NAME

arp − Linux ARP kernel module.

DESCRIPTION

This kernel protocol module implements the Address Resolution Protocol defined in RFC 826. It is used to convert between Layer2 hardware addresses and IPv4 protocol addresses on directly connected networks. The user normally doesn't interact directly with this module except to configure it; instead it provides a service for other protocols in the kernel.

A user process can receive ARP packets by using **packet**(7) sockets. There is also a mechanism for managing the ARP cache in user-space by using **netlink**(7) sockets. The ARP table can also be controlled via **ioctl**(2) on any **AF_INET** socket.

The ARP module maintains a cache of mappings between hardware addresses and protocol addresses. The cache has a limited size so old and less frequently used entries are garbage-collected. Entries which are marked as permanent are never deleted by the garbage-collector. The cache can be directly manipulated by the use of ioctls and its behavior can be tuned by the */proc* interfaces described below.

When there is no positive feedback for an existing mapping after some time (see the */proc* interfaces below), a neighbor cache entry is considered stale. Positive feedback can be gotten from a higher layer; for example from a successful TCP ACK. Other protocols can signal forward progress using the **MSG_CONFIRM** flag to **sendmsg**(2). When there is no forward progress, ARP tries to reprobe. It first tries to ask a local arp daemon **app_solicit** times for an updated MAC address. If that fails and an old MAC address is known, a unicast probe is sent **ucast_solicit** times. If that fails too, it will broadcast a new ARP request to the network. Requests are sent only when there is data queued for sending.

Linux will automatically add a nonpermanent proxy arp entry when it receives a request for an address it forwards to and proxy arp is enabled on the receiving interface. When there is a reject route for the target, no proxy arp entry is added.

Ioctls

Three ioctls are available on all **AF_INET** sockets. They take a pointer to a *struct arpreq* as their argument.

```
struct arpreq {
    struct sockaddr arp_pa;      /* protocol address */
    struct sockaddr arp_ha;      /* hardware address */
    int             arp_flags;   /* flags */
    struct sockaddr arp_netmask; /* netmask of protocol address */
    char            arp_dev[16];
};
```

SIOCSARP, **SIOCDARP** and **SIOCGARP** respectively set, delete, and get an ARP mapping. Setting and deleting ARP maps are privileged operations and may be performed only by a process with the **CAP_NET_ADMIN** capability or an effective UID of 0.

arp_pa must be an **AF_INET** address and *arp_ha* must have the same type as the device which is specified in *arp_dev*. *arp_dev* is a zero-terminated string which names a device.

arp_flags	
flag	meaning
ATF_COM	Lookup complete
ATF_PERM	Permanent entry
ATF_PUBL	Publish entry
ATF_USETRAILERS	Trailers requested
ATF_NETMASK	Use a netmask
ATF_DONTPUB	Don't answer

If the **ATF_NETMASK** flag is set, then *arp_netmask* should be valid. Linux 2.2 does not support proxy network ARP entries, so this should be set to 0xffffffff, or 0 to remove an existing proxy arp entry.

ATF_USETRAILERS is obsolete and should not be used.

/proc interfaces

ARP supports a range of */proc* interfaces to configure parameters on a global or per-interface basis. The interfaces can be accessed by reading or writing the */proc/sys/net/ipv4/neigh/*/** files. Each interface in the system has its own directory in */proc/sys/net/ipv4/neigh/*. The setting in the "default" directory is used for all newly created devices. Unless otherwise specified, time-related interfaces are specified in seconds.

anycast_delay (since Linux 2.2)
: The maximum number of jiffies to delay before replying to a IPv6 neighbor solicitation message. Anycast support is not yet implemented. Defaults to 1 second.

app_solicit (since Linux 2.2)
: The maximum number of probes to send to the user space ARP daemon via netlink before dropping back to multicast probes (see *mcast_solicit*). Defaults to 0.

base_reachable_time (since Linux 2.2)
: Once a neighbor has been found, the entry is considered to be valid for at least a random value between *base_reachable_time*/2 and 3**base_reachable_time*/2. An entry's validity will be extended if it receives positive feedback from higher level protocols. Defaults to 30 seconds. This file is now obsolete in favor of *base_reachable_time_ms*.

base_reachable_time_ms (since Linux 2.6.12)
: As for *base_reachable_time*, but measures time in milliseconds. Defaults to 30000 milliseconds.

delay_first_probe_time (since Linux 2.2)
: Delay before first probe after it has been decided that a neighbor is stale. Defaults to 5 seconds.

gc_interval (since Linux 2.2)
: How frequently the garbage collector for neighbor entries should attempt to run. Defaults to 30 seconds.

gc_stale_time (since Linux 2.2)
: Determines how often to check for stale neighbor entries. When a neighbor entry is considered stale, it is resolved again before sending data to it. Defaults to 60 seconds.

gc_thresh1 (since Linux 2.2)
: The minimum number of entries to keep in the ARP cache. The garbage collector will not run if there are fewer than this number of entries in the cache. Defaults to 128.

gc_thresh2 (since Linux 2.2)
: The soft maximum number of entries to keep in the ARP cache. The garbage collector will allow the number of entries to exceed this for 5 seconds before collection will be performed. Defaults to 512.

gc_thresh3 (since Linux 2.2)
: The hard maximum number of entries to keep in the ARP cache. The garbage collector will always run if there are more than this number of entries in the cache. Defaults to 1024.

locktime (since Linux 2.2)
: The minimum number of jiffies to keep an ARP entry in the cache. This prevents ARP cache thrashing if there is more than one potential mapping (generally due to network misconfiguration). Defaults to 1 second.

mcast_solicit (since Linux 2.2)
: The maximum number of attempts to resolve an address by multicast/broadcast before marking the entry as unreachable. Defaults to 3.

proxy_delay (since Linux 2.2)
: When an ARP request for a known proxy-ARP address is received, delay up to *proxy_delay* jiffies before replying. This is used to prevent network flooding in some cases. Defaults to 0.8 seconds.

proxy_qlen (since Linux 2.2)
: The maximum number of packets which may be queued to proxy-ARP addresses. Defaults to 64.

retrans_time (since Linux 2.2)
: The number of jiffies to delay before retransmitting a request. Defaults to 1 second. This file is now obsolete in favor of *retrans_time_ms*.

retrans_time_ms (since Linux 2.6.12)
: The number of milliseconds to delay before retransmitting a request. Defaults to 1000 milliseconds.

ucast_solicit (since Linux 2.2)
: The maximum number of attempts to send unicast probes before asking the ARP daemon (see *app_solicit*). Defaults to 3.

unres_qlen (since Linux 2.2)
: The maximum number of packets which may be queued for each unresolved address by other network layers. Defaults to 3.

VERSIONS

The *struct arpreq* changed in Linux 2.0 to include the *arp_dev* member and the ioctl numbers changed at the same time. Support for the old ioctls was dropped in Linux 2.2.

Support for proxy arp entries for networks (netmask not equal 0xffffffff) was dropped in Linux 2.2. It is replaced by automatic proxy arp setup by the kernel for all reachable hosts on other interfaces (when forwarding and proxy arp is enabled for the interface).

The *neigh/** interfaces did not exist before Linux 2.2.

BUGS

Some timer settings are specified in jiffies, which is architecture- and kernel version-dependent; see **time**(7).

There is no way to signal positive feedback from user space. This means connection-oriented protocols implemented in user space will generate excessive ARP traffic, because ndisc will regularly reprobe the MAC address. The same problem applies for some kernel protocols (e.g., NFS over UDP).

This man page mashes together functionality that is IPv4-specific with functionality that is shared between IPv4 and IPv6.

SEE ALSO

capabilities(7), **ip**(7), **arpd**(8)

RFC 826 for a description of ARP. RFC 2461 for a description of IPv6 neighbor discovery and the base algorithms used. Linux 2.2+ IPv4 ARP uses the IPv6 algorithms when applicable.