ETHER_ARP

The ETHER_ARP module implements the ethernet ARP protocol and completes the ethernet headers for frames sent by the IP layer. Usually, there will be a single ‘earp’ process in the system, controlling a number of ethernet interfaces. The mapping between interface numbers at the earp layer and physical devices is soft, and controlled by the configuration strings passed to the process at startup.

Process Information

Prototype Name	earp
Link Order	before any PCI or ATA bus drivers
Process Name	must match the configuration strings used in the IP layer.

Module Options

EARP_P_TRACE	If this symbol is defined, the MAC addresses for machines are displayed as they are resolved. Setting this option will also cause the Finite State Machine symbols to be generated.

Configuration Commands

The EARP process is configured using two strings passed to it as *COMMAND* messages.

config

config *ifno i1.i2.i3.i4 m1.m2.m3.m4*

The *config* command sets the IP address and netmask for the interface numbered *ifno*. The netmask is used to determine the address range of the interface, and so how many ARP table entries to allocate for this interface. This command can only be issued after the corresponding *interface* command. The default broadcast address is set to the highest address possible in the netmask range.

Example: “config 0 138.15.103.240 255.255.255.0”

interface

interface *ifno devname devindex*

The *interface* command associates an interface at the earp layer with a physical device. The device is specified by a process name, *devname*, and an index within that process, *devindex*, passed in the *scheme* and *port* fields of the open URL respectively. When the interface is defined, a local file is opened to the device for the ethernet ARP protocol, and this file is used to read the local MAC address from the device. This command must be issued before the interface can be configured.

Example: “interface 0 ether 0”

Process Operation

The operation of the EARP process can be divided into the STREAMS-style messages, which are examined in order to have the correct ethernet addresses placed in the headers, and the ARP protocol processing which terminates at this layer. Internally, these two levels of operation are distinguished by the *src_context* field in the messages. STREAMS messages have an upstream/downstream link in this field, whereas protocol messages have an interface number. This assumes that the addresses of contexts cannot be small positive integers.

The module has a main process and a queue handler. The description of the two different levels of processing will be described separately in the messaging below

STREAMS Processing

Because of the need to reserve space at the head of the buffer, it is assumed that the IP process always uses *NEWMBLK* to request buffers and never supplies a buffer that has insufficient space at the start for the ethernet header.

CLOSE messages are handled by the main process. The downstream link to the device is closed and the local data structure is freed before the reply is generated. Any messages queued on pending ARP requests will still be sent, or returned upstream.

COMMAND messages are handled in the main process according to the ‘Configuration’ section above.

EVENT messages are passed down to the underlying device from the queue handler, and the replies passed upwards towards the application.

FETMBLK/GETMBLK messages are passed down to the underlying device from the queue handler. The replies are passed upwards towards the application after moving the read pointer past the ethernet header.

FLUSH messages are passed transparently through the process. It is not expected that the IP layer, which operates in datagram mode, will issue *FLUSH* requests to this process.

NEWMBLK messages are passed down to the underlying device from the queue handler after increasing the request size to allow for the ethernet header. The read pointer is adjusted on the reply to reserve the header space.

OPEN messages are handled in the main process. The *port* field of the URL contains the earp interface number and the *ip_port* field contains the ethernet protocol selector. The interface number is used to construct the corresponding URL to open the device and the queue structure is initialised for up/down-stream processing.

OUTMBLK/PUTMBLK messages are processed in the queue handler. If the message has non-zero length, the read pointer is decremented to expose the space for the ethernet header. If the packet is of type

M_IPDATA then the *b_immed* field is assumed to contain the destination IP address, which is used to complete the ethernet header. otherwise it is assumed that the application has completed the fields itself. If the destination IP address is not in the IP cache, the message is held and the ARP protocol initiated to resolve the address within the main process, otherwise the message is passed downstream to the device. Replies are passed back upstream, after putting the read pointer back after the ethernet header (to allow for re-transmissions).

RETMBLK messages are passed down to the underlying device from the queue handler, and the replies passed upwards towards the application.

Protocol Processing

When an IP address is not in the ARP cache, the message is passed into the main process to start ARP protocol processing. The protocol is implemented as a small finite-state machine for each potential IP address. The main issue in processing the protocol is noting whether or not an IP address has a message waiting for resolution. Only one message is kept for each IP address, later messages replace earlier ones, which are returned to the sender. As the link is not reliable, each ARP request is re-transmitted on a timer, up to a maximum number of times.

	IDLE	PENDING	OK
LOOKUP	save requestsend requeststart timer->PENDING	return old messagesave new message	(ignored)
RESOLVED	set ARP entry->OK	set ARP entrysend saved message->OK	(ignored)
TIMEOUT	(does not occur)	too many retries?Y: return old message—–>IDLEN: (re)-send requeststart timer	(ignored)

In order that machines may change their MAC addresses, ARP entries time out slowly. Associated with each entry is an ‘epoch’ which changes every minute. Entries older than five minutes are removed from the cache, so forcing a new ARP request the next time they are used. Within the earp process, the messages are handled as follows:

EVENT messages are sent when the driver replies to the configuration request with an indication that it supports event notifications. Replies containing card insertion events cause the interface to be re-initialised.

GETMBLK messages are set up when the interface is configured. Replies contain either responses to outgoing ARP requests or requests generated by other machines. The process responds to requests for its IP address, and processes replies to its own requests according to the state machine above.

NEWMBLK messages are used to request buffers from the driver for protocol messages.

PUTMBLK messages are used to transmit ARP requests and responses to the driver. Even though a message may need to be re-transmitted, a new buffer is allocated each time, to avoid tying up driver resources.

RETMBLK messages are used to return received buffers after the contents have been processed.

TIMEOUT messages are used to control the re-transmission of ‘lost’ ARP requests according to the state machine above. Longer-interval timers are used to invalidate old ARP entries at the ‘epoch’ level.

Shared Library Macros and Routines

earp_clear_cache

void *earp_clear_cache*(
 int *ifno*)

The *earp_clear_cache* routine clears the ARP cache associated with the interface numbered *ifno*. Passing a value of -1 clears all the ARP caches.

Debug Support

The *earp_dump_cache* routine is callable from the debugger and prints the contents of all the ARP caches to the polled-mode I/O interface.