NAME

inet6 - Internet protocol version 6 family

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

The **inet6** family is an updated version of inet(4) family. While inet(4) implements Internet Protocol version 4, **inet6** implements Internet Protocol version 6.

inet6 is a collection of protocols layered atop the *Internet Protocol version 6* (IPv6) transport layer, and utilizing the IPv6 address format. The **inet6** family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IPv6 protocol.

ADDRESSING

IPv6 addresses are 16 byte quantities, stored in network standard byteorder. The include file *<netinet/in.h>* defines this address as a discriminated union.

Sockets bound to the **inet6** family utilize the following addressing structure:

```
struct sockaddr_in6 {
        uint8_t         sin6_len;
        sa_family_t     sin6_family;
        in_port_t       sin6_port;
        uint32_t        sin6_flowinfo;
        struct in6_addr sin6_addr;
        uint32_t        sin6_scope_id;
};
```

Sockets may be created with the local address "::" (which is equal to IPv6 address 0:0:0:0:0:0:0:0) to effect "wildcard" matching on incoming messages.

The IPv6 specification defines scoped addresses, like link-local or site-local addresses. A scoped address is ambiguous to the kernel, if it is specified without a scope identifier. To manipulate scoped addresses properly from the userland, programs must use the advanced API defined in RFC2292. A compact description of the advanced API is available in ip6(4). If a scoped address is specified without an explicit scope, the kernel may raise an error. Note that scoped addresses are not for daily use at this moment, both from a specification and an implementation point of view.

The KAME implementation supports an extended numeric IPv6 address notation for link-local addresses, like "fe80::1%de0" to specify "fe80::1 on de0 interface". This notation is supported by getaddrinfo(3) and getnameinfo(3). Some of normal userland programs, such as telnet(1) or ftp(1), are able to use this notation. With special programs like ping(8), you can specify the outgoing interface by an extra command line option to disambiguate scoped addresses.

Scoped addresses are handled specially in the kernel. In kernel structures like routing tables or interface structures, a scoped address will have its interface index embedded into the address. Therefore, the address in some kernel structures is not the same as that on the wire. The embedded index will become visible through a PF_ROUTE socket, kernel memory accesses via kvm(3) and on some other occasions. HOWEVER, users should never use the embedded form. For details please consult *IMPLEMENTATION* supplied with KAME kit.

PROTOCOLS

The **inet6** family is comprised of the IPv6 network protocol, Internet Control Message Protocol version 6 (ICMPv6), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. Note that TCP and UDP are common to inet(4) and **inet6**. A raw interface to IPv6 is available by creating an Internet socket of type SOCK_RAW. The ICMPv6 message protocol is accessible from a raw socket.

MIB Variables

A number of variables are implemented in the *net.inet6* branch of the sysctl(3) MIB. In addition to the variables supported by the transport protocols (for which the respective manual pages may be consulted), the following general variables are defined:

IPV6CTL_FORWARDING	(ip6.forwarding) Boolean: enable/disable forwarding of IPv6 packets. Also, identify if the node is acting as a router. Defaults to off.
IPV6CTL_SENDREDIRECTS	(ip6.redirect) Boolean: enable/disable sending of ICMPv6 redirects in response to unforwardable IPv6 packets. This option is ignored unless the node is routing IPv6 packets, and should normally be enabled on all systems. Defaults to on.
IPV6CTL_DEFHLIM	(ip6.hlim) Integer: default hop limit value to use for outgoing IPv6 packets. This value applies to all the transport protocols on top of IPv6. There are APIs to override the value.
IPV6CTL_MAXFRAGS	(ip6.maxfrags) Integer: maximum number of fragments the host will

accept and simultaneously hold across all reassembly queues in all VNETs. If set to 0, fragment reassembly is disabled. If set to -1, this limit is not applied. This limit is recalculated when the number of mbuf clusters is changed. This is a global limit.

IPV6CTL_MAXFRAGPACKETS (ip6.maxfragpackets) Integer: maximum number of fragmented packets the node will accept and simultaneously hold in the reassembly queue for a particular VNET. 0 means that the node will not accept any fragmented packets for that VNET. -1 means that the node will not apply this limit for that VNET. This limit is recalculated when the number of mbuf clusters is changed. This is a per-VNET limit.

IPV6CTL_MAXFRAGBUCKETSIZE
(ip6.maxfragbucketsize) Integer: maximum number of reassembly queues per bucket. Fragmented packets are hashed to buckets. Each bucket has a list of reassembly queues. The system must compare the incoming packets to the existing reassembly queues in the bucket to find a matching reassembly queue. To preserve system resources, the system limits the number of reassembly queues allowed in each bucket. This limit is recalculated when the number of mbuf clusters is changed or when the value of *ip6.maxfragpackets* changes. This is a per-VNET limit.

IPV6CTL_MAXFRAGSPERPACKET
(ip6.maxfragsperpacket) Integer: maximum number of fragments the host will accept and hold in the ressembly queue for a packet. This is a per-VNET limit.

IPV6CTL_ACCEPT_RTADV (ip6.accept_rtadv) Boolean: the default value of a per-interface flag to enable/disable receiving of ICMPv6 router advertisement packets, and autoconfiguration of address prefixes and default routers. The node must be a host (not a router) for the option to be meaningful. Defaults to off.

IPV6CTL_AUTO_LINKLOCAL (ip6.auto_linklocal) Boolean: the default value of a per-interface flag to enable/disable performing automatic link-local address configuration. Defaults to on.

IPV6CTL_LOG_INTERVAL (ip6.log_interval) Integer: default interval between IPv6 packet

forwarding engine log output (in seconds).

IPV6CTL_HDRNESTLIMIT (ip6.hdrnestlimit) Integer: default number of the maximum IPv6 extension headers permitted on incoming IPv6 packets. If set to 0, the node will accept as many extension headers as possible.

IPV6CTL_DAD_COUNT (ip6.dad_count) Integer: default number of IPv6 DAD (duplicated address detection) probe packets. The packets will be generated when IPv6 interface addresses are configured.

IPV6CTL_AUTO_FLOWLABEL (ip6.auto_flowlabel) Boolean: enable/disable automatic filling of IPv6 flowlabel field, for outstanding connected transport protocol packets. The field might be used by intermediate routers to identify packet flows. Defaults to on.

IPV6CTL_DEFMCASTHLIM (ip6.defmcasthlim) Integer: default hop limit value for an IPv6 multicast packet sourced by the node. This value applies to all the transport protocols on top of IPv6. There are APIs to override the value as documented in ip6(4).

IPV6CTL_GIF_HLIM (ip6.gifhlim) Integer: default maximum hop limit value for an IPv6 packet generated by gif(4) tunnel interface.

IPV6CTL_KAME_VERSION (ip6.kame_version) String: identifies the version of KAME IPv6 stack implemented in the kernel.

IPV6CTL_USE_DEPRECATED (ip6.use_deprecated) Boolean: enable/disable use of deprecated address, specified in RFC2462 5.5.4. Defaults to on.

IPV6CTL_RR_PRUNE (ip6.rr_prune) Integer: default interval between IPv6 router renumbering prefix babysitting, in seconds.

IPV6CTL_V6ONLY (ip6.v6only) Boolean: enable/disable the prohibited use of IPv4 mapped address on AF_INET6 sockets. Defaults to on.

ip6.log_cannot_forward Boolean: log packets that can’t be forwarded because of unspecified source address or destination address beyond the scope of the source address as described in RFC4443. Enabled by default.

ip6.source_address_validation Boolean: perform source address validation for packets destined for

the local host. Consider this as following Section 3.2 of RFC3704/BCP84, where we treat local host as our own infrastructure. This has no effect on packets to be forwarded, so don't consider it as anti-spoof feature for a router. Enabled by default.

Interaction between IPv4/v6 sockets

By default, FreeBSD does not route IPv4 traffic to AF_INET6 sockets. The default behavior intentionally violates RFC2553 for security reasons. Listen to two sockets if you want to accept both IPv4 and IPv6 traffic. IPv4 traffic may be routed with certain per-socket/per-node configuration, however, it is not recommended to do so. Consult ip6(4) for details.

The behavior of AF_INET6 TCP/UDP socket is documented in RFC2553. Basically, it says this:

- A specific bind on an AF_INET6 socket (bind(2) with an address specified) should accept IPv6 traffic to that address only.
- If you perform a wildcard bind on an AF_INET6 socket (bind(2) to IPv6 address ::), and there is no wildcard bind AF_INET socket on that TCP/UDP port, IPv6 traffic as well as IPv4 traffic should be routed to that AF_INET6 socket. IPv4 traffic should be seen as if it came from an IPv6 address like ::ffff:10.1.1.1. This is called an IPv4 mapped address.
- If there are both a wildcard bind AF_INET socket and a wildcard bind AF_INET6 socket on one TCP/UDP port, they should behave separately. IPv4 traffic should be routed to the AF_INET socket and IPv6 should be routed to the AF_INET6 socket.

However, RFC2553 does not define the ordering constraint between calls to bind(2), nor how IPv4 TCP/UDP port numbers and IPv6 TCP/UDP port numbers relate to each other (should they be integrated or separated). Implemented behavior is very different from kernel to kernel. Therefore, it is unwise to rely too much upon the behavior of AF_INET6 wildcard bind sockets. It is recommended to listen to two sockets, one for AF_INET and another for AF_INET6, when you would like to accept both IPv4 and IPv6 traffic.

It should also be noted that malicious parties can take advantage of the complexity presented above, and are able to bypass access control, if the target node routes IPv4 traffic to AF_INET6 socket. Users are advised to take care handling connections from IPv4 mapped address to AF_INET6 sockets.

SEE ALSO

ioctl(2), socket(2), sysctl(3), icmp6(4), intro(4), ip6(4), tcp(4), udp(4)

A. Conta, S. Deering, and M. Gupta, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC 4443, March 2006.

STANDARDS

Tatsuya Jinmei and Atsushi Onoe, *An Extension of Format for IPv6 Scoped Addresses*, internet draft, draft-ietf-ipngwg-scopedaddr-format-02.txt, June 2000, work in progress material.

HISTORY

The **inet6** protocol interfaces are defined in RFC2553 and RFC2292. The implementation described herein appeared in the WIDE/KAME project.

BUGS

The IPv6 support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

Users are suggested to implement "version independent" code as much as possible, as you will need to support both inet(4) and **inet6**.