of channel definition block
A1 points at the start of the data block
A3 base of driver definition block
D2.W number of bytes in data block
Exit
D0.W The required checksum
Zero flag set on an error

ndo_getbyte
ndo_getword Read a device name parameter, Converts an ASCII string into a number in D7
Entry
D5 number of digits to read
A4 pointer to start of ASCII number
Exit
D7 ndo_byte - byte value
ndo_word - unsigned word
zero flag not set on error

nstr2long — Check the IP address of a null terminated string, returning it as a long word in D7
Uses the str2long routine
Entry
D0 length of string
A2 Pointer to end of string
Exit
D0 0, or QDOS error code
Bad Parameter
D7 IP address in network order

opn_decode_address
Decode IP address/Domain name parameter in OPEN command
Entry
D1 length of the IP address/Domain name
A0 start of device name
A1 pointer to start of parameter string
A3 base of driver definition block
Exit
A4 IP address
D0 0, or an error code

opn_decode_port
Decode port/service parameter in OPEN command
Entry
D0 offset from A1 to the colon ':' in the parameter string
D3 open type
D4 length of the whole parameter string
D6 open type/device type
A0 start of device name
A1 pointer to start of parameter string
A3 base of driver definition block
Exit
D4 port number
D0 0, or an error code

opn_dmy_cdb — Create a dummy channel definition block for a DNS request. Then links it into the linked list of open IP channels. Expects to be in supervisor mode
Note, trashes A0
Entry
A3 base of driver definition block
Exit
A0 base of the created channel definition block
D0 0, or an error code

clo_dmy_cdb — Close the dummy channel definition block for a DNS request
Entry
A0 base of dummy channel definition block
A3 base of driver definition block
Exit
none

send_mac_request

Send a request on the network for a MAC address for the IP address in D0
Entry
D0 IP address of required computer
A3 base of driver definition block
Exit
D0 0 or QDOS error code
buffer full transmit buffer is not empty, after waiting 1.5 seconds
transmission error if the last packet was not transmitted successfully

str2long

Converts an IP address string on the Maths stack and return it as a long word in D7
Entry
A1 Pointer to Maths stack
Exit
D0 0, or QDOS error code
Bad Parameter
D7 IP address in network order

str2mac

Check the MAC address QDOS string on the Maths stack and return it as a long word in D5 & a word in D6
The string should be in the format "aa-bb-cc-dd-ee-ff"
Entry
A1 Pointer to Maths stack
Exit
D0 0, or Bad Parameter
D5.L Top four bytes of MAC address
D6.W Bottom two bytes of MAC address

tcp_accept

Deal with an IP_ACCEPT. Accept a connection for a socket specified by the channel ID supplied in D3
Entry
D3 channel ID of LISTENing channel
D6 upper word is open type
A0 start of device name - must be a SCK_
A3 base of driver definition block
A5 base of driver definition block
A6 base of system variables
Exit
D0 0, or an error code
A0 base of channel definition block

tcp_close

Do a TCP close connection sequence
Entry
A0 base of channel definition block
A3 base of driver definition block
A6 base of system variables
Exit none

tcp_connect	Attempt to make a Three Way Handshake connection to a TCP server that is Listening for connection requests. Returns 'Transmission error' if a connection cannot be made This routine may be called from either the OPEN routine, or IP_CONNECT Entry A0 base of channel definition block A3 base of driver definition block A6 base of system variables Exit D0 0, or an error code ????
uh_alloc	Allocate an area in a user heap Entry D1 required space on the user heap A0 pointer to pointer to pointer to free space in user heap Exit D1 length allocated A0 base of user heap area allocated D0 0, or 'Out of memory'
uh_extend	Extend an existing User Heap Entry D1 required space for the extension A1 pointer to pointer to pointer to free space in user heap Exit A0 undefined D0 0, 'Out of memory' or 'job does not exist'
uh_setup	Assign and set up a new User Heap Entry D1 required space for the user heap A1 pointer to pointer to pointer to free space in user heap A2 pointer to address to store base of allocated area Exit A0 undefined (A1) pointer to pointer to free space in user heap (A2) base of common heap allocated D0 0, 'Out of memory' or 'job does not exist'
uh_rechp	Release an allocated area in a user heap Entry A0 base of space to free A1 pointer to pointer to free space in user heap Exit A0 undefined A1 undefined
valIPV4hdr	Validate an IPV4 network header by its checksum Entry A0 base of packets buffer A3 base of driver definition block Exit D0 error return

valICMPhdr Validate an ICMP transport header by its checksum
Entry
A0 base of packets buffer
A3 base of driver definition block
Exit
D0 error return

valTCPhdr Vaildate a TCP transport header by its checksum
Entry
A0 base of packets buffer
A3 base of driver definition block
Exit
D0 error return

valUDPhdr Vaildate an UDP transport header by its checksum
Entry
A0 base of packets buffer
A3 base of driver definition block
Exit
D0 error return

writepacket Adds the required headers to the payload in the transmit buffer, and writes it to the CP2200 for transmission
Entry
A0 base of channel definition block
A3 base of driver definition block
Exit
D0 0 or QDOS error code

There are also some other entry points into wtitepacket in addition to A0 & A3 above

translayer_udp, translayer_tcp, translayer_icmp
On entry D2.W is the number of bytes in the payload
A1 is a pointer to the start of the payload

(D1-D4 may be smashed)

netlayer_ip D2 is the number of bytes in the payload
A1

phylayer D2
A1

DHCP client routines

DHCP_client Attempt to obtain an IP address from a DHCP server. And handle lease renewals
This is the main code of the DHCP client. Sets the base address of it's data block, and a flag in the main drivers definition block.

Entry
A6 points at the base of the job
(A6,A4) points at the bottom of the data space
(A6,A5) points at the top of the data area
A7 points at two words of zero on the stack

dhcp_open_udp
Open a UDP channel

dhcp_get_reply
Attempt to obtain DHCP reply, checks for operation=reply, and the magic cookie throws away anything else call in supervisor mode
Entry
A0 pointer to channel definition block
A3 pointer to driver linkage block
A6 pointer to system variables
Exit
A4 pointer to start of payload
D0 0, or 'not complete'

dhcp_parse_opts
Parse the options of a DHCP reply
Entry
A0 pointer to channel definition block
A3 pointer to driver linkage block
A6 pointer to data area
Exit
A4 pointer to start of payload

dhcp_renewal Apply for renewal , or rebinding of IP address lease
Entry
A3 pointer to driver linkage block
A6 pointer to data area

dhc_find_client Checks to see if the DHCP client is running. Returns not zero if client is running.
Exit
D4 DHCP client job ID (if job is running)

dhcp_release Sends a DHCPRELEASE to the DHCP server. The DHCP client must be running
Entry
A6 start of BASIC
Exit
A6 preserved
D0 zero, or an error code
Note all other registers libel to be trashed

dhcp_confirm DHCP_CONFIRM If the DHCP client is running, the user is notified in #0, and asked to confirm that the DHCP client should be shut down. The DHCP client job is shut down if the user elects to continue.
Entry
A6 start of BASIC
Exit
A6 preserved
D0 zero, or an error code
Note all other registers libel to be trashed

dhcp_start Start the DHCP client job and attempt to obtain an IP address form a DHCP server Called from a S*BASIC Proc/Fun.
Entry
A6 Points at start of BASIC
Exit
D0 zero, or error code 'not complete' indicates DHCP was unsuccessful
or BREAK was pressed

Supported System Trap calls

Trap #2

D0	Name	Notes
$01	IO_OPEN	D3=0-2
$01	IP_ACCEPT	D3=LISTENing channel ID
$02	IO_CLOSE	

Trap #3

D0	Name	Notes
$00	IO_PEND	
$01	IO_FBYTE	
$02	IO_FLINE	
$03	IO_FSTRG	
$05	IO_SBYTE	
$07	IO_SSTRG	D2 is word sized, So should limit data size to 32K
$48	FS_LOAD	
$49	FS_SAVE	
$50	IP_LISTEN	
$51	IP_SEND	data size limited to 64K
$52	IP_SENDTO	data size limited to 64K
$53	IP_RECV	
$54	IP_RECVFM	
$58	IP_BIND	
$59	IP_CONNECT	
$5B	IP_GETHOSTNAME	
$5C	IP_GETSOCKNAME	
$5D	IP_GETPEERNAME	
$5E	IP_GETHOSTBYNAME	
$5F	IP_GETHOSTBYADDR	
$64	IP_GETSERVBYNAME	
$65	IP_GETSERVBYPORT	
$6E	IP_GETPROTOBYNAME	
$6F	IP_GETPROTOBYNUMBER	
$72	IP_INET_ATON	
$73	IP_INET_ADDR	
$74	IP_INET_NETWORK	
$75	IP_INET_NTOA	
$76	IP_INET_MAKEADDR	
$77	IP_INET_LNAOF	
$78	IP_INET_NETOF	
$7C	IP_ERRNO	

Device Driver Definition Block

$00	ndd_eilk	link to next external interrupt
$04	ndd_eiro	address of external interrupt routine
$08	ndd_5ilk	link to next 50/60Hz interrupt
$0c	ndd_5iro	address of 50/60Hz interrupt routine
$10	ndd_silk	link to next scheduler interrupt
$14	ndd_siro	address of scheduler interrupt routine
$18	ndd_ddlk	link next device
$1c	ndd_iolk	link to I/O routine
$20	ndd_oplk	link to open routine
$24	ndd_cllk	link to close routine
$28	ndd_pmptr	Port map pointer, increments after each allocation
$2A	ndd_last_err	Last IP error, cleared after reading
$2C	ndd_chlist	Link to list of open IP channels
$30	iod_cnam	Pointer to routine to make the channel name (QPAC2)
$34	ndd_ipid	Network (IPV4) layer identification. Increments for every packet sent
$36	ndd_q68e	Q68 Ethernet identity string
$3a	ndd_base	base address of CP2200 direct registers
$3e	ndd_etir	Q68 Ethernet interrupt register
$42	ndd_mac	6 bytes of the MAC address of the CP2200
$48	ndd_ip	IP address of this computer
$4C	ndd_subnetmask	IP subnet mask
$50	ndd_gateway	default gateway IP address
$54	ndd_netname	computers network name. word + up to 26 characters
$70	ndd_arp	start of ARP table of MAC to IP addresses
$74	ndd_queue_base	base of packet queue user heap
$78	ndd_queue_p2p	pointer to the pointer to the user heap free space (packet queue)
$7C	ndd_arp_base	base of ARP table user heap
$80	ndd_arp_p2p	pointer to the pointer to the user heap free space (ARP table)
$84	ndd_txpackets	number of packets sent by the CP2200
$88	ndd_txbytes	number of bytes sent by the CP2200. Includes header bytes
$8C	ndd_rxpackets	number of packets received by the CP2200
$90	ndd_rxbytes	number of bytes received by the CP2200. Includes header bytes
$94	ndd_DHCPclient	base of DHCP data area, 0 if no client running
$98	ndd_DHCPstatus	DHCP client status, 4=bound, negative QDOS code for an error
$99		1 spare byte
$9A	ndd_dns	DNS server IP address
$9E		2 spare bytes
$A0	ndd_portmap	32 byte port allocation map
$D0	ndd_buffer	1514 byte buffer for interrupt routine packet handling, or other buffering
	ndd_endi	ndd_buffer+1514 End of definition block
	ndd.leni	ndd_endi-ndd_eilk Length of definition block

ARP table linkage block

$08	arp_next	pointer to next link
$0C	arp_ip	IP address
$10	arp_mac	6 byte MAC address
$16	arp_free	2 spare bytes
$18	arp_end	end of entry

Channel Definition Block

$18		2 spare bytes
$1A	nd_ARPtmr	ARP request timeout
$1E	nd_MACbad	when set, The destination MAC address is bad, $7F after 20 sec
$1F		1 spare byte
$20	nd_destmac	destination MAC address
$26	nd_desip	destination IP address
$2A	nd_destport	destination IP port
$2C	nd_myip	my IP address - used for BINDing a channel
$30	nd_myport	my IP port
$32	nd_devicetype	device type -1=SCK, 0=UDP, 1=TCP
$33	nd_protocol	device protocol - eg 17 for UDP
$34	nd_acces	access mode (D3 on open call)
$35	nd_sock_state	socket status
$36	nd_flagsOffset	IPV4 flags and offsets
$38	nd_sequence	TCP sequence number
$3C	nd_acksequence	TCP acknowledge sequence
$40	nd_offsetResFlags	UDP flags and offsets : TCP flags and things - needs sorting
$42	nd_windows	TCP windows - needs sorting
$44	nd_urgent	TCP urgent - needs sorting
$46	nd_tcp_opt_len	length of option part of TCP header
$48	tcb_SND.UNA	oldest unacknowledged sequence number
$4C	tcb_SND.NXT	next sequence number to be sent
$50	tcb_SND.WIN	send window size
$52	tcb_RCV.NXT	next sequence number to be received
$56	tcb_RCV.WND	receive window size
$58	nd_SEG.ACK	next sequence number expected by the receiving host
$5C	nd_SEG.SEQ	first sequence number of a segment
$60	nd_SEG.LEN	the number of octets of data in the segment
$62	nd_SEG.LAST	last sequence number of a segment SEG.SEQ+SEG.LEN-1
$66	tcb_mss	host maximum segment size
$68	tcb_ws	window scale
$69	tcb_TCP_STACK	TCP SACK permitted true/false
$6A	nd_TCP_packcount	the number of packets to send before waiting for an ACK
$6B	nd_listenQ	length of IP_LISTEN backlog queue (LISTEN channel only)
		8 spare bytes
$74	nd_ddbase	assumed start of device definition block
$78	nd_nextch	link to next open IP channel
$7C	nd_packqueue	link to linked list of received queued data packets
$80	nd_txptr	transmit buffer pointer running pointer
$82	nd_txendptr	transmit buffer end pointer
$84	nd_rxptr	receive buffer pointer running pointer
$86	nd_rxendptr	receive buffer end pointer
$88	nd_rxdatabase	address of start of current rx packet's payload
$8C	nd_txbufferbase	transmit buffer base (transmit header area)
	nd_txdatabase	nd_txbufferbase + 78 bytes Start of transmit buffer data area
	nd_int_reply	pointer to end of transmit buffer for interrupt routine
	nd_chend	nd_txdatabase + 1600 bytes End of channel definition block

The end of the channel definition block, from nd_txbufferbase onwards is used as the transmit buffer for the channel. The data packet is composed in this area.

There are two pointers, nd_txptr and nd_txendptr, used to track the current data position and the end of the available buffer space.

```
  ----------------------------------------   <-----  nd_txbufferbase
|          TX header area              |           This area is reserved for the Ethernet headers
|                                      |
  -----------------------------------------   <----- nd_txdatabase
|                                      |           This is the actual payload data area
|          TX buffer area              |
|                                      |
  ---------------------------------------
|                                      |           This is a buffer area for the interrupt routines
|          Interrupt TX                |           Used to send ACK's and FIN's
|          buffer area                 |
  ----------------------------------------  <-----   nd_int_reply
|                                      |
|          10 spare bytes              |           10 bytes of wriggle room
|                                      |
  ---------------------------------------- <-----    nd_chend          End of channel definition block
```

Dummy channel definition block

A dummy channel definition block (in the user heap) is used by LISTEN to handle the 3 way handshake of a connection request. This dummy channel definition block is the same as a normal channel definition block, only shorter. It has a small transmit buffer area, as it only has to send a SYN,ACK.

The dummy channel definition block is is linked into a list of connection requests maintained by the LISTENing channel. And also the linked list of open IP channels.

When IP_ACCEPT, accepts the established connection, then the dummy channel definition block is unlinked from the two lists, and copied into the real channel definition block, and the dummy one is then deleted.

Some channel definition block entries are re-tasked for the dummy channel definition block

$00	dmy_base	8 byte user heap header (don't touch)
$08	dmy_next	link to next dummy channel definition block

dmy_owner	nd_ARPtmr	long	address of owner listing channel

PING

Some channel definition block entries are re-tasked for ICMP, Ping

Normal	Re-assignment		
nd_ARPtmr	ping_timeout	long	time to wait for ping reply
nd_sequence	ping_ident	word	ping identifier
nd_sequence+2	ping_sequence	word	ping sequence number
nd_acksequence	ping_startTime	long	start time for loop travel time
nd_urgent	ping_type	byte	ICMP transport layer type
nd_window	ping_ttl	word	Time To Live
nd_txdatabase+$40	ping_myIPtext	20 bytes	my IP address as a string
nd_txdatabase+$54	ping_gatewayIPtext	20 bytes	default gateway IP address as a string
nd_txdatabase+$68	ping_targetIPtext	20 bytes	target IP address as a string
nd_txdatabase+$7C	ping_TTLtext	8 bytes	time to live as a string
nd_txdatabase+$84	ping_triptimes	4 words	4 trip times in ms
nd_txdatabase+$8C	ping_received	word	number of received replies

Receive data buffering

The CP2200 Ethernet controller can only buffer up to 4K bytes of received data, or up to 8 data packets. Whichever come first.

There is an interrupt routine that constantly monitors the Ethernet controller for data packets being received.

The basic operation of the routine is that, If the reception of a data packet is detected, Then a buffer is allocated in memory, and the data packet is copied into it.

The content of the packet is then examined, and a scan of the opened IP channels is made to see if the packet is intended for one the open channels. If a match is found, then the data packets buffer is linked onto the end of a queue of data packets intended for that channel.

If no match can be found, or the routine does not know what to do with the received packet, Then the buffer is deleted, throwing the data packet away.

Receive buffer format

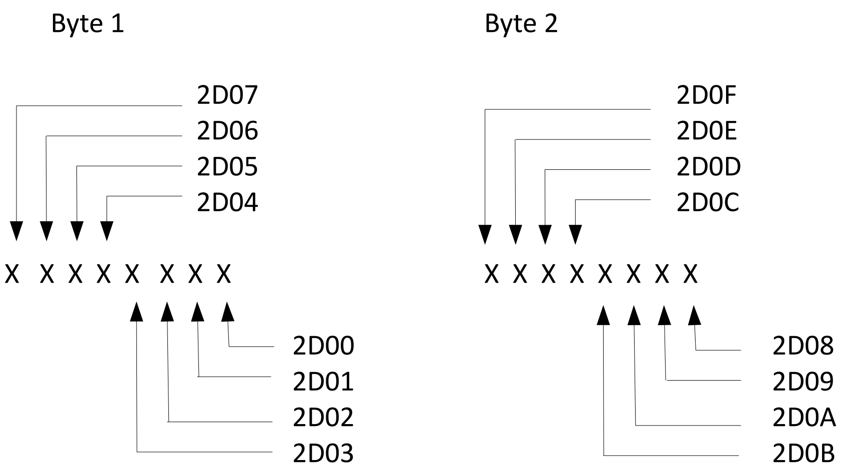
$00	rxp_base	heap allocation header, don't use (8 bytes)
$08	rxp_next	link to the next receive data buffer
$0C	rxp_datastart	offset from start of buffer to start of payload
$0E	rxp_datalen	length of payload data
$10	rxp_lifetime	number of read attempts left. If 'rxp_ok2read' is not true, then this is the number of times the channels I/O (timeout) will try to read this packet before it gives up and deletes the incomplete fragmented packet
$12	rxp_ok2read	true if the packet is ready to be read. False if the packet is an incomplete fragmented packet
$13	rxp_sockstate	status of packet for server connection
$14	rxp_start	start of the data packet

Managed port area

Sometimes the driver has to choose a port to receive data on. Rather than try to manage all 65536 ports, and to try to keep the load on the system resources down, the driver will only pick ports from a managed area. It uses a pool of 256 ports, from $D200 to $D300. There is a 32 byte port map in the driver definition block 'ndd_portmap', where each bit represents one of the 256 ports.
This does not mean that you cannot uses ports outside of this area. It just means that the driver will not accidentality try to reuse a port in this area, that is already in use.

Each time a port is allocated from the managed port area, a pointer 'ndd_pmptr' is incremented to prevent a port being used twice in a row.

The following diagram shows the relationship between the port map and the addresses
A bit set to '1' indicates that the port has been allocated.



Background packet reading routine   (INT_SERV)

The interrupt driver background reading routines are responsible for reading data packets from the CP2200 Ethernet controller. Analyse them, act on, or allocate the data packet to a channel.

The background packet reading has to operate autonomously with no direct feedback to the user of any problems. The only feedback is via the **ETH_ERRNO** S*BASIC function. Whenever the background packet reading routines, don't know what to do with a data packet, it just quietly throws it away.

The background packet reading is handled by both the 50/60Hz interrupt, and a hardware interrupt.

When an interrupt occurs, A test is made to see if a data packet is available in the CP2200 Ethernet controller. If there is no data packet available, then the interrupt ends (is_leave).

If a data packet is available (is_dopacket). A buffer is allocated in the 256K user heap, and the data packet is copied from the CP2200 Ethernet controller to the user heap.

The type of the data packet is now tested in (is_assign). If it is an ARP request it is dealt with in (is_doarp). If it's an IP packet, it is dealt with in (is_doip). Otherwise the data packet is just quietly thrown away (is_delpacket).

ARP requests   (IS_DOARP)

The ARP packet is examined to see if it's a reply to a request we made, A request for a MAC address, or a general announcement that a computer has joined the network.

The appropriate action is preformed. Either store the supplied MAC address in the ARP table user heap, or send an ARP packet with the Q68's MAC address to the requester.

The ARP packet is then deleted.

IP packets   (IS_DOIP)

The packets protocol is checked to see if it is either, UDP (is_doudp), ICMP (is_doicmp), or TCP (is_dotcp).

UDP packets   (IS_DOUDP)

The packet is checked to see if it fragmented. If this is the first fragment of a group, then a new data packet is created in the user heap that is large enough to hold all the fragments of the group. And as further fragments arrive, they are inserted into this new packet. So you end up with one complete packet for the whole group of fragments.

A search is made of all the open IP channels looking for match of protocols and ports. If a channel is found, then the packet is added to the end of a linked list of packets waiting to be read.

TCP packets   (IS_DOTCP)

A search is made of all the open IP channels looking for match of protocols, IP addresses and ports. If a channel is found, the TCP flags are analysed to decide what kind of TCP packet it is (is_tcp_decode).

Depending on these flags, and the status of the connection. Different actions take place. The actions may result in the packet being added to the end of the linked list of packets waiting to be read by a channel. Or data packets being created and sent back to the sender.

ICMP packets (IS_DOICMP)

If the packet is a Ping request (is_ping_req), Then the request is patched into a reply and sent back to the sender.

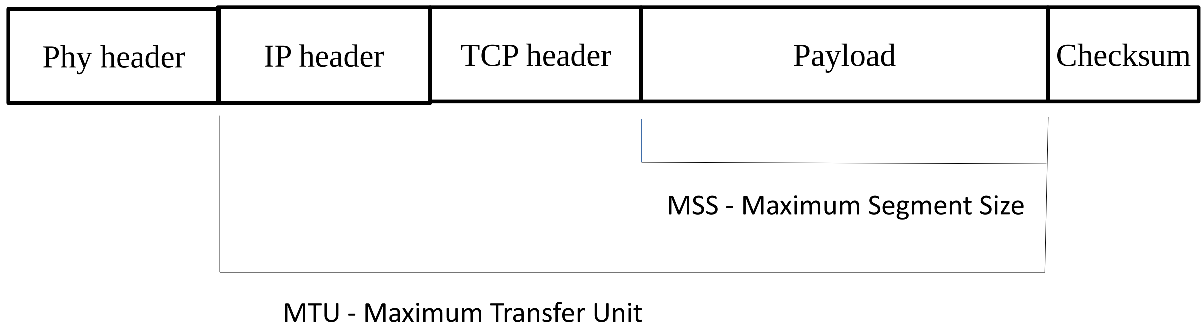
If the packet is a Ping reply (is_ping_reply) from the **ETH_PING** command, Then the requesting SMSQ/E channel is found. The round trip time is calculated in 25nS intervals, and saved in case there are any delays before getting back to S*BASIC. The packet is then linked to the SMSQ/E channel to be dealt with by the **ETH_PING** S*BASIC command.

Any other ICMP packets are discarded.

TCP Support

There is a pseudo TCP implementation in this driver. With very little TCP error handling. So all data packets must arrive complete and in the right order.

TCP name meanings



TCP Protocol Operation   (Parts taken from Wikipedia)

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (*connection establishment*) before entering the *data transfer* phase. After data transmission is completed, the *connection termination* closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a resource that represents the local end-point for communications, the *Internet socket*. During the lifetime of a TCP connection, the local end-point undergoes a series of state changes:

LISTEN
(server) represents waiting for a connection request from any remote TCP and port.
SYN-SENT
(client) represents waiting for a matching connection request after having sent a connection request.
SYN-RECEIVED
(server) represents waiting for a confirming connection request acknowledgement after having both received and sent a connection request.
ESTABLISHED
(both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.
FIN-WAIT-1
(both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgement of the connection termination request previously sent.
FIN-WAIT-2
(both server and client) represents waiting for a connection termination request from the remote TCP.
CLOSE-WAIT
(both server and client) represents waiting for a connection termination request from the local user.

CLOSING
(both server and client) represents waiting for a connection termination request acknowledgement from the remote TCP.
LAST-ACK
(both server and client) represents waiting for an acknowledgement of the connection termination request previously sent to the remote TCP (which includes an acknowledgement of its connection termination request).
TIME-WAIT
(either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgement of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two maximum segment lifetime (MSL).]
CLOSED
(both server and client) represents no connection state at all.

Keys used by the driver for socket status

0	sts_none	
1	sts_listen	LISTEN
2	sts_syn_sent	SYN-SENT
3	sts_syn_recv	SYN-RECEIVED
4	sts_estab	ESTABLISHED(c & s) connection is established
5	sts_fin_wait1	FIN-WAIT-1
6	sts_fin_wait2	FIN-WAIT-2
7	sts_close_wait	CLOSE-WAIT
8	sts_closing	CLOSING
9	sts_last_ack	LAST-ACK
10	sts_time_wait	TIME-WAIT
11	sts_closed	CLOSED

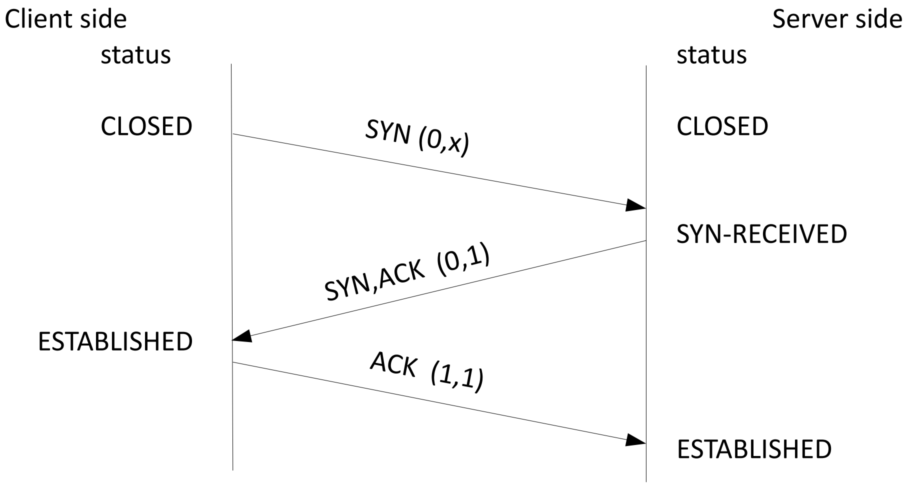
Connection establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. **SYN**: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
2. **SYN-ACK**: In response, the server replies with a SYN-ACK. The acknowledgement number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B.
3. **ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1.

At this point, both the client and server have received an acknowledgement of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

3 way handshake



The numbers in brackets indicate offsets from a randomly generated numbers on each side, of SEQ and ACK numbers, where x is undefined. In practice x will be zero.

Connecting to a server

Connecting to a server involves the SMSQ/E Open channel routine calling the 'TCPCONNECT' routine.

This routine will try to make a 'Three Way Handshake' connection to a TCP server that is 'Listening' for connection requests.

At this point, as far as SMSQ/E is concerned, The channel has not yet been opened, so no normal I/O requests can be processed. The area that will be the channel definition block is loaded with data to send a SYN message, and then sends it.

The routine then waits for the interrupt driven background packet reading routines to receive the SYN,ACK message. This is handled by the 'IS_TCP_DECODE' routine, which sets the socket status byte in the channel definition block (nd_sock_state) to 'ESTABLISHED'

The routine then sends an ACK message, completing the connection.

Server accepting a connection

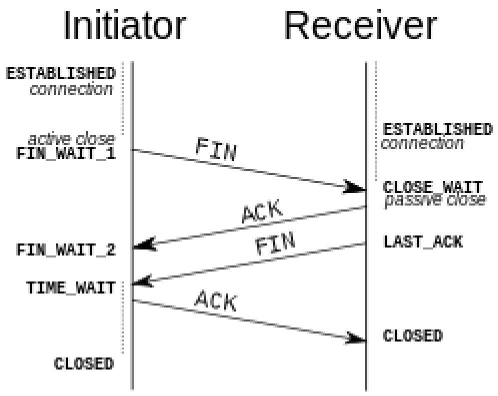
The server has a 'Listening' channel that waits for incoming connection requests. When a 'SYN' is received by the background packet reading routines, for a listening channel. Then a dummy channel definition block is created to handle the 3 way handshake (is_sendSYNACK). This dummy channel definition block is then added to a list of queued requests, and the linked list of open IP channels.

When an IP_ACCEPT system trap is called (tcp_accept), the request queue of the supplied listening channel is scanned for the oldest queued request of dummy channel definition blocks. This dummy block is then unlinked, and a new channel definition block (that will be the real one) is created. Data is copied from the dummy block to the real one, and the dummy block is then removed.

Closing a connection

Closing a connection involves a '4 way handshake', or a '3 way handshake' process. It's a bit more complicated than making a 'connection', and involves the socket (channel) going through a number of states. Depending on which side initiates the close. And one side may leave a channel open as far as SMSQ/E is concerned.

Below is shown the sequence of messages to terminate a TCP connection. As taken from the TCP Wikipedia page.



Sample SEQ and ACK numbers for each step

FIN -> 100,300

ACK <- 300,101
FIN <- 300,101

ACK -> 101,301

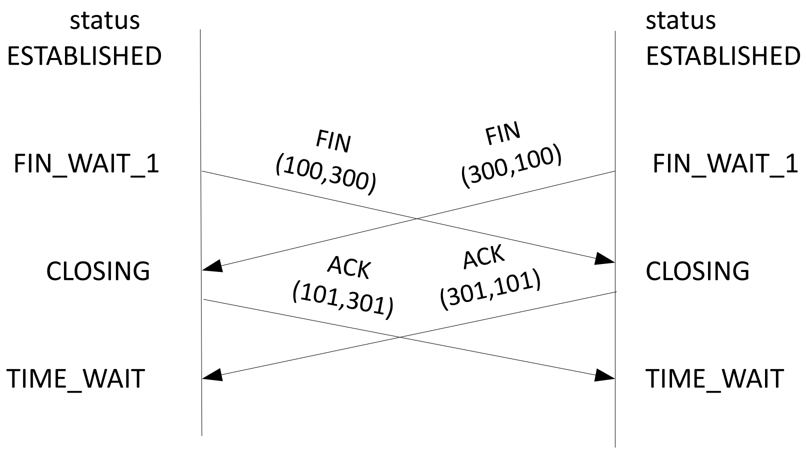
The '3 way handshake' involves the middle two messages being combined into one message.
The 'Close channel' routine will send anything left in its buffer, then call the 'TCP_CLOSE' routine.

The 'TCP_CLOSE' routine co-operates with the 'IS_TCP_DECODE' routine, as in making a connection. To move the channels socket status through the various stages, depending on which side initiates the close.

Note that in real packet transfers, extra flags may be set in the messages that are sent. For example, the first message sent may be, FIN, or FIN,ACK, or PSH,FIN,ACK.

The 'IS_TCP_DECODE' routine tries to account for all these combinations, and also the state that the socket is currently in.

There is also the possibility that both ends of the TCP connection will try to initiate the close at the same time. In this situation, the process is slightly different



The numbers in brackets indicate sample SEQ and ACK numbers

Ethernet Header Formats

Physical (Ethernet) layer header format 14 bytes

$00	6 bytes	destination MAC address	
$06	6 bytes	source MAC address	
$0C	word	length/type	
		<$0800 Length of the packet	
		$0800	Ethernet IPV4 datagram
		$0806	ARP Frame
		$0835	RARP
		$8100	IEEE802.1 Q tag 10/100 VLAN Frame
		$86DD	IPV6
		$8808	10/100 Control Frame

Network (IPV4) layer header format 20 bytes Used by ICMP, IGMP, TCP, UDP, ENCAP, OSPF, SCTP

$00	byte	Version/HL, Upper nibble=Version, Lower nibble=IHL
$01	byte	Type of service, Bits 7-2=DSCP, Bits 1-0=ECN
$02	word	Length
$04	word	Identification
$06	word	Flags and offset, Bits 15-13=Flags, Bits 12-0=Fragment offset
$08	byte	TTL Time to live
$09	byte	Protocol
$0A	word	Checksum
$0C	long	Source IP address
$10	long	Destination IP address

Description of header format

Version

The first header field in an IP packet is the four-bit version field. For IPv4, this is always equal to 4.

Internet Header Length (IHL)

The IPv4 header is variable in size due to the optional 14th field (options). The IHL field contains the size of the IPv4 header, it has 4 bits that specify the number of 32-bit words in the header. The minimum value for this field is 5,[26] which indicates a length of 5 × 32 bits = 160 bits = 20 bytes. As a 4-bit field, the maximum value is 15, this means that the maximum size of the IPv4 header is 15 × 32 bits, or 480 bits = 60 bytes.

Differentiated Services Code Point (DSCP)

Originally defined as the type of service (ToS), this field specifies differentiated services (DiffServ) per RFC 2474 (updated by RFC 3168 and RFC 3260). New technologies are emerging that require real-time data streaming and therefore make use of the DSCP field. An example is Voice over IP (VoIP), which is used for interactive voice services.

Explicit Congestion Notification (ECN)

This field is defined in RFC 3168 and allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that is only used when both endpoints support it and are willing to use it. It is effective only when supported by the underlying network.

Total Length

This 16-bit field defines the entire packet size in bytes, including header and data. The minimum size is 20 bytes (header without data) and the maximum is 65,535 bytes. All hosts are required to be able to reassemble datagrams of size up to 576 bytes, but most modern hosts handle much larger packets. Sometimes links impose further restrictions on the packet size, in which case datagrams must be fragmented. Fragmentation in IPv4 is handled in either the host or in routers.

Identification

This field is an identification field and is primarily used for uniquely identifying the group of fragments of a single IP datagram. Some experimental work has suggested using the ID field for other purposes, such as for adding packet-tracing information to help trace datagrams with spoofed source addresses,[27] but RFC 6864 now prohibits any such use.
If IP packet is fragmented during the transmission, all the fragments contain same identification number. to identify original IP packet they belong to.

Flags

A three-bit field follows and is used to control or identify fragments. They are (in order, from most significant to least significant):

- bit 0: Reserved; must be zero.[note 1]
- bit 1: Don't Fragment (DF)
- bit 2: More Fragments (MF)

If the DF flag is set, and fragmentation is required to route the packet, then the packet is dropped. This can be used when sending packets to a host that does not have resources to handle fragmentation. It can also be used for path MTU discovery, either automatically by the host IP software, or manually using diagnostic tools such as ping or traceroute. For unfragmented packets, the MF flag is cleared. For fragmented packets, all fragments except the last have the MF flag set. The last fragment has a non-zero Fragment Offset field, differentiating it from an unfragmented packet.

Fragment Offset

The fragment offset field is measured in units of eight-byte blocks. It is 13 bits long and specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram. The first fragment has an offset of zero. This allows a maximum offset of (213 – 1) × 8 = 65,528 bytes, which would exceed the maximum IP packet length of 65,535 bytes with the header length included (65,528 + 20 = 65,548 bytes).
The fragment offsets are calculated from the start of the transport layer

Time To Live (TTL)

An eight-bit time to live field helps prevent datagrams from persisting (e.g. going in circles) on an internet. This field limits a datagram's lifetime. It is specified in seconds, but time intervals less than 1 second are rounded up to 1. In practice, the field has become a hop count—when the datagram arrives at a router, the router decrements the TTL field by one. When the TTL field hits zero, the router discards the packet and typically sends an ICMP Time Exceeded message to the sender. The program traceroute uses these ICMP Time Exceeded messages to print the routers used by packets to go from the source to the destination.

Protocol

This field defines the protocol used in the data portion of the IP datagram. The Internet Assigned Numbers Authority maintains a list of IP protocol numbers as directed by RFC 790.
Tells the Network layer at the destination host, to which Protocol this packet belongs to, i.e. the next level Protocol. For example protocol number of ICMP is 1, TCP is 6 and UDP is 17.

Some of the common payload protocols are:

Protocol Number	Protocol Name	Abbreviation
1	Internet Control Message Protocol	ICMP
2	Internet Group Management Protocol	IGMP
6	Transmission Control Protocol	TCP
17	User Datagram Protocol	UDP
41	IPv6 encapsulation	ENCAP
89	Open Shortest Path First	OSPF
132	Stream Control Transmission Protocol	SCTP

Header Checksum

The 16-bit IPv4 header checksum field is used for error-checking of the header. When a packet arrives at a router, the router calculates the checksum of the header and compares it to the checksum field. If the values do not match, the router discards the packet. Errors in the data field must be handled by the encapsulated protocol. Both UDP and TCP have checksum fields.

When a packet arrives at a router, the router decreases the TTL field. Consequently, the router must calculate a new checksum.

Source address

This field is the IPv4 address of the sender of the packet. Note that this address may be changed in transit by a network address translation device.

Destination address

This field is the IPv4 address of the receiver of the packet. As with the source address, this may be changed in transit by a network address translation device.

Options

The options field is not often used. Note that the value in the IHL field must include enough extra 32-bit words to hold all the options (plus any padding needed to ensure that the header contains an integer number of 32-bit words). The list of options may be terminated with an EOL (End of Options List, 0x00) option; this is only necessary if the end of the options would not otherwise coincide with the end of the header. The possible options that can be put in the header are as follows:

Field	Size (bits)	Description
Copied	1	Set to 1 if the options need to be copied into all fragments of a fragmented packet.
Option Class	2	A general options category. 0 is for "control" options, and 2 is for "debugging and measurement". 1 and 3 are reserved.
Option Number	5	Specifies an option.
Option Length	8	Indicates the size of the entire option (including this field). This field may not exist for simple options.
Option Data	Variable	Option-specific data. This field may not exist for simple options.

Note: If the header length is greater than 5 (i.e., it is from 6 to 15) it means that the options field is present and must be considered.

Note: Copied, Option Class, and Option Number are sometimes referred to as a single eight-bit field, the Option Type.

Packets containing some options may be considered as dangerous by some routers and be blocked

Network (ARP) layer format 28 bytes

$00	word	HDR	Hardware type, $0001 ethernet
$02	word	PRO	Protocol, $0800=ethernet internet protocol
$04	byte	HLN	MAC address length, usually 6
$05	byte	PLN	IP address length, usually 4
$06	word	OP	Operation, 1=request, 2=reply
$08	6 bytes	SHA	Sender MAC address
$0E	long	SPA	Sender IP address
$12	6 bytes	THA	Target MAC address
$18	long	TPA	Target IP address

Description of header format

HDR 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

1	Ethernet (10Mb)
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 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 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 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 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

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 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 Sender Protocol Address

The IP address of the device sending this message.

THA 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 Target Protocol Address

The IP address of the device this message is being sent to.

Transport (UDP) layer format

Pseudo header for checksum calculation. Not to be included in actual header 12 bytes

$00	long	Source IP address
$04	long	Destination IP address
$08	byte	zero
$09	byte	$11 (17) UDP
$0A	word	UDP length from actual header

Actual header 8 bytes

$00	word	Source port
$02	word	Destination port
$04	word	UDP length, payload length + 8 bytes of header
$06	word	UDP checksum

Transport (TCP) layer format

Pseudo header for checksum calculation. Not to be included in actual header 12 bytes

$00	long	Source IP address
$04	long	Destination IP address
$08	byte	zero
$09	byte	$06 TCP
$0A	word	TCP length, Actual header length + options + payload

Actual header 20 bytes

$00	word	Source port
$02	word	Destination port
$04	long	Sequence number
$08	long	ACK Sequence number
$0C	word	offset/res/flags
$0E	word	Window
$10	word	Checksum
$12	word	Urgent pointer
$14		Options bytes

Description of header format

Source Port

The source port number.

Destination Port

The destination port number.

Sequence Number

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 3 bits

Reserved for future use. Must be zero.

Control Bits: 9 bits (from left to right):

NS: ECN - nonce - concealment protection
CWR: Congestion Window Reduced
ECE: ECN - Echo has a dual role
URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

```
O O O O R R R C C C C C C C C C
                      |     |-------- FIN
                      |--------------- PSH
```

Window

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers the 96 bit pseudo header conceptually

Urgent Pointer

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.
3	3	Window scale.
4	2	TCPSACK permitted.

Specific Option Definitions

End of Option List
Kind=0

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation
Kind=1

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

```
+-------------+-------------+-------------+------------+
|00000010|00000100|      max seg size      |
 +------------+--------------+------------+------------+
Kind=2   Length=4
```

Maximum Segment Size Option Data: 16 bits
If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Window Scale

```
+-------------+-------------+--------------+-------------+
|00000001|00000011|00000011|     scale    |
 +------------+-------------+--------------+-------------+
Kind=3   Length=3
```

The scale factor is the number of bits to left shift the 16 bit window size (ignored in SYN message)

TCPASCK permitted

```
+-------------+-------------+--------------+-------------+
|00000001|00000001|00000100|00000010|
 +------------+-------------+--------------+-------------+
Kind=4   Length=2
```

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

Transport (Ping) layer format   ICMP   8 bytes

$00	byte	Type, 8=IPV4 request, 0=IPV4 reply Type of ICMP message
$01	byte	Code, 0
$02	word	Header checksum - including payload
$04	word	Identifier
$06	word	Sequence number
		32 byte payload

Transport (IGMP) layer format   8 bytes

$00	byte	Type, Membership Query, membership Report, Leave group
$01	byte	Max response time, only in Membership Query messages
$02	word	Checksum
$04	long	Group address, Behaviour of this field varies by the type of message sent:

Membership Query: (set to)
  General Query: All zeroes
  Group Specific Query: multicast group address
Membership Report: multicast group address
Leave Group: multicast group address

Fragmented Packet layout

First packet :-
Standard Physical layer.
Standard Network layer, other than Flags/offset = $2000
Transport layer has a length (and checksum?) for the entire unfragmented payload.

Second packet:-
Standard Physical layer.
Standard Network layer, other than Flags = %001 or %000 on the last fragment.
Offset = offset from start of the payload, divided by 8.
<u>There is no transport layer</u>

ARP Handling

The handling of acquiring and supplying MAC addresses works as follows.

When the background packet receiving routine receives an ARP packet. If the packet is a Gratuitous request, or a reply to a request we made. Then the details are entered into the ARP table.
If it is a request for our MAC address, then a reply is sent to the sender.

When the systems IP address is set, or changed. Then a Gratuitous packet is broadcast over the network.

When the required MAC address, for an IP address, is not availablefrom the ARP table. The method for obtaining it goes along these lines.

When a channel is opened with OPEN_IN, If there is not a MAC address entry in the ARP table, then a ARP request is sent. A 40 second timer, and a flag, are initialized in the channel definition block to indicate that the MAC address is invalid. The channel open, then finishes normally.

What happens next depends on how quickly the ARP request is replied to, and how much time passes between opening the channel, and trying to do any I/O.

Assuming a worst case scenario, where there is never an ARP reply received, and channel I/O starts straight after opening the channel. The process will be as follows.

When <u>any</u> I/O is done to the channel, there will be a wait for 20 seconds, followed by another ARP request, In case the first one way lost. Then a further wait of 20 seconds, followed by a final ARP request. The timer will be reset to another 40 seconds, ETH_ERRNO will be set to error 34, and finally a system 'transmission error' will be returned.

If an ARP reply is received within the 40 seconds, then the above procedure will terminate, and I/O will continue normally.

DHCP Handling

The Q68 driver implements a DHCP client as a background job.

The basic theory of operation is as follows.

When the DHCP Client job is started, it tries to contact a DHCP server by sending DHCP DISCOVER packets every 12 seconds for up to 2 minutes. If it cannot obtain any offers of IP addresses, or if the user presses CTRL-SPACE, the job will end itself.

Upon receiving, and accepting an offer of an IP address. The DHCP client will then go to sleep, waking every 5 minutes to check the lease times. When renewal time is reached, the DHCP client will attempt every 5 minutes to renew the current lease from the DHCP server that supplied it. If it is unable to do this by the time it reaches the rebinding time, it will then attempt every 5 minutes to renew the current lease from any DHCP server. If it is unable to do this by the time it reaches the lease end time, the DHCP client it will end itself.

DHCP client job maintains a data block pointed to by A6 as follows

----

$00	dh_spare	7 spare bytes
$07	dh_status	client status
$08	dh_ciaddr	client IP
$0C	dh_yiaddr	your IP
$10	dh_siaddr	server IP
$14	dh_giaddr	relay agent IP
$18	dh_givenIP	IP address assigned
$1C	dh_serverIP	assinging server IP address
$20	dh_serverMAC	MAC address of assigning server
$26		2 spare bytes
$28	dh_routerIP	supplied router IP address
$2C	dh_leaseStart	lease start time
$30	dh_leaseRenewal	lease renewal time T1
$34	dh_leaseRebind	lease rebind time T2
$38	dh_leaseEnd	lease end offset/time
$3C	dh_domainServer	domain name server IP address
$40	dh_domainName	domain name up to 64 bytes in standard QL format
$82	dh_end	

General DHCP process

DHCP uses UDP as its transport protocol. DHCP messages from a client to a server are sent to the 'DHCP server' port (67), and DHCP messages from a server to a client are sent to the 'DHCP client' port (68). A server with multiple network address (e.g., a multi-homed host) MAY use any of its network addresses in outgoing DHCP messages.

The client computer sends a broadcast 'DHCP DISCOVER' request, looking for a DHCP server to answer.

The server receives the DISCOVER packet. The server determines an appropriate address (if any) to give to the client. The server then temporarily reserves that address for the client and sends back to the client a 'DHCP OFFER' packet, with that address information. The server also configures the client's DNS servers, WINS servers, NTP servers, and sometimes other services as well.

The client sends a 'DHCP REQUEST' packet, letting the server know that it intends to use the address.

The server sends an 'DHCP ACK' packet, confirming that the client has a been given a lease on the address for a server-specified period of time.

DHCP header block

$00	byte	OP	Operation
$01	byte	HTYPE	Hardware type
$02	byte	SHLEN	Hardware address length (6 for MAC address)
$03	byte	HOPS	
$04	long	XID	Transaction ID
$08	word	SECS	Seconds elapsed
$0A	word	FLAGS	
$0C	long	CIADDR	Client IP address
$10	long	YIADDR	Your IP address
$14	long	SIADDR	Next server IP address
$18	long	GIADDR	Relay agent IP adderss
$1C	16 bytes	CHADDR	Client hardware address (MAC) padded out with 0's
$2C	64 bytes	SNAME	Server host name
$6C	128 bytes	FILE	Boot file name
$EC	variable	OPTIONS	Options

Description of header format

OP	Message op code / message type. This field specifies the type of the message, A request, or a reply. 1 = BOOTREQUEST 2 = BOOTREPLY
HTYPE	Hardware address type. This field specifies the hardware type. 1 = 10mb Ethernet
HLEN	Hardware address length (e.g. '6' for 10mb Ethernet).
HOPS	Client sets to zero, optionally used by relay agents when booting via a relay agent.

XID Transaction ID, a random number chosen by the client, used by the client and server to associate messages and responses between a client and a server.

SECS Filled in by client, seconds elapsed since client began address acquisition or renewal process.

FLAGS Flags. (see figure 2).
To work around some clients that cannot accept IP unicast datagrams before the TCP/IP software is configured, DHCP uses the 'flags' field. The leftmost bit is defined as the BROADCAST (B) flag. The remaining bits of the flags field are reserved for future use. They MUST be set to zero by clients and ignored by servers and relay agents.

```
                     1 1 1 1 1 1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|B|             MBZ             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

B: BROADCAST flag

MBZ: MUST BE ZERO (reserved for future use)

CIADDR Client IP address; only filled in if client is in BOUND, RENEW or REBINDING state and can respond to ARP requests.

YIADDR 'your' (client) IP address.

SIADDR IP address of next server to use in bootstrap; returned in DHCPOFFER, DHCPACK by server.

GIADDR Relay agent IP address, used in booting via a relay agent.

CHADDR Client hardware address.

SNAME Optional server host name, null terminated string.

FILE Boot file name, null terminated string; "generic" name or null in DHCPDISCOVER, fully qualified directory-path name in DHCPOFFER.

OPTIONS Optional parameters field.

The 'options' field is variable length. A DHCP client must be prepared to receive DHCP messages with an 'options' field of at least length 312 octets. This requirement implies that a DHCP client must be prepared to receive a message of up to 576 octets, the minimum IP datagram size an IP host must be prepared to accept.

The first four octets of the 'options' field of the DHCP message contain the (decimal) values 99, 130, 83 and 99 ($63, $82, $53, $63), respectively (this is the same magic cookie as is defined in RFC 1497 [17]).

Several options have been defined so far. One particular option - the "DHCP message type" option - must be included in every DHCP message. This option defines the "type" of the DHCP message.

Additional options may be allowed, required, or not allowed, depending on the DHCP message type.

The last option must always be the 'end' option.

The following text is taken from the RFC1533 document.

BOOTP Extension/DHCP Option Field Format

DHCP options have the same format as the BOOTP "vendor extensions" defined in RFC 1497 [2]. Options may be fixed length or variable length. All options begin with a tag octet, which uniquely identifies the option. Fixed-length options without data consist of only a tag octet. Only options 0 and 255 are fixed length. All other options are variable-length with a length octet following the tag octet. The value of the length octet does not include the two octets specifying the tag and length. The length octet is followed by "length" octets of data. In the case of some variable-length options the length field is a constant but must still be specified.

All multi-octet quantities are in network byte-order.

When used with BOOTP, the first four octets of the vendor information field have been assigned to the "magic cookie" (as suggested in RFC 951). This field identifies the mode in which the succeeding data is to be interpreted. The value of the magic cookie is the 4 octet dotted decimal 99.130.83.99 (or hexadecimal number 63.82.53.63) in network byte order.

All of the "vendor extensions" defined in RFC 1497 are also DHCP options.

Option codes 128 to 254 (decimal) are reserved for site-specific options.

Except for the options in section 9, all options may be used with either DHCP or BOOTP.

Many of these options have their default values specified in other documents. In particular, RFC 1122 [4] specifies default values for most IP and TCP configuration parameters.

3. RFC 1497 Vendor Extensions

This section lists the vendor extensions as defined in RFC 1497.
They are defined here for completeness.

3.1. Pad Option
The pad option can be used to cause subsequent fields to align on word boundaries.

The code for the pad option is 0, and its length is 1 octet.

```
 Code
+-----+
|  0  |
+-----+
```

3.2. End Option
The end option marks the end of valid information in the vendor field. Subsequent octets should be filled with pad options.

The code for the end option is 255, and its length is 1 octet.

```
 Code
+------+
| 255 |
+------+
```

3.3. Subnet Mask
The subnet mask option specifies the client's subnet mask as per RFC 950 [5].

If both the subnet mask and the router option are specified in a DHCP reply, the subnet mask option MUST be first.

The code for the subnet mask option is 1, and its length is 4 octets.

```
 Code   Len       Subnet Mask
+-----+-----+------+------+------+-------+
|  1  |  4  | m1 |  m2 |  m3 |  m4 |
+-----+-----+------+------+------+-------+
```

3.4. Time Offset
The time offset field specifies the offset of the client's subnet in seconds from Coordinated Universal Time (UTC). The offset is expressed as a signed 32-bit integer.

The code for the time offset option is 2, and its length is 4 octets.

```
 Code   Len       Time Offset
+-----+-----+-----+------+-----+-----+
|  2  |  4  | n1 |  n2 |  n3 |  n4 |
+-----+-----+-----+------+-----+-----+
```

3.5. Router Option
The router option specifies a list of IP addresses for routers on the client's subnet. Routers SHOULD be listed in order of preference.

The code for the router option is 3. The minimum length for the router option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len       Address 1            Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  3  |  n  | a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.6. Time Server Option

The time server option specifies a list of RFC 868 [6] time servers available to the client. Servers SHOULD be listed in order of preference.

The code for the time server option is 4. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1               Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  4  |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.7. Name Server Option

The name server option specifies a list of IEN 116 [7] name servers available to the client. Servers SHOULD be listed in order of preference.

The code for the name server option is 5. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1               Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  5  |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.8. Domain Name Server Option

The domain name server option specifies a list of Domain Name System (STD 13, RFC 1035 [8]) name servers available to the client. Servers SHOULD be listed in order of preference.

The code for the domain name server option is 6. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1               Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  6  |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.9. Log Server Option

The log server option specifies a list of MIT-LCS UDP log servers available to the client. Servers SHOULD be listed in order of preference.

The code for the log server option is 7. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1               Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  7  |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.10. Cookie Server Option

The cookie server option specifies a list of RFC 865 [9] cookie servers available to the client. Servers SHOULD be listed in order of preference.

The code for the log server option is 8. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1              Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  8  |  n  | a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.11. LPR Server Option

The LPR server option specifies a list of RFC 1179 [10] line printer servers available to the client. Servers SHOULD be listed in order of preference.

The code for the LPR server option is 9. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1              Address 2
+-----+-----+-----+-----+------+-----+-----+------+--
|  9  |  n  | a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+-----+------+--
```

3.12. Impress Server Option

The Impress server option specifies a list of Imagen Impress servers available to the client. Servers SHOULD be listed in order of preference.

The code for the Impress server option is 10. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1              Address 2
+-----+-----+-----+-----+------+-----+------+-----+--
|  10 |  n  | a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+------+-----+--
```

3.13. Resource Location Server Option

This option specifies a list of RFC 887 [11] Resource Location servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 11. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len        Address 1              Address 2
+-----+-----+-----+-----+------+-----+------+-----+--
|  11 |  n  | a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
+-----+-----+-----+-----+------+-----+------+-----+--
```

3.14. Host Name Option

This option specifies the name of the client. The name may or may not be qualified with the local domain name (see section 3.17 for the preferred way to retrieve the domain name). See RFC 1035 for character set restrictions.

The code for this option is 12, and its minimum length is 1.

```
 Code   Len              Host Name
+-----+-----+-----+------+-----+------+-----+------+--
|  12 |  n  |  h1 |  h2 |  h3 |  h4 |  h5 |  h6 |  ...
+-----+-----+-----+------+-----+------+-----+------+--
```

3.15. Boot File Size Option

This option specifies the length in 512-octet blocks of the default boot image for the client. The file length is specified as an unsigned 16-bit integer.

The code for this option is 13, and its length is 2.

```
 Code   Len   File Size
+-----+-----+-----+-----+
|  13 |  2  |  l1  |  l2 |
+-----+-----+-----+-----+
```

3.16. Merit Dump File

This option specifies the path-name of a file to which the client's core image should be dumped in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 14. Its minimum length is 1.

```
 Code   Len      Dump File Pathname
+-----+-----+-----+------+-----+------+---
|  14 |  n  |  n1 |  n2 |  n3 |  n4 | ...
+-----+-----+-----+------+-----+------+---
```

3.17. Domain Name

This option specifies the domain name that client should use when resolving hostnames via the Domain Name System.

The code for this option is 15. Its minimum length is 1.

```
 Code   Len       Domain Name
+-----+-----+-----+------+-----+------+--
|  15 |  n  |  d1 |  d2 |  d3 |  d4 |  ...
+-----+-----+-----+------+-----+------+--
```

3.18. Swap Server

This specifies the IP address of the client's swap server.

The code for this option is 16 and its length is 4.

```
 Code   Len    Swap Server Address
+-----+-----+-----+------+-----+-----+
|  16 |  n  |  a1 |  a2 |  a3 |  a4 |
+-----+-----+-----+------+-----+-----+
```

3.19. Root Path

This option specifies the path-name that contains the client's root disk. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 17. Its minimum length is 1.

```
 Code   Len      Root Disk Pathname
+-----+-----+------+-----+-----+------+---
|  17 |  n  |  n1 |  n2 |  n3 |  n4 | ...
+-----+-----+------+-----+-----+------+---
```

3.20. Extensions Path

A string to specify a file, retrievable via TFTP, which contains information which can be interpreted in the same way as the 64-octet vendor-extension field within the BOOTP response, with the following exceptions:

- the length of the file is unconstrained;
- all references to Tag 18 (i.e., instances of the BOOTP Extensions Path field) within the file are ignored.

The code for this option is 18. Its minimum length is 1.

```
 Code   Len      Extensions Pathname
+-----+-----+-----+------+-----+------+---
|  18 |  n  |  n1 |  n2 |  n3 |  n4 | ...
+-----+-----+-----+------+-----+------+---
```

4. IP Layer Parameters per Host

This section details the options that affect the operation of the IP layer on a per-host basis.

4.1. IP Forwarding Enable/Disable Option

This option specifies whether the client should configure its IP layer for packet forwarding. A value of 0 means disable IP forwarding, and a value of 1 means enable IP forwarding.

The code for this option is 19, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  19 |  1  | 0/1 |
+-----+-----+------+
```

4.2. Non-Local Source Routing Enable/Disable Option

This option specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes (see Section 3.3.5 of [4] for a discussion of this topic). A value of 0 means disallow forwarding of such datagrams, and a value of 1 means allow forwarding.

The code for this option is 20, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  20 |  1  | 0/1 |
+-----+-----+------+
```

4.3. Policy Filter Option

This option specifies policy filters for non-local source routing.
The filters consist of a list of IP addresses and masks which specify destination/mask pairs with which to filter incoming source routes.

Any source routed datagram whose next-hop address does not match one of the filters should be discarded by the client.

The code for this option is 21. The minimum length of this option is 8, and the length MUST be a multiple of 8.

```
 Code   Len       Address 1              Mask 1
+-----+-----+-----+------+-----+-----+-------+------+------+-------+
|  21 |  n  |  a1 |  a2 |  a3 |  a4 |  m1 |  m2 |  m3 |  m4 |
+-----+-----+-----+------+-----+-----+-------+------+------+-------+
       Address 2              Mask 2
+-----+-----+------+-----+------+-------+------+-------+---
|  a1 |  a2 |  a3 |  a4 |  m1 |  m2 |  m3 |  m4 | ...
+-----+-----+------+-----+------+-------+------+-------+---
```

4.4. Maximum Datagram Reassembly Size

This option specifies the maximum size datagram that the client should be prepared to reassemble. The size is specified as a 16-bit unsigned integer. The minimum value legal value is 576.

The code for this option is 22, and its length is 2.

```
 Code   Len     Size
+-----+-----+-----+-----+
|  22 |  2  |  s1 |  s2 |
+-----+-----+-----+-----+
```

4.5. Default IP Time-to-live

This option specifies the default time-to-live that the client should use on outgoing datagrams. The TTL is specified as an octet with a value between 1 and 255.

The code for this option is 23, and its length is 1.

```
 Code   Len   TTL
+-----+-----+-----+
|  23 |  1  | ttl  |
+-----+-----+-----+
```

4.6. Path MTU Aging Timeout Option
This option specifies the timeout (in seconds) to use when aging Path MTU values discovered by the mechanism defined in RFC 1191 [12]. The timeout is specified as a 32-bit unsigned integer.

The code for this option is 24, and its length is 4.

```
 Code   Len         Timeout
+-----+-----+-----+-----+-----+-----+
|  24 |  4  |  t1 |  t2 |  t3  |  t4 |
+-----+-----+-----+-----+-----+-----+
```

4.7. Path MTU Plateau Table Option
This option specifies a table of MTU sizes to use when performing Path MTU Discovery as defined in RFC 1191. The table is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest.
The minimum MTU value cannot be smaller than 68.

The code for this option is 25. Its minimum length is 2, and the length MUST be a multiple of 2.

```
 Code   Len     Size 1      Size 2
+-----+-----+-----+-----+-----+-----+---
|  25 |  n  |  s1 |  s2 |  s1 |  s2 | ...
+-----+-----+-----+-----+-----+-----+---
```

5. IP Layer Parameters per Interface
This section details the options that affect the operation of the IP layer on a per-interface basis. It is expected that a client can issue multiple requests, one per interface, in order to configure interfaces with their specific parameters.

5.1. Interface MTU Option
This option specifies the MTU to use on this interface. The MTU is specified as a 16-bit unsigned integer. The minimum legal value for the MTU is 68.

The code for this option is 26, and its length is 2.

```
 Code   Len      MTU
+-----+-----+------+-------+
|  26 |  2  |  m1 |  m2 |
+-----+-----+------+-------+
```

5.2. All Subnets are Local Option
This option specifies whether or not the client may assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. A value of 1 indicates that all subnets share the same MTU. A value of 0 means that the client should assume that some subnets of the directly connected network may have smaller MTUs.

The code for this option is 27, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  27 |  1  | 0/1 |
+-----+-----+------+
```

5.3. Broadcast Address Option

This option specifies the broadcast address in use on the client's subnet. Legal values for broadcast addresses are specified in section 3.2.1.3 of [4].

The code for this option is 28, and its length is 4.

```
  Code   Len     Broadcast Address
 +-----+-----+-----+------+-----+------+
 |  28 |  4  |  b1 |  b2 |  b3 |  b4 |
 +-----+-----+-----+------+-----+------+
```

5.4. Perform Mask Discovery Option

This option specifies whether or not the client should perform subnet mask discovery using ICMP. A value of 0 indicates that the client should not perform mask discovery. A value of 1 means that the client should perform mask discovery.

The code for this option is 29, and its length is 1.

```
  Code   Len  Value
 +-----+-----+------+
 |  29 |  1  | 0/1 |
 +-----+-----+------+
```

5.5. Mask Supplier Option

This option specifies whether or not the client should respond to subnet mask requests using ICMP. A value of 0 indicates that the client should not respond. A value of 1 means that the client should respond.

The code for this option is 30, and its length is 1.

```
  Code   Len  Value
 +-----+-----+------+
 |  30 |  1  | 0/1 |
 +-----+-----+------+
```

5.6. Perform Router Discovery Option

This option specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in RFC 1256 [13]. A value of 0 indicates that the client should not perform router discovery. A value of 1 means that the client should perform router discovery.

The code for this option is 31, and its length is 1.

```
  Code   Len  Value
 +-----+-----+------+
 |  31 |  1  | 0/1 |
 +-----+-----+------+
```

5.7. Router Solicitation Address Option
This option specifies the address to which the client should transmit router solicitation requests.

The code for this option is 32, and its length is 4.

```
 Code   Len         Address
+-----+-----+-----+------+-----+-----+
|  32 |  4  |  a1 |  a2 |  a3 |  a4 |
+-----+-----+-----+------+-----+-----+
```

5.8. Static Route Option
This option specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are listed in descending order of priority.

The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination.

The default route (0.0.0.0) is an illegal destination for a static route. See section 3.5 for information about the router option.

The code for this option is 33. The minimum length of this option is 8, and the length MUST be a multiple of 8.

```
 Code   Len       Destination 1         Router 1
+-----+-----+-----+------+-----+-----+-----+-----+-----+-----+
|  33 |  n  |  d1 |  d2 |  d3 |  d4 |  r1 |  r2 |  r3 |  r4 |
+-----+-----+-----+------+-----+-----+-----+-----+-----+-----+
       Destination 2         Router 2
+-----+------+-----+------+-----+-----+-----+----+---
|  d1 |  d2 |  d3 |  d4 |  r1 |  r2 |  r3 |  r4 | ...
+-----+------+-----+------+-----+-----+-----+----+---
```

6. Link Layer Parameters per Interface
This section lists the options that affect the operation of the data link layer on a per-interface basis.

6.1. Trailer Encapsulation Option
This option specifies whether or not the client should negotiate the use of trailers (RFC 893 [14]) when using the ARP protocol. A value of 0 indicates that the client should not attempt to use trailers. A value of 1 means that the client should attempt to use trailers.

The code for this option is 34, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  34 |  1  | 0/1 |
+-----+-----+------+
```

6.2. ARP Cache Timeout Option
This option specifies the timeout in seconds for ARP cache entries.
The time is specified as a 32-bit unsigned integer.

The code for this option is 35, and its length is 4.

```
 Code   Len         Time
+-----+-----+-----+-----+-----+-----+
|  35 |  4  |  t1 |  t2  |  t3 |  t4 |
+-----+-----+-----+-----+-----+-----+
```

6.3. Ethernet Encapsulation Option
This option specifies whether or not the client should use Ethernet Version 2 (RFC 894 [15]) or IEEE 802.3 (RFC 1042 [16]) encapsulation if the interface is an Ethernet. A value of 0 indicates that the client should use RFC 894 encapsulation. A value of 1 means that the client should use RFC 1042 encapsulation.

The code for this option is 36, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  36 |  1  | 0/1 |
+-----+-----+------+
```

7. TCP Parameters
This section lists the options that affect the operation of the TCP layer on a per-interface basis.

7.1. TCP Default TTL Option
This option specifies the default TTL that the client should use when sending TCP segments. The value is represented as an 8-bit unsigned integer. The minimum value is 1.

The code for this option is 37, and its length is 1.

```
 Code   Len   TTL
+-----+-----+-----+
|  37 |  1  |  n   |
+-----+-----+-----+
```

7.2. TCP Keepalive Interval Option
This option specifies the interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection.
The time is specified as a 32-bit unsigned integer. A value of zero indicates that the client should not generate keepalive messages on connections unless specifically requested by an application.

The code for this option is 38, and its length is 4.

```
 Code   Len         Time
+-----+-----+-----+-----+-----+-----+
|  38 |  4  |  t1 |  t2 |  t3  |  t4 |
+-----+-----+-----+-----+-----+-----+
```

7.3. TCP Keepalive Garbage Option
This option specifies the whether or not the client should send TCP keepalive messages with an octet of garbage for compatibility with older implementations. A value of 0 indicates that a garbage octet should not be sent. A value of 1 indicates that a garbage octet should be sent.

The code for this option is 39, and its length is 1.

```
 Code   Len  Value
+-----+-----+------+
|  39 |  1  | 0/1  |
+-----+-----+------+
```

8. Application and Service Parameters
This section details some miscellaneous options used to configure miscellaneous applications and services.

8.1. Network Information Service Domain Option
This option specifies the name of the client's NIS [17] domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 40. Its minimum length is 1.

```
 Code   Len      NIS Domain Name
+-----+-----+-----+------+-----+------+---
|  40 |  n  |  n1 |  n2  |  n3 |  n4  | ...
+-----+-----+-----+------+-----+------+---
```

8.2. Network Information Servers Option
This option specifies a list of IP addresses indicating NIS servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 41. Its minimum length is 4, and the length MUST be a multiple of 4.

```
 Code   Len       Address 1            Address 2
+-----+-----+-----+------+-----+-----+------+-----+--
|  41 |  n  |  a1 |  a2  |  a3 |  a4 |  a1  |  a2 |  ...
+-----+-----+-----+------+-----+-----+------+-----+--
```

8.3. Network Time Protocol Servers Option
This option specifies a list of IP addresses indicating NTP [18] servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 42. Its minimum length is 4, and the length MUST be a multiple of 4.

```
 Code   Len       Address 1            Address 2
+-----+-----+-----+------+-----+-----+------+-----+--
|  42 |  n  |  a1 |  a2  |  a3 |  a4 |  a1  |  a2 |  ...
+-----+-----+-----+------+-----+-----+------+-----+--
```

8.4. Vendor Specific Information

This option is used by clients and servers to exchange vendor- specific information. The information is an opaque object of n octets, presumably interpreted by vendor-specific code on the clients and servers. The definition of this information is vendor specific. The vendor is indicated in the class-identifier option. Servers not equipped to interpret the vendor-specific information sent by a client MUST ignore it (although it may be reported). Clients which do not receive desired vendor-specific information SHOULD make an attempt to operate without it, although they may do so (and announce they are doing so) in a degraded mode.

If a vendor potentially encodes more than one item of information in this option, then the vendor SHOULD encode the option using "Encapsulated vendor-specific options" as described below:

The Encapsulated vendor-specific options field SHOULD be encoded as a sequence of code/length/value fields of identical syntax to the DHCP options field with the following exceptions:

1) There SHOULD NOT be a "magic cookie" field in the encapsulated vendor-specific extensions field.

2) Codes other than 0 or 255 MAY be redefined by the vendor within the encapsulated vendor-specific extensions field, but SHOULD conform to the tag-length-value syntax defined in section 2.

3) Code 255 (END), if present, signifies the end of the encapsulated vendor extensions, not the end of the vendor extensions field. If no code 255 is present, then the end of the enclosing vendor-specific information field is taken as the end of the encapsulated vendor-specific extensions field.

The code for this option is 43 and its minimum length is 1.

```
 Code   Len   Vendor-specific information
 +-----+-----+-----+-----+---
 |  43 |  n  |  i1 |  i2 | ...
 +-----+-----+-----+-----+---
```

When encapsulated vendor-specific extensions are used, the information bytes 1-n have the following format:

```
  Code   Len   Data item        Code   Len   Data item       Code
 +-----+-----+-----+------+-----+-----+-----+------+-----+-----+-----+
 |  T1 |  n  |  d1 |  d2  | ... |  T2 |  n  |  D1  |  D2 | ... | ... |
 +-----+-----+-----+------+-----+-----+-----+------+-----+-----+-----+
```

8.5. NetBIOS over TCP/IP Name Server Option

The NetBIOS name server (NBNS) option specifies a list of RFC 1001/1002 [19] [20] NBNS name servers listed in order of preference.

The code for this option is 44. The minimum length of the option is 4 octets, and the length must always be a multiple of 4.

```
  Code   Len         Address 1            Address 2
 +-----+-----+-----+------+-----+-----+------+-----+-----+------+----
 |  44 |  n  |  a1 |  a2  |  a3 |  a4 |  b1  |  b2 |  b3 |  b4  | ...
 +-----+-----+-----+------+-----+-----+------+-----+-----+------+----
```

8.6. NetBIOS over TCP/IP Datagram Distribution Server Option

The NetBIOS datagram distribution server (NBDD) option specifies a list of RFC 1001/1002 NBDD servers listed in order of preference. The code for this option is 45. The minimum length of the option is 4 octets, and the length must always be a multiple of 4.

```
 Code   Len         Address 1           Address 2
+-----+-----+-----+------+-----+-----+------+-----+------+-----+----
|  45 |  n  |  a1 |  a2  |  a3 |  a4 |  b1  |  b2 |  b3  |  b4 | ...
+-----+-----+-----+------+-----+-----+------+-----+------+-----+----
```

8.7. NetBIOS over TCP/IP Node Type Option

The NetBIOS node type option allows NetBIOS over TCP/IP clients which are configurable to be configured as described in RFC 1001/1002. The value is specified as a single octet which identifies the client type as follows:

```
  Value       Node Type
  -----       ---------
  0x1         B-node
  0x2         P-node
  0x4         M-node
  0x8         H-node
```

In the above chart, the notation '0x' indicates a number in base-16 (hexadecimal).

The code for this option is 46. The length of this option is always 1.

```
 Code   Len  Node Type
+-----+-----+---------------+
|  46 |  1  | see above |
+-----+-----+---------------+
```

8.8. NetBIOS over TCP/IP Scope Option

The NetBIOS scope option specifies the NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002. See [19], [20], and [8] for character-set restrictions.

The code for this option is 47. The minimum length of this option is 1.

```
 Code   Len       NetBIOS Scope
+-----+-----+-----+-----+-----+-----+----
|  47 |  n  |  s1 |  s2 |  s3 |  s4 | ...
+-----+-----+-----+-----+-----+-----+----
```

8.9. X Window System Font Server Option

This option specifies a list of X Window System [21] Font servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 48. The minimum length of this option is 4 octets, and the length MUST be a multiple of 4.

```
 Code   Len        Address 1            Address 2
+-----+-----+-----+------+-----+-----+------+-----+---
|  48 |  n  |  a1 |  a2  |  a3 |  a4 |  a1  |  a2 |   ...
+-----+-----+-----+------+-----+-----+------+-----+---
```

8.10. X Window System Display Manager Option
This option specifies a list of IP addresses of systems that are running the X Window System Display Manager and are available to the client.

Addresses SHOULD be listed in order of preference.

The code for the this option is 49. The minimum length of this option is 4, and the length MUST be a multiple of 4.

```
 Code   Len       Address 1            Address 2
+-----+-----+-----+------+-----+-----+------+-----+---
|  49 |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |   ...
+-----+-----+-----+------+-----+-----+------+-----+---
```

9. DHCP Extensions
This section details the options that are specific to DHCP.

9.1. Requested IP Address
This option is used in a client request (DHCPDISCOVER) to allow the client to request that a particular IP address be assigned.

The code for this option is 50, and its length is 4.

```
 Code   Len        Address
+-----+-----+-----+------+-----+-----+
|  50 |  4  |  a1 |  a2 |  a3 |  a4 |
+-----+-----+-----+------+-----+-----+
```

9.2. IP Address Lease Time
This option is used in a client request (DHCPDISCOVER or DHCPREQUEST) to allow the client to request a lease time for the IP address. In a server reply (DHCPOFFER), a DHCP server uses this option to specify the lease time it is willing to offer.

The time is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 51, and its length is 4.

```
 Code   Len       Lease Time
+-----+-----+-----+-----+-----+-----+
|  51 |  4  |  t1 |  t2 |  t3  |  t4 |
+-----+-----+-----+-----+-----+-----+
```

9.3. Option Overload
This option is used to indicate that the DHCP "sname" or "file" fields are being overloaded by using them to carry DHCP options. A DHCP server inserts this option if the returned parameters will exceed the usual space allotted for options.

If this option is present, the client interprets the specified additional fields after it concludes interpretation of the standard option fields.

The code for this option is 52, and its length is 1. Legal values for this option are:

```
  Value   Meaning
  -------   ------------
    1    the "file" field is used to hold options
    2    the "sname" field is used to hold options
    3    both fields are used to hold options
```

```
 Code   Len  Value
+-----+-----+-------+
|  52 |  1  |1/2/3|
+-----+-----+-------+
```

9.4. DHCP Message Type
his option is used to convey the type of the DHCP message. The code for this option is 53, and its length is 1. Legal values for this option are:

```
  Value   Message Type
  -------   -------------------
    1    DHCPDISCOVER
    2    DHCPOFFER
    3    DHCPREQUEST
    4    DHCPDECLINE
    5    DHCPACK
    6    DHCPNAK
    7    DHCPRELEASE
```

```
 Code   Len  Type
+-----+-----+------+
|  53 |  1  | 1-7 |
+-----+-----+------+
```

9.5. Server Identifier
This option is used in DHCPOFFER and DHCPREQUEST messages, and may optionally be included in the DHCPACK and DHCPNAK messages. DHCP servers include this option in the DHCPOFFER in order to allow the client to distinguish between lease offers. DHCP clients indicate which of several lease offers is being accepted by including this option in a DHCPREQUEST message.

The identifier is the IP address of the selected server.

The code for this option is 54, and its length is 4.

```
 Code   Len          Address
+-----+-----+-----+------+-----+-----+
|  54 |  4  |  a1 |  a2 |  a3 |  a4 |
+-----+-----+-----+------+-----+-----+
```

9.6. Parameter Request List

This option is used by a DHCP client to request values for specified configuration parameters. The list of requested parameters is specified as n octets, where each octet is a valid DHCP option code as defined in this document.

The client MAY list the options in order of preference. The DHCP server is not required to return the options in the requested order, but MUST try to insert the requested options in the order requested by the client.

The code for this option is 55. Its minimum length is 1.

```
 Code   Len   Option Codes
+-----+-----+-----+-----+---
|  55 |  n  |  c1 |  c2 | ...
+-----+-----+-----+-----+---
```

9.7. Message

This option is used by a DHCP server to provide an error message to a DHCP client in a DHCPNAK message in the event of a failure. A client may use this option in a DHCPDECLINE message to indicate the why the client declined the offered parameters. The message consists of n octets of NVT ASCII text, which the client may display on an available output device.

The code for this option is 56 and its minimum length is 1.

```
 Code   Len    Text
+-----+-----+-----+-----+---
|  56 |  n  |  c1 |  c2 | ...
+-----+-----+-----+-----+---
```

9.8. Maximum DHCP Message Size

This option specifies the maximum length DHCP message that it is willing to accept. The length is specified as an unsigned 16-bit integer. A client may use the maximum DHCP message size option in DHCPDISCOVER or DHCPREQUEST messages, but should not use the option in DHCPDECLINE messages.

The code for this option is 57, and its length is 2. The minimum legal value is 576 octets.

```
 Code   Len    Length
+-----+-----+-----+-----+
|  57 |  2  |  l1 |  l2 |
+-----+-----+-----+-----+
```

9.9. Renewal (T1) Time Value

This option specifies the time interval from address assignment until the client transitions to the RENEWING state.

The value is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 58, and its length is 4.

```
  Code   Len        T1 Interval
 +-----+-----+-----+-----+-----+-----+
 |  58 |  4  |  t1 |  t2 |  t3 |  t4 |
 +-----+-----+-----+-----+-----+-----+
```

9.10. Rebinding (T2) Time Value

This option specifies the time interval from address assignment until the client transitions to the REBINDING state.

The value is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 59, and its length is 4.

```
  Code   Len        T2 Interval
 +-----+-----+-----+-----+-----+-----+
 |  59 |  4  |  t1 |  t2 |  t3 |  t4 |
 +-----+-----+-----+-----+-----+-----+
```

9.11. Class-identifier

This option is used by DHCP clients to optionally identify the type and configuration of a DHCP client. The information is a string of n octets, interpreted by servers. Vendors and sites may choose to define specific class identifiers to convey particular configuration or other identification information about a client. For example, the identifier may encode the client's hardware configuration. Servers not equipped to interpret the class-specific information sent by a client MUST ignore it (although it may be reported).

The code for this option is 60, and its minimum length is 1.

```
 Code   Len   Class-Identifier
 +-----+-----+-----+-----+---
 |  60 |  n  |  i1 |  i2 | ...
 +-----+-----+-----+-----+---
```

9.12. Client-identifier

This option is used by DHCP clients to specify their unique identifier. DHCP servers use this value to index their database of address bindings. This value is expected to be unique for all clients in an administrative domain.

Identifiers consist of a type-value pair, similar to the It is expected that this field will typically contain a hardware type and hardware address, but this is not required. Current legal values for hardware types are defined in [22].

The code for this option is 61, and its minimum length is 2.

```
Code   Len   Type  Client-Identifier
+-----+-----+-----+-----+-----+---
|  61 |  n  |  t1 |  i1  |  i2 | ...
+-----+-----+-----+-----+-----+---
```

DHCP messages

Message	Use
DHCPDISCOVER	Client broadcast to locate available servers.
DHCPOFFER	Server to client in response to DHCPDISCOVER with offer of configuration parameters.
DHCPREQUEST	Client message to servers either (a) requesting offered parameters from one server and implicitly declining offers from all others, (b) confirming correctness of previously allocated address after, e.g., system reboot, or (c) extending the lease on a particular network address.
DHCPACK	Server to client with configuration parameters, including committed network address.
DHCPNAK	Server to client indicating client's notion of network address is incorrect (e.g., client has moved to new subnet) or client's lease as expired
DHCPDECLINE	Client to server indicating network address is already in use.
DHCPRELEASE	Client to server relinquishing network address and cancelling remaining lease.
DHCPINFORM	Client to server, asking only for local configuration parameters; client already has externally configured network address.

Domain names

The Open routine (ND_OPEN) splits the supplied parameters into two parts at the colon ':'

The right hand side is converted into a port number by OPN_DECODE_PORT. This routine checks to see if a number, or a service name has been supplied. If a service name has been supplied, it uses the services table to convert it into a porn number.

The left hand side is converted into a long word IP address by OPN_DECODE_ADDRESS. This routine checks see if an IP address, or a domain name has been supplied. If a domain name has been supplied, it first checks the DNS cache with CACHE_NAME to see if the IP address is already known. Otherwise it sends a DNS request with DO_DNS.

If DO_DNS obtains an IP address, then OPN_DECODE_ADDRESS will add it to the DNS cache.

DNS Cache

The DNS cache is a linked list of known domain names/IP addresses. Stored in a self expanding user heap. Entries in this list that reach their expiry dates, are removed when the list is scanned. And the scan encounters an expired entry. If the scan succeeds before the expired entry is found, then the entry is not removed. So expired entries may remain in the DNS cache for some time.

DNS cache linked list

$00	dcache_base	8 byte user heap header, do not use
$08	dcache_next	next link
$0C	dcache_ttl	time to live - the expiry date in QL format
$10	dcache_ip	IP address
$14	dcache_nlen	length of domain name
$16	dcache_name	domain name (length varies)

DNS Requests

The reply returned from a DNS request comprises a header (see section 4 below).
The original question query.
A number of RR records (see section 3 below).

The following text is taken from the RFC1035 document.

3.2. RR definitions

3.2.1. Format

All RRs have the same top level format shown below:

```
                               1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

NAME	an owner name, i.e., the name of the node to which this resource record pertains.
TYPE	two octets containing one of the RR TYPE codes.
CLASS	two octets containing one of the RR CLASS codes.
TTL	a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.
RDLENGTH	an unsigned 16 bit integer that specifies the length in octets of the RDATA field.
RDATA	a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.

3.2.2. TYPE values

TYPE fields are used in resource records. Note that these types are a subset of QTYPEs.

TYPE	value and meaning	
A	1	a host address
NS	2	an authoritative name server
MD	3	a mail destination (Obsolete - use MX)
MF	4	a mail forwarder (Obsolete - use MX)
CNAME	5	the canonical name for an alias
SOA	6	marks the start of a zone of authority
MB	7	a mailbox domain name (EXPERIMENTAL)
MG	8	a mail group member (EXPERIMENTAL)
MR	9	a mail rename domain name (EXPERIMENTAL)
NULL	10	a null RR (EXPERIMENTAL)
WKS	11	a well known service description
PTR	12	a domain name pointer
HINFO	13	host information
MINFO	14	mailbox or mail list information
MX	15	mail exchange
TXT	16	text strings

3.2.3. QTYPE values

QTYPE fields appear in the question part of a query. QTYPES are a superset of TYPEs, hence all TYPEs are valid QTYPEs. In addition, the following QTYPEs are defined:

AXFR	252	A request for a transfer of an entire zone
MAILB	253	A request for mailbox-related records (MB, MG or MR)
MAILA	254	A request for mail agent RRs (Obsolete - see MX)
*	255	A request for all records

3.2.4. CLASS values

CLASS fields appear in resource records. The following CLASS mnemonics and values are defined:

IN	1	the Internet
CS	2	the CSNET class (Obsolete - used only for examples in some obsolete RFCs)
CH	3	the CHAOS class
HS	4	Hesiod [Dyer 87]

3.2.5. QCLASS values

QCLASS fields appear in the question section of a query. QCLASS values are a superset of CLASS values; every CLASS is a valid QCLASS. In addition to CLASS values, the following QCLASSes are defined:

*	255 any class

3.3. Standard RRs

The following RR definitions are expected to occur, at least potentially, in all classes. In particular, NS, SOA, CNAME, and PTR will be used in all classes, and have the same format in all classes.
Because their RDATA format is known, all domain names in the RDATA section of these RRs may be compressed.

<domain-name> is a domain name represented as a series of labels, and terminated by a label with zero length. <character-string> is a single length octet followed by that number of characters. <character-string> is treated as binary information, and can be up to 256 characters in length (including the length octet).

3.3.1. CNAME RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     CNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

CNAME A <domain-name> which specifies the canonical or primary name for the owner. The owner name is an alias.

CNAME RRs cause no additional section processing, but name servers may choose to restart the query at the canonical name in certain cases. See the description of name server logic in [RFC-1034] for details.

3.3.2. HINFO RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                      CPU                      /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                       OS                      /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

CPU A <character-string> which specifies the CPU type.
OS A <character-string> which specifies the operating system type.

Standard values for CPU and OS can be found in [RFC-1010].

HINFO records are used to acquire general information about a host. The main use is for protocols such as FTP that can use special procedures when talking between machines or operating systems of the same type.

3.3.3. MB RDATA format (EXPERIMENTAL)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     MADNAME                   /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

MADNAME A <domain-name> which specifies a host which has the specified mailbox.

MB records cause additional section processing which looks up an A type RRs corresponding to MADNAME.

3.3.4. MD RDATA format (Obsolete)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     MADNAME                   /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which should be able to deliver mail for the domain.

MD records cause additional section processing which looks up an A type record corresponding to MADNAME.

MD is obsolete. See the definition of MX and [RFC-974] for details of the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 0.

3.3.5. MF RDATA format (Obsolete)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     MADNAME                   /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which will accept mail for forwarding to the domain.

MF records cause additional section processing which looks up an A type record corresponding to MADNAME.

MF is obsolete. See the definition of MX and [RFC-974] for details ofw the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 10.

3.3.6. MG RDATA format (EXPERIMENTAL)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     MGMNAME                     /
/                                                 /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

MGMNAME A <domain-name> which specifies a mailbox which is a member of the mail group specified by the domain name.

MG records cause no additional section processing.

3.3.7. MINFO RDATA format (EXPERIMENTAL)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     RMAILBX                     /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     EMAILBX                     /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

RMAILBX A <domain-name> which specifies a mailbox which is responsible for the mailing list or mailbox. If this domain name names the root, the owner of the MINFO RR is responsible for itself. Note that many existing mailing lists use a mailbox X-request for the RMAILBX field of mailing list X, e.g., Msgroup-request for Msgroup. This field provides a more general mechanism.

EMAILBX A <domain-name> which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR (similar to the ERRORS-TO: field which has been proposed). If this domain name names the root, errors should be returned to the sender of the message.

MINFO records cause no additional section processing. Although these records can be associated with a simple mailbox, they are usually used with a mailing list.

3.3.8. MR RDATA format (EXPERIMENTAL)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     NEWNAME                     /
/                                                 /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

NEWNAME A <domain-name> which specifies a mailbox which is the proper rename of the specified mailbox.

MR records cause no additional section processing. The main use for MR is as a forwarding entry for a user who has moved to a different mailbox.

3.3.9. MX RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                  PREFERENCE                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                   EXCHANGE                    /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

PREFERENCE A 16 bit integer which specifies the preference given to this RR among others at the same owner. Lower values are preferred.

EXCHANGE A <domain-name> which specifies a host willing to act as a mail exchange for the owner name.

MX records cause type A additional section processing for the host specified by EXCHANGE. The use of MX RRs is explained in detail in [RFC-974].

3.3.10. NULL RDATA format (EXPERIMENTAL)

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                  <anything>                   /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Anything at all may be in the RDATA field so long as it is 65535 octets or less.

NULL records cause no additional section processing. NULL RRs are not allowed in master files. NULLs are used as placeholders in some experimental extensions of the DNS.

3.3.11. NS RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                   NSDNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

NSDNAME A <domain-name> which specifies a host which should be authoritative for the specified class and domain.

NS records cause both the usual additional section processing to locate a type A record, and, when used in a referral, a special search of the zone in which they reside for glue information.

The NS RR states that the named host should be expected to have a zone starting at owner name of the specified class. Note that the class may not indicate the protocol family which should be used to communicate with the host, although it is typically a strong hint. For example, hosts which are name servers for either Internet (IN) or Hesiod (HS) class information are normally queried using IN class protocols.

3.3.12. PTR RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                   PTRDNAME                    /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

PTRDNAME A <domain-name> which points to some location in the domain name space.

PTR records cause no additional section processing. These RRs are used in special domains to point to some other location in the domain space. These records are simple data, and don't imply any special processing similar to that performed by CNAME, which identifies aliases. See the description of the IN-ADDR.ARPA domain for an example.

3.3.13. SOA RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     MNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     RNAME                     /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    SERIAL                     |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    REFRESH                    |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     RETRY                     |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    EXPIRE                     |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    MINIMUM                    |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

MNAME	The <domain-name> of the name server that was the original or primary source of data for this zone.
RNAME	A <domain-name> which specifies the mailbox of the person responsible for this zone.
SERIAL	The unsigned 32 bit version number of the original copy of the zone. Zone transfers preserve this value. This value wraps and should be compared using sequence space arithmetic.
REFRESH	A 32 bit time interval before the zone should be refreshed.
RETRY	A 32 bit time interval that should elapse before a failed refresh should be retried.
EXPIRE	A 32 bit time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.
MINIMUM	The unsigned 32 bit minimum TTL field that should be exported with any RR from this zone.

SOA records cause no additional section processing.

All times are in units of seconds.

Most of these fields are pertinent only for name server maintenance operations. However, MINIMUM is used in all query operations that retrieve RRs from a zone. Whenever a RR is sent in a response to a query, the TTL field is set to the maximum of the TTL field from the RR and the MINIMUM field in the appropriate SOA. Thus MINIMUM is a lower bound on the TTL field for all RRs in a zone. Note that this use of MINIMUM should occur when the RRs are copied into the response and not when the zone is loaded from a master file or via a zone transfer. The reason for this provison is to allow future dynamic update facilities to change the SOA RR with known semantics.

3.3.14. TXT RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                   TXT-DATA                    /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

TXT-DATA One or more <character-string>s.

TXT RRs are used to hold descriptive text. The semantics of the text depends on the domain where it is found.

3.4. Internet specific RRs

3.4.1. A RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ADDRESS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

ADDRESS A 32 bit Internet address.

Hosts that have multiple Internet addresses will have multiple A records.

A records cause no additional section processing. The RDATA section of an A line in a master file is an Internet address expressed as four decimal numbers separated by dots without any imbedded spaces (e.g., "10.2.0.52" or "192.0.5.6").

3.4.2. WKS RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ADDRESS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|       PROTOCOL        |                       |
+--+--+--+--+--+--+--+--+                       |
|                                               |
/                   <BIT MAP>                   /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

ADDRESS	An 32 bit Internet address
PROTOCOL	An 8 bit IP protocol number
<BIT MAP>	A variable length bit map. The bit map must be a multiple of 8 bits long.

The WKS record is used to describe the well known services supported by a particular protocol on a particular internet address. The PROTOCOL field specifies an IP protocol number, and the bit map has one bit per port of the specified protocol. The first bit corresponds to port 0, the second to port 1, etc. If the bit map does not include a bit for a protocol of interest, that bit is assumed zero. The appropriate values and mnemonics for ports and protocols are specified in [RFC-1010].

For example, if PROTOCOL=TCP (6), the 26th bit corresponds to TCP port 25 (SMTP). If this bit is set, a SMTP server should be listening on TCP port 25; if zero, SMTP service is not supported on the specified address.

The purpose of WKS RRs is to provide availability information for servers for TCP and UDP. If a server supports both TCP and UDP, or has multiple Internet addresses, then multiple WKS RRs are used.

WKS RRs cause no additional section processing.

In master files, both ports and protocols are expressed using mnemonics or decimal numbers.

3.5. IN-ADDR.ARPA domain

The Internet uses a special domain to support gateway location and Internet address to host mapping. Other classes may employ a similar strategy in other domains. The intent of this domain is to provide a guaranteed method to perform host address to host name mapping, and to facilitate queries to locate all gateways on a particular network in the Internet.

Note that both of these services are similar to functions that could be performed by inverse queries; the difference is that this part of the domain name space is structured according to address, and hence can guarantee that the appropriate data can be located without an exhaustive search of the domain space.

The domain begins at IN-ADDR.ARPA and has a substructure which follows the Internet addressing structure.

Domain names in the IN-ADDR.ARPA domain are defined to have up to four labels in addition to the IN-ADDR.ARPA suffix. Each label represents one octet of an Internet address, and is expressed as a character string for a decimal value in the range 0-255 (with leading zeros omitted except in the case of a zero octet which is represented by a single zero).

Host addresses are represented by domain names that have all four labels specified. Thus data for Internet address 10.2.0.52 is located at domain name 52.0.2.10.IN-ADDR.ARPA. The reversal, though awkward to read, allows zones to be delegated which are exactly one network of address space. For example, 10.IN-ADDR.ARPA can be a zone containing data for the ARPANET, while 26.IN-ADDR.ARPA can be a separate zone for MILNET. Address nodes are used to hold pointers to primary host names in the normal domain space.

Network numbers correspond to some non-terminal nodes at various depths in the IN-ADDR.ARPA domain, since Internet network numbers are either 1, 2, or 3 octets. Network nodes are used to hold pointers to the primary host names of gateways attached to that network. Since a gateway is, by definition, on more than one network, it will typically have two or more network nodes which point at it. Gateways will also have host level pointers at their fully qualified addresses.

Both the gateway pointers at network nodes and the normal host pointers at full address nodes use the PTR RR to point back to the primary domain names of the corresponding hosts.

For example, the IN-ADDR.ARPA domain will contain information about the ISI gateway between net 10 and 26, an MIT gateway from net 10 to MIT's net 18, and hosts A.ISI.EDU and MULTICS.MIT.EDU. Assuming that ISI gateway has addresses 10.2.0.22 and 26.0.0.103, and a name MILNET-GW.ISI.EDU, and the MIT gateway has addresses 10.0.0.77 and 18.10.0.4 and a name GW.LCS.MIT.EDU, the domain database would contain:

```
10.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
18.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
26.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
22.0.2.10.IN-ADDR.ARPA.     PTR MILNET-GW.ISI.EDU.
103.0.0.26.IN-ADDR.ARPA.    PTR MILNET-GW.ISI.EDU.
77.0.0.10.IN-ADDR.ARPA.     PTR GW.LCS.MIT.EDU.
4.0.10.18.IN-ADDR.ARPA.     PTR GW.LCS.MIT.EDU.
103.0.3.26.IN-ADDR.ARPA.    PTR A.ISI.EDU.
6.0.0.10.IN-ADDR.ARPA.      PTR MULTICS.MIT.EDU.
```

Thus a program which wanted to locate gateways on net 10 would originate a query of the form QTYPE=PTR, QCLASS=IN, QNAME=10.IN-ADDR.ARPA. It would receive two RRs in response:

```
10.IN-ADDR.ARPA.            PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.            PTR GW.LCS.MIT.EDU.
```

The program could then originate QTYPE=A, QCLASS=IN queries for MILNET-GW.ISI.EDU. and GW.LCS.MIT.EDU. to discover the Internet addresses of these gateways.

A resolver which wanted to find the host name corresponding to Internet host address 10.0.0.6 would pursue a query of the form QTYPE=PTR, QCLASS=IN, QNAME=6.0.0.10.IN-ADDR.ARPA, and would receive:

```
6.0.0.10.IN-ADDR.ARPA.      PTR MULTICS.MIT.EDU.
```

Several cautions apply to the use of these services:

- Since the IN-ADDR.ARPA special domain and the normal domain for a particular host or gateway will be in different zones, the possibility exists that that the data may be inconsistent.

- Gateways will often have two names in separate domains, only one of which can be primary.

- Systems that use the domain database to initialize their routing tables must start with enough gateway information to guarantee that they can access the appropriate name server.

- The gateway data only reflects the existence of a gateway in a manner equivalent to the current HOSTS.TXT file. It doesn't replace the dynamic availability information from GGP or EGP.

3.6. Defining new types, classes, and special namespaces

The previously defined types and classes are the ones in use as of the date of this memo. New definitions should be expected. This section makes some recommendations to designers considering additions to the existing facilities. The mailing list NAMEDROPPERS@SRI-NIC.ARPA is the forum where general discussion of design issues takes place.

In general, a new type is appropriate when new information is to be added to the database about an existing object, or we need new data formats for some totally new object. Designers should attempt to define types and their RDATA formats that are generally applicable to all classes, and which avoid duplication of information. New classes are appropriate when the DNS is to be used for a new protocol, etc which requires new class-specific data formats, or when a copy of the existing name space is desired, but a separate management domain is necessary.

New types and classes need mnemonics for master files; the format of the master files requires that the mnemonics for type and class be disjoint.

TYPE and CLASS values must be a proper subset of QTYPEs and QCLASSes respectively.

The present system uses multiple RRs to represent multiple values of a type rather than storing multiple values in the RDATA section of a single RR. This is less efficient for most applications, but does keep RRs shorter. The multiple RRs assumption is incorporated in some experimental work on dynamic update methods.

The present system attempts to minimize the duplication of data in the database in order to insure consistency. Thus, in order to find the address of the host for a mail exchange, you map the mail domain name to a host name, then the host name to addresses, rather than a direct mapping to host address. This approach is preferred because it avoids the opportunity for inconsistency.

In defining a new type of data, multiple RR types should not be used to create an ordering between entries or express different formats for equivalent bindings, instead this information should be carried in the body of the RR and a single type used. This policy avoids problems with caching multiple types and defining QTYPEs to match multiple types.

For example, the original form of mail exchange binding used two RR types one to represent a "closer" exchange (MD) and one to represent a "less close" exchange (MF). The difficulty is that the presence of one RR type in a cache doesn't convey any information about the other because the query which acquired the cached information might have used a QTYPE of MF, MD, or MAILA (which matched both). The redesigned service used a single type (MX) with a "preference" value in the RDATA section which can order different RRs. However, if any MX RRs are found in the cache, then all should be there.

4. MESSAGES

4.1. Format

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:

```
+---------------------+
|        Header       |
+---------------------+
|       Question      |   the question for the name server
+---------------------+
|        Answer       |   RRs answering the question
+---------------------+
|      Authority      |   RRs pointing toward an authority
+---------------------+
|      Additional     |   RRs holding additional information
+---------------------+
```

The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query or a response, a standard query or some other opcode, etc.

The names of the sections after the header are derived from their use in standard queries. The question section contains fields that describe a question to a name server. These fields are a query type (QTYPE), a query class (QCLASS), and a query domain name (QNAME). The last three sections have the same format: a possibly empty list of concatenated resource records (RRs). The answer section contains RRs that answer the question; the authority section contains RRs that point toward an authoritative name server; the additional records section contains RRs which relate to the query, but are not strictly answers for the question.

4.1.1. Header section format

The header contains the following fields:

```
                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      ID                       |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    QDCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ANCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    NSCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ARCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

ID	A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.
QR	A one bit field that specifies whether this message is a query (0), or a response (1).
OPCODE	A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are: 0 a standard query (QUERY) 1 an inverse query (IQUERY) 2 a server status request (STATUS) 3-15 reserved for future use
AA	Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section. Note that the contents of the answer section may have multiple owner names because of aliases. The AA bit corresponds to the name which matches the query name, or the first owner name in the answer section.
TC	TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
RD	Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.
RA	Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.
Z	Reserved for future use. Must be zero in all queries and responses.
RCODE	Response code - this 4 bit field is set as part of responses. The values have the following interpretation: 0 No error condition 1 Format error - The name server was unable to interpret the query. 2 Server failure - The name server was unable to process this query due to a problem with the name server. 3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist. 4 Not Implemented - The name server does not support the requested kind of query. 5 Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data. 6-15 Reserved for future use.
QDCOUNT	an unsigned 16 bit integer specifying the number of entries in the question section.

ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section.

4.1.2. Question section format

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:

```
                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                                               |
    /                     QNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     QTYPE                     |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     QCLASS                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

QNAME a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

QTYPE a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

QCLASS a two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.

4.1.3. Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                     NAME                      /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     TYPE                      |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

NAME a domain name to which this resource record pertains.

TYPE two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.

CLASS two octets which specify the class of the data in the RDATA field.

TTL a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

4.1.4. Message compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurrence of the same name.

The pointer takes the form of a two octet sequence:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 1  1|                OFFSET                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

The compression scheme allows a domain name in a message to be represented as either:

- a sequence of labels ending in a zero octet

- a pointer

- a sequence of labels ending with a pointer

Pointers can only be used for occurrences of a domain name where the format is not class specific. If this were not the case, a name server or resolver would be required to know the format of all RRs it handled. As yet, there are no such cases, but they may occur in future RDATA formats.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of an RR), and compression is used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Programs are free to avoid using pointers in messages they generate, although this will reduce datagram capacity, and may cause truncation. However all programs are required to understand arriving messages that contain pointers.

**** This is the NAME field
Offset is from the start of the transaction ID field. To the start of some labels etc ****

For example, a datagram might need to use the domain names F.ISI.ARPA, FOO.F.ISI.ARPA, ARPA, and the root. Ignoring the other fields of the message, these domain names might be represented as:

```
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
20     |           1           |           F           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
22     |           3           |           I           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
24     |           S           |           I           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
26     |           4           |           A           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
28     |           R           |           P           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
30     |           A           |           0           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
40     |           3           |           F           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
42     |           O           |           O           |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
44     | 1  1|                20                       |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
64     | 1  1|                26                       |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
92     |           0           |                       |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The domain name for F.ISI.ARPA is shown at offset 20. The domain name FOO.F.ISI.ARPA is shown at offset 40; this definition uses a pointer to concatenate a label for FOO to the previously defined F.ISI.ARPA. The domain name ARPA is defined at offset 64 using a pointer to the ARPA component of the name F.ISI.ARPA at 20; note that this pointer relies on ARPA being the last label in the string at 20. The root domain name is defined by a single octet of zeros at 92; the root domain name has no labels.

4.2. Transport

The DNS assumes that messages will be transmitted as datagrams or in a byte stream carried by a virtual circuit. While virtual circuits can be used for any DNS activity, datagrams are preferred for queries due to their lower overhead and better performance. Zone refresh activities must use virtual circuits because of the need for reliable transfer.

The Internet supports name server access using TCP [RFC-793] on server port 53 (decimal) as well as datagram access using UDP [RFC-768] on UDP port 53 (decimal).

4.2.1. UDP usage

Messages sent using UDP user server port 53 (decimal).

Messages carried by UDP are restricted to 512 bytes (not counting the IP or UDP headers). Longer messages are truncated and the TC bit is set in the header.

UDP is not acceptable for zone transfers, but is the recommended method for standard queries in the Internet. Queries sent using UDP may be lost, and hence a retransmission strategy is required. Queries or their responses may be reordered by the network, or by processing in name servers, so resolvers should not depend on them being returned in order.

The optimal UDP retransmission policy will vary with performance of the Internet and the needs of the client, but the following are recommended:

- The client should try other servers and server addresses before repeating a query to a specific address of a server.

- The retransmission interval should be based on prior statistics if possible. Too aggressive retransmission can easily slow responses for the community at large. Depending on how well connected the client is to its expected servers, the minimum retransmission interval should be 2-5 seconds.

More suggestions on server selection and retransmission policy can be found in the resolver section of this memo.

4.2.2. TCP usage

Messages sent over TCP connections use server port 53 (decimal). The message is prefixed with a two byte length field which gives the message length, excluding the two byte length field. This length field allows the low-level processing to assemble a complete message before beginning to parse it.

Several connection management policies are recommended:

- The server should not block other activities waiting for TCP data.

- The server should support multiple connections.

- The server should assume that the client will initiate connection closing, and should delay closing its end of the connection until all outstanding client requests have been satisfied.

- If the server needs to close a dormant connection to reclaim resources, it should wait until the connection has been idle for a period on the order of two minutes. In particular, the server should allow the SOA and AXFR request sequence (which begins a refresh operation) to be made on a single connection. Since the server would be unable to answer queries anyway, a unilateral close or reset may be used instead of a graceful close.

IP Error messages

Receive errors

iperr.rxomem	1	Insufficient memory to read packet into
iperr.rxcsum	2	Read packet validation failed, checksum mismatch
iperr.badmac	3	Received packet from unexpected MAC address
iperr.ctrl	4	Unable to process TCP control bits
iperr.segerr	5	Unexpected TCP segment numbers
iperr.seqerr	6	Unexpected TCP sequence number

Send errors

iperr.txerr	8	Last packet was not transmitted successfully
iperr.txnogo	10	Timeout waiting for transmit buffer to empty
iperr.txprot	11	Protocol not found when creating a packet
iperr.txarpf	12	Failure sending ARP request
iperr.txpingf	13	Failure sending Ping reply
iperr.txnoak	14	Attempt to send too many packets with received ACK's (TCP)
iperr.txSYAC	15	Failure sending a SYN,ACK
iperr.maxt	16	Too many attempts made to send a packet

Open errors

iperr.inparm	20	Invalid parameters for requested OPEN type
iperr.noport	21	No managed ports available
iperr.nomac	22	Unable to squire MAC address for specified IP address
iperr.portna	23	Requested port already in use
iperr.noserv	24	No response from requested TCP server for a SYN request
iperr.badport	25	Bad port number, or service name supplied
iperr.badip	26	Bad IP address, or Domain name supplied
iperr.nocolon	27	No colon found in supplied parameters

I/O errors

iperr.txcsum	30	Error creating checksum for transmission
iperr.toobig	31	Attempt to send more than 64K bytes
iperr.scklen	32	Bad sockaddr length
iperr.fragto	33	Fragmented packet incomplete before life ran out
iperr.macto	34	I/O timeout while waiting for a MAC address
iperr.outrng	35	IP_LISTEN queue out of range
iperr.nottcp	36	Not a TCP channel
iperr.lsomem	37	Out of memory creating a dummy channel definition block
iperr.lqfull	38	The queue for a listening channel is full

Close errors

iperr.closto	40	Timed out waiting for a TCP close acknowledgement
iperr.wait1	41	While in FIN-WAIT-1, Timed out waiting for an ACK
iperr.wait2	42	While in FIN-WAIT-2, Timed out waiting for a FIN (last ACK)
iperr.wait3	43	While in LAST-ACK, Timed out waiting for final ACK
iperr.uxfin	44	Unexpected FIN received
iperr.wait4	45	While in CLOSING, Timed out waiting for final ACK
iperr.nofin	46	Failed sending a FIN

IP Trap errors

iperr.dbrec	50	Error reading a database entry

DHCP errors

iperr.dhbreak	60	BREAK pressed during DHCP
iperr.dhto	61	DHCP timed out
iperr.dhlend	62	DHCP lease ran out

DNS errors

iperr.nodns	70	No DNS server setup
iperr.dnsto	71	Timed out waiting for answer from DNS server

CP2200 Initialization errors

iperr.inito	100	Timed out while waiting for controller to reset
iperr.aufail	110	Timed out while waiting for auto-neg in Physical layer