ecTCP/IPv6

User’s Guide

Index

1 Introduction

1.1 Introduction

ecRTOS is a Real-Time Operating System based on the uT-Kernel 3.0 specification and fully compliant with the IEEE 2050-2018 standard.

Beyond the core kernel, ecRTOS also offers a comprehensive suit of middleware components for rapid development of networking, communication, storage, and security applications.

This document contains explanations related to ecTCP/IPv6, the dedicated IPv6 protocol stack for ecRTOS. Before using ecTCP/IPv6 and before making inquiries, please read this document carefully. For matters related to conventional TCP/IP protocol stack and ecRTOS kernel, please refer to the following documents:

ecRTOS Kernel User's Guide
ecTCP/IP User's Guide

1.2 Features

- ✧ IPv4/IPv6 dual stack
- ✧ Existing IPv4 applications can operate without modification
- ✧ Existing Ethernet drivers can be used as-is
- ✧ Supports IPsec, enabling communication of more confidential data
- ✧ Has acquired the IPv6 Ready Logo

1.3 Folder Structure

When ecTCP/IPv6 is installed, IPv6 code is copied into the following folders:

ecRTOS/IPV6/SRC	IPv6-specific source code is installed.
ecRTOS/IPV6/INC	IPv6-specific header files are installed.
ecRTOS/IPV6/LIB	IPv6-specific library files are installed.

Notes

Since neighbor discovery uses multicast packets, the driver must be configured to receive multicast packets.

1.4 File structure

ecnet6.c

Contains IPv6-related configuration and utility functions, as well as Neighbor Discovery processing.

ecnetip6.c

Contains IPv6 initialization functions and basic processing.

ecnip6f.c

Contains functions for IPv6 fragment processing.

ecnicmpv6.c

Contains processing related to ICMPv6.

ecnipsec.c

Contains processing related to IPsec.

ecnmcast.c

Contains processing related to the IPv6 Multicast Listener Discovery Protocol (MLD).

ecnmcast2.c

Contains processing related to the IPv6 Multicast Listener Discovery Protocol (MLDv2).

ecrypt.h

Defines the API for the encryption library.

ecnipsec.h

Defines variables and functions for IPsec-related processing.

ecn6xxxx.lib (*)

IPv6/IPv4 dual-stack library.

ecryptxxx.lib (*)

Encryption library used by IPsec.

(*) The library file extension varies depending on the compiler.

2 Overview of IPv6

This chapter provides an overview of IPv6. For more detailed information, please refer to relevant technical literature. Please note that some features are not supported by ecTCP/IPv6.

2.1 Changes from IPv4 to IPv6

IPv6 is a new version of the Internet Protocol designed as a successor to IPv4. The changes from IPv4 to IPv6 can be classified primarily as follows:

- **Address Extension and Automatic Configuration**
  The address length was extended to 128 bits to enable the use of a larger number of nodes and hierarchical structures (diversification of the address scheme), as well as to support automatic address configuration.

- **Simplified Header Format**
  The header length was reduced to 40 bytes, and several fields from IPv4 are now treated as optional. This reduces the load on header processing.

- **Improved Support for Extensions and Options**
  While IPv4 options were included in the header, in IPv6 they are treated as extension headers. Extension headers are inserted between the IPv6 header and the payload (the data itself). This enhances scalability and makes it easier to introduce new options in the future.

- **Flow Labeling Feature**
  This feature is primarily used for real-time communications. It allows the source to attach a label to packets that require special processing or quality of service.

- **Authentication and Privacy Protection Features**
  Support for authentication features and extensions for data integrity and confidentiality have been added.

2.2 Address Format

While IPv4 uses 32-bit addresses, IPv6 uses a 128-bit address space. IPv6 addresses are classified into three types.

- **Unicast**
  Indicates a single interface. Packets sent to a unicast address are delivered to the interface associated with that address.
- **Multicast**
  Indicates a group of IPv6 interfaces. Packets sent to a specified multicast address are delivered to all members belonging to that multicast group.
- **Anycast**
  Assigned to multiple interfaces. Anycast addresses generally belong to multiple different nodes. Packets sent to an anycast address are delivered to only one of those interfaces. Typically, they are delivered to the closest interface.

2.3 Address Notation

IPv6 uses 128-bit (16-byte) addresses. The address is divided into eight 16-bit segments, each represented by four hexadecimal digits and separated by colons (:).

FE80:0000:0000:0000:0240:CAFF:FE28:4ADA

If a 16-bit segment consists of consecutive zeros, or if the address begins or ends with zeros, these can be replaced with a colon (:).

FE80::240:CAFF:FE28:4ADA

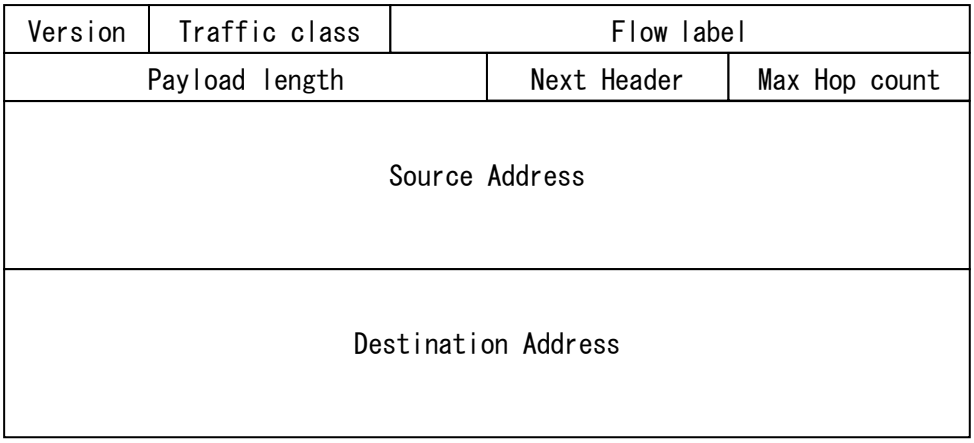
However, two colons can only be used in one place within an address. In the case of,

FE80:0000:0000:0000:0234:0000:FE78:9A9A

it becomes

FE80::234:0:FE78:9A9A

2.4 Definition of the IPv6 Packet Header



・Version (4bit)
Protocol Version number

・Traffic class (1byte)
Indicates the attributes of the IPv6 packet

・Flow label (20bit)
Used to improve router efficiency. A number that instructs routers to process a series of packets in the same way.

・Payload length (2byte)
The length of the data section following the IPv6 header

・Next header (1byte)
The type of the upper-layer protocol (same as the IPv4 protocol type field). If extension headers are used, the value of the extension header following the IPv6 header is written here.

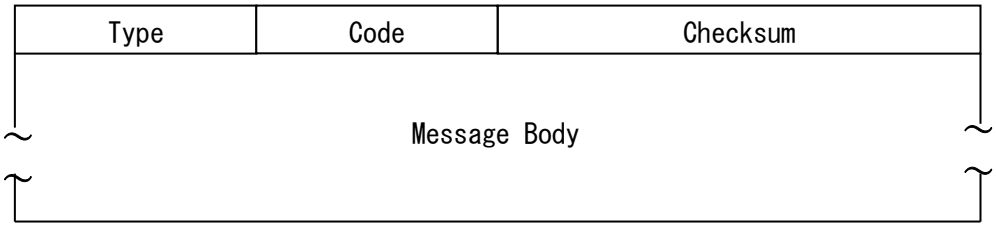
・Maximum Hop count (1byte)
Equivalent to IPv4's TTL

・Source address (16byte)
Source IP address

・Destination address (16byte)
Destination IP address

2.5 ICMP Version 6

ICMPv6 is used in IPv6 in place of IPv4's ICMP. ICMPv6 is used to notify of errors that occur during packet processing and to perform diagnostics (ICMPv6 "ping").



- **Type (1byte)**
  Indicates the type of message. It defines the format of the subsequent fields.
- **Code (1byte)**
  The code field varies depending on the message type, and detailed information specific to each message is set here.
- **Checksum (2byte)**
  Used to detect data corruption in the ICMPv6 header and parts of the IPv6 header.
- **Message body (Variable length)**
  The contents differ depending on the message type.

ICMPv6 messages can be classified into three categories.

- **ICMPv6 error message**

  If the MSB of the Type field is 0, the message is an error message.

  Therefore, the values for ICMP error messages range from 0 to 127.

  1 Destination unreachable
  2 Packet siz exceeded
  3 Time exceeded
  4 Parameter problem

- **ICMPv6 information message**

For informational messages, the MSB of the Type field is set to 1. Therefore, error message values range from 128 to 255.

  128 Echo Request
  129 Echo Reply

- **Neighbor discovery message**

  133 Router Solicitation
  134 Router Advertisement
  135 Neighbor Solicitation
  136 Neighbor Advertisement
  137 ICMP Redirect

2.6 Neighbor Discovery

Neighbor discovery includes functions equivalent to IPv4's ARP, as well as ICMP Router Discovery and ICMP Redirect. Additionally, functions to determine whether communication with neighboring nodes is possible and to detect duplicate addresses have been added. Neighbor discovery uses ICMPv6 messages.

2.6.1 ICMP Messages Used in Neighbor Discovery

The Neighbor Discovery Protocol uses the following five ICMP messages:

- Router Solicitation Message / Router Advertisement Message
- Neighbor Solicitation Message / Neighbor Advertisement Message
- ICMP Redirect Message

2.6.2 Neighbor Discovery Operation

Neighbor Discovery is used for the following purposes:

1) Address resolution to discover the Layer 2 (MAC address in the case of Ethernet) addresses of nodes on the same link (equivalent to IPv4 ARP).

A Neighbor Solicitation message is sent using a multicast address, and if the receiving host matches the target address, it returns a Neighbor Advertisement message.

2) Reachability detection: Maintaining information on whether neighboring nodes are reachable and detecting changes in link-layer addresses

A neighbor solicitation message is sent using a unicast address to check if the address is valid.

If there is no response to the neighbor solicitation message, the address may be invalid.

3) Duplicate address detection: Checking whether an address a node intends to use is already in use by another node

A neighbor request message is sent with the address to be checked for duplication; if a response is received via a neighbor notification message, the address is considered duplicated.

4) Acquiring the default route to discover a neighboring router that can forward packets

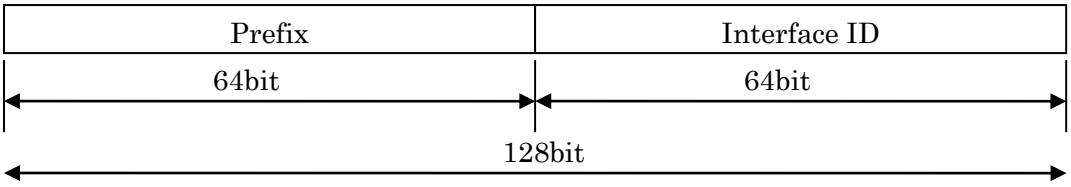
Routers periodically send router advertisement messages. A host can request a router advertisement message by issuing a router solicitation message.

2.7 Automatic Configuration

In IPv4, network configuration for hosts (such as automatic IP address acquisition) was performed using DHCP. While a DHCP server is required to use DHCP, IPv6 allows for automatic IP address configuration without using DHCP. The method of using DHCP for automatic address configuration is called stateful. In contrast, the method in

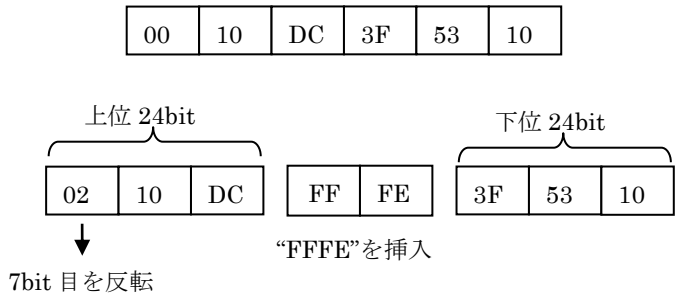
which a host generates its own address using information it can generate itself (interface ID) and information notified by the router (prefix) is called stateless. Using the stateless method eliminates the need to configure one's own IP address and eliminates the need for a DHCP server. Addresses created using this method are called link-local addresses. Link-local addresses can only be used within a local scope that does not cross routers.

In IPv6, the 64-bit address is divided into two parts with distinct functions: the upper 64 bits are called the prefix, and the lower 64 bits are called the interface ID. The prefix is used to identify the network and corresponds to the network address in IPv4. The interface ID is used to identify hosts on the network and corresponds to the host address in IPv4.



Since link-local addresses are used only within the same network, the prefix is FE80::. The interface ID must be unique for each host, and when using Ethernet, the MAC address is used as the interface ID.

Specifically, the 48-bit MAC address is split into two 24-bit segments, a 16-bit "fffe" is inserted between them, and the 0 and 1 in the 7th bit at the beginning are swapped. For example, if the MAC address is 00:10:DC:3F:53:10, the result is as follows.



The address becomes FF80::2010:DCFF:FE3F:5310.
Before using this address, the host performs a neighbor discovery to check for address conflicts. If no conflicts are found, the host begins using this address.

2.8 Security Management Using IPsec

IPsec is implemented as a standard feature in IPv6. IPsec performs authentication and encryption at the IP layer. Since IPsec encrypts and decrypts the payload (data) at the IP layer during transmission, TCP and UDP do not need to be aware that IPsec is in use.

2.8.1 Authentication Mechanisms

IPsec ensures the integrity of IP packets (verifying that the data has not been tampered with). To do this, it uses an extension header called the Authentication Extension Header (AH). The authentication mechanism typically uses the MD5-96 or SHA-1-96 algorithms.

2.8.2 Encryption

To enhance data confidentiality, IPsec uses an extension header called the Encapsulating Security Payload (ESP) header.

The standard encryption algorithms specified are DES-CBC (Data Encryption Standard in Cipher Block Chaining mode) and 3DES-CBC.

2.8.3 Security Association

When using IPv6 security features, the communicating parties must exchange and agree upon information such as keys, encryption algorithms, and the parameters used by those algorithms. This information is referred to as a Security Association (hereinafter SA). The SA supports two encapsulation modes.

- Transport Mode
  Transport mode is defined between two end systems, and only the payload (data) of all IP packets is encrypted.

- Tunnel Mode
  Tunnel mode is defined between two security gateways. IP packets are encapsulated within another IP packet, and the entire packet, including the IP header, is encrypted.

SAs are managed by two types of databases: the Security Association Database (SAD) and the Security Policy Database (SPD). ecTCP/IPv6 supports only transport mode.

3 Overview of ecTCP/IPv6

This chapter describes the features and overview of ecTCP/IPv6.

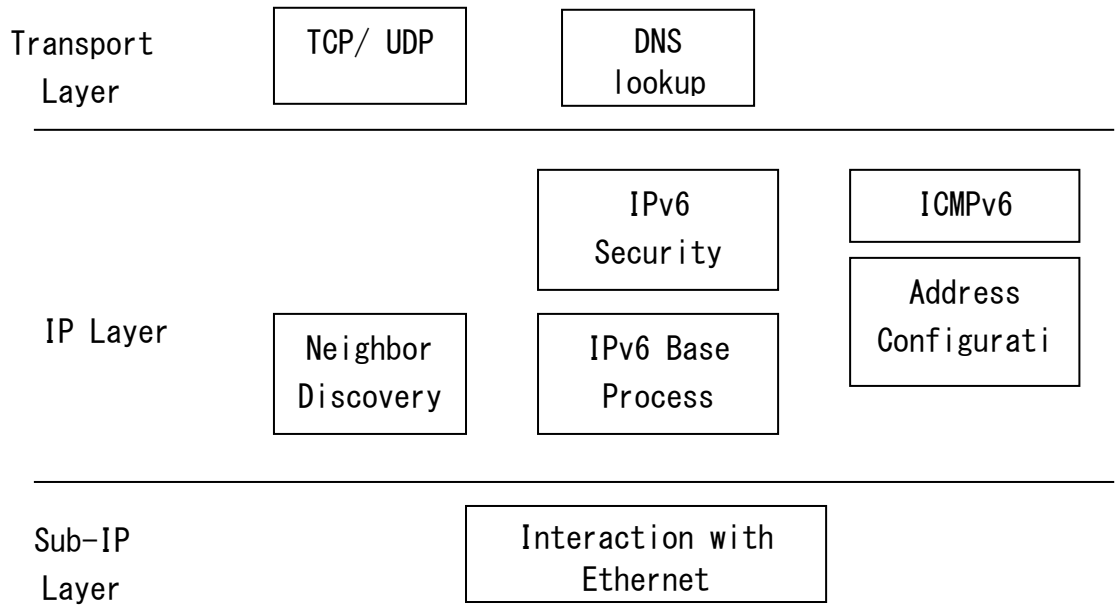
3.1 Layer Structure of ecTCP/IPv6

ecTCP/IPv6 consists of the following layers:

<table>
<tr><td>Application Layer</td><td colspan="3">μITRON TCP/IP API</td></tr>
<tr><td>Transport Layer</td><td colspan="3">TCP/UDP</td></tr>
<tr><td rowspan="3">Internet Layer</td><td colspan="2">IPv4</td><td>IPv6</td></tr>
<tr><td>ICMPv4</td><td>ARP</td><td>ICMPv6</td></tr>
<tr><td colspan="3">Data Link Module</td></tr>
<tr><td>Network Interface Layer</td><td colspan="3">Device Controller</td></tr>
</table>

3.2 Module Structure of ecTCP/IPv6

ecTCP/IPv6 consists of the following modules.



3.3 Address Formats

ecTCP/IPv6 supports the following two address formats:

・Unicast addresses
(Link-local addresses and aggregatable global unicast addresses)

・Multicast addresses (link-local multicast addresses only)

3.4 ICMPv6 and Neighbor Discovery

ecTCP/IPv6 supports the following ICMPv6 messages:

- Echo Request / Echo Reply
- Multicast Listener Query / Report / Terminate

ecTCP/IPv6 supports the following Neighbor Discovery messages:

- Neighbor Solicitation / Neighbor Advertisement
- Router Solicitation / Router Advertisement

3.5 Automatic Configuration

ecTCP/IPv6 supports stateless automatic address configuration. It generates a link-local address from the MAC address and uses Neighbor Solicitation messages to check for address conflicts. This check is performed when tcp_ini is called.

3.6 Compatibility

ecTCP/IPv6 supports both IPv4 and IPv6.

The IP layer implements both the IPv4 and IPv6 protocols. In addition to applications built on the traditional IPv4, applications using IPv6 can also be created.

・Resources Used

In addition to the traditional send and receive tasks, there is a task for periodic IPv6 processing.

・API

The conventional IPv4 API remains unchanged except for the address specification portion. T_IPV4EP has been extended for IPv6 and renamed to T_IPEP.

・Ethenet Driver

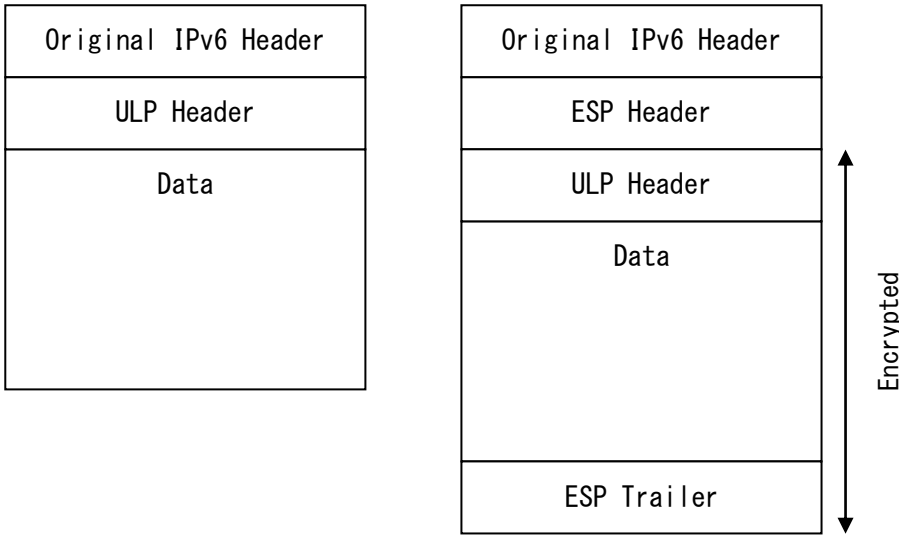
The conventional Ethernet driver can be used, but it must be configured to receive

multicast traffic. If you do not receive a response to commands such as ping6, please verify this setting.

3.7 IPsec

When IPsec is integrated into ecTCP/IPv6, both the Authentication Extension Header (hereinafter referred to as AH) and the Encapsulating Security Payload Header (hereinafter referred to as ESP) are used. Encapsulation is supported only in transport mode.

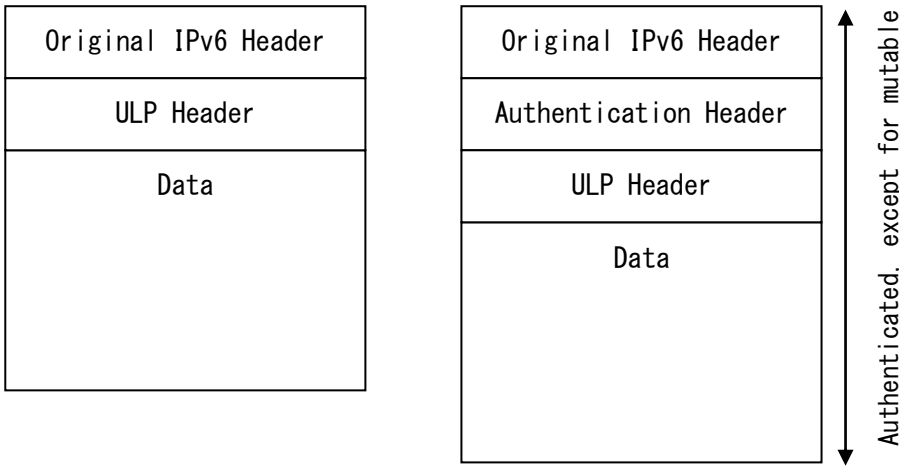
・Packets using ESP



Original IPv6 packet

IPv6 packet with Encryption applied

・Packets using AH



Original IPv6 packet

Authenticated IPv6 packet

・Encryption Algorithms

AH uses SHA-1-96 and MD5-96, while ESP uses DES-CBC or 3DES-CBC.

・SA Management

SA management is performed manually. The Security Association Database (SAD) and Security Policy Database (SPD) are defined in the file ipsec_db.c. For details, refer to Chapter 5.

3.8 MLD (Multicast Listener Discovery)

Multicast Listener Discovery (MLD) is used for managing multicast groups. The following API is available in ecTCP/IPv6.

<table>
<tr><td>Syntax</td><td colspan="3">ER udp_set_opt(ID cepid, INT optname, VP optval, INT optlen)
cepid Communication endpoint ID
optname Option name
optval Pointer to buffer holding the option
optlen Option length</td></tr>
<tr><td>Return value</td><td colspan="3">E_OK Successful completion
E_ID Invalid ID
E_NOEXS Communication endpoint not created
E_PAR Specified address is not a multicast address
E_NOMEM Out of memory</td></tr>
<tr><td rowspan="6">Option</td><td>optname</td><td>optval</td><td>contents</td></tr>
<tr><td>IPV6_JOIN_GROUP</td><td>Multicast address</td><td>Join a multicast group</td></tr>
<tr><td>IPV6_LEAVE_GROUP</td><td>Multicast address</td><td>Leave a multicast group</td></tr>
<tr><td>MCAST_INCLUDE</td><td>Source List</td><td>Set the source list to INCLUDE mode</td></tr>
<tr><td>MCAST_EXCLUDE</td><td>Source List</td><td>Set the source list to EXCLUDE mode</td></tr>
<tr><td>ALLOW_SOURCES</td><td>Source List</td><td>Enable filter specifications for the source list</td></tr>
<tr><td></td><td>BLOCK_SOURCES</td><td>Source List</td><td>Block filter specifications for the source list</td></tr>
</table>

3.9 Add/Delete/Retrieve Addresses & Status Notification Callbacks

Func	Adding a Unicast Address
Syntax	ER ipv6_add_addr(T_NIF *nif, UW *addr, UW prefix_len) nif Pointer to the network interface control block (if NULL, the default interface is used) addr Pointer to the area containing the IP address to be added prefix_len Prefix length of the IP address to be added
Return value	E_OK Successful completion E_OBJ Network interface does not exist E_ILUSE Address already exists on the network, or insufficient storage space for the address E_PAR One of the arguments is invalid
Detail	Add a unicast address to the network interface. You can configure up to four unicast addresses per interface. The `addr` variable is of type `UW` and reserves 128 bits of memory to store the address to be added. To add FE80:0000:0000:0000:1034:56FF:FE78:9A9A (prefix length: 64 bits) to the default interface, use the following example: ER ercd; T_NIF *nif = getnif_default(); UW addr = { FE800000, 0, 103456FF, FE789A9A }; ercd = ipv6_add_addr(nif, addr, 64); Note that this API will be blocked until duplicate address detection is complete.

Func	Adding a Unicast Address
Syntax	ER ipv6_add_addr_by_ch(int ch, UW *addr, UW prefix_len) nif CH number addr Pointer to the area where the IP addresses to be added are stored prefix_len Prefix length of the IP address to be added
Return value	E_OK Successful completion E_OBJ Network interface of CH number does not exist E_ILUSE Address already exists on the network, or insufficient storage space for the address E_PAR One of the arguments is invalid
Detail	Adds a unicast address to the network interface specified by the CH number. Except for the requirement to specify the CH number, this follows the same procedure as `ipv6_add_addr`.

Func	Adding a Unicast Address
Syntax	ER ipv6_add_addr_by_name(const char if_name, UW *addr, UW prefix_len) if_name Network I/F name addr Pointer to the area where the IP addresses to be added are stored prefix_len Prefix length of the IP address to be added
Return value	E_OK Successful completion E_OBJ Network interface with specified name does not exist E_ILUSE Address already exists on the network, or insufficient storage space for the address E_PAR One of the arguments is invalid
Detail	This command adds a unicast address to the network interface specified by the network interface name. It is similar to ipv6_add_addr except that it specifies the network interface name.

Func	Deletes unicast addresses
Syntax	ER ipv6_del_addr(T_NIF *nif, UW *addr) nif Pointer to the NIF network interface control block (uses default I/F, if NULL) addr Pointer to the IP address to delete
Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid E_NOEXS Specified address does not exist
Detail	This removes unicast addresses from the network interface. The `addr` variable is of type `UW` and reserves 128 bits of memory to store the address. To remove `FE80:0000:0000:0000:1034:56FF:FE78:9A9A` from the default interface, use the following example: ER ercd; T_NIF *nif = getnif_default(); UW addr = { FE800000, 0, 103456FF, FE789A9A }; ercd = ipv6_del_addr(nif, addr);

Func	Delete a unicast address
Syntax	ER ipv6_del_addr_by_ch(int ch, UW *addr) ch Channel number addr Pointer to the IP address to be deleted

Return value	E_OK Successful completion E_OBJ Network interface corresponding to the channel number does not exist E_PAR One of the arguments is invalid E_NOEXS Specified address does not exist
Detail	Deletes a unicast address on the network interface specified by the CH number. Except for the requirement to specify the CH number, this function follows the same rules as `ipv6_del_addr`.

Func	Delete unicast address
Syntax	ER ipv6_del_addr_by_name(const char *if_name, UW *addr) if_name Network interface name addr Pointer to the IP address to be deleted
Return value	E_OK Successful completion E_OBJ Network I/F with specified interface name does not exist E_PAR One of the arguments is invalid E_NOEXS Specified address does not exist
Detail	Deletes a unicast address from the network interface specified by the network interface name. Except for the use of the network interface name, this function follows the same behavior as `ipv6_del_addr`.

Func	Retrieving Address Information
Syntax	ER ipv6_get_addr(T_NIF *nif, T_IPV6_ADDR *buf, UW buf_size, UW *found_num, UH addr_type_flag) nif Pointer to the network interface control block (if NULL, the default interface is used) buf Pointer to the area where address information is stored buf_size Number of areas where address information is stored found_num Pointer to the area for storing the number of addresses assigned to the specified network interface addr_type_flag Type of address to be retrieved typedef struct t_ipv6_addr { UW addr[4]; IPv6 address UH scope; IPv6 address scope UW prefix_len; IPv6 address prefix length B status; IPv6 address status } T_IPV6_ADDR;

Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid
Detail	The following two address types can be specified: You can specify either of the following two address types. ADDR_UNICAST Retrieves information on the unicast address configured on the specified network interface ADDR_MULTICAST Retrieves information on the multicast address configured on the specified network interface The information stored in T_IPV6_ADDR is as follows: addr IPv6 address In case of FE80:0000:0000:0000:1034:56FF:FE78:9A9A, It is stored in this format. addr[0] FE800000 addr[1] 0 addr[2] 103456FF addr[3] FE789A9A scope IPv6 address scope One of the following four types will be stored. ADDR_IF_LOCAL 0x0100 Interface local ADDR_LINK_LOCAL 0x0200 Link local ADDR_SITE_LOCAL 0x0800 Site local ADDR_GLOBAL 0x2000 Global prefix_len IPv6 address prefix length status Status of the IPv6 address One of the following four types is stored. ADDR_TENTATIVE (8) Tentative address (before duplicate address detection is complete) ADDR_PREFERRED (1) Valid address (after duplicate address detection is complete) ADDR_DEPRECATED (2) Deprecated address (valid lifetime about to expire) ADDR_PERMANENT (7) Permanent address (address specified using ipv6_add_addr, etc.) To retrieve unicast address information from the default interface, use the following example: ER ercd; T_NIF *nif = getnif_default();

	T_IPV6_ADDR buf[4]; UW found_num ercd = ipv6_get_addr(nif, (T_IPV6_ADDR *)buf, 4, &found_num, ADDR_UNICAST); Since the maximum number of unicast and multicast addresses that can be assigned to a single network interface is 4, `buf_size` must be 4 or less. Valid addresses and persistent addresses are available for communication. Deprecated addresses are available, but they will be deleted if no router advertisement is received before their valid lifetime expires. Temporary addresses cannot be used for communication because duplicate address detection has not yet been completed.

Func	Retrieving Address Information
Syntax	ER ipv6_get_addr_by_ch(int ch, T_IPV6_ADDR *buf, UW buf_size, UW *found_num, UH addr_type_flag) ch Channel number buf Pointer to the area where address information is stored buf_size Number of address entries to be stored found_num Pointer to the area for storing the number of addresses assigned to the specified network interface addr_type_flag Type of address to retrieve typedef struct t_ipv6_addr { UW addr[4]; IPv6 address UH scope; IPv6 address scope UW prefix_len; IPv6 address prefix length B status; IPv6 address status } T_IPV6_ADDR;
Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid
Detail	Retrieves address information for the network interface specified by the CH number. Except for the use of the CH number, this function follows the same behavior as `ipv6_get_addr`.

Func	Registering/Removing Status Notification Callback Functions
Syntax	ER ipv6_def_addr_chg_cbk(T_NIF *nif, IPV6_ADDR_CHG_CALLBACK func) nif Pointer to the network interface control block (if NULL, the default interface is used) func Pointer to the callback function. If NULL is specified, the callback function is removed. For details on callback functions, refer to Callbacks below.
Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid
Detail	This function registers or removes a callback function that is called when the address status of the specified network interface changes. The definition of the callback function is as follows: typedef void (*IPV6_ADDR_CHG_CALLBACK)(int ch, const char *if_name);

Func	Registering/Removing Status Notification Callback Functions
Syntax	ER ipv6_def_addr_chg_cbk_by_ch(int ch, IPV6_ADDR_CHG_CALLBACK func) ch Channel number func Pointer to the callback function. If NULL is specified, the callback function is removed. For details on callback functions, refer to Callbacks below.
Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid
Detail	This function registers or removes a callback function that is invoked when the address status of the network interface specified by the CH number changes. Except for the use of the CH number, this function follows the same behavior as `ipv6_def_addr_chg_cbk`.

Func	Registering/Removing Status Notification Callback Functions
Syntax	ER ipv6_def_addr_chg_cbk_by_name(const char *if_name, IPV6_ADDR_CHG_CALLBACK func) if_name Network interface name func Pointer to the callback function. If NULL is specified, the callback function is removed. For details on callback functions, refer to Callbacks below.

Return value	E_OK Successful completion E_OBJ Network interface does not exist E_PAR One of the arguments is invalid
Detail	This function registers or removes a callback function that is invoked when the address status of the network interface specified by the network interface name changes. Except for the requirement to specify the network interface name, this function follows the same behavior as `ipv6_def_addr_chg_cbk`.

Func	Callback
Syntax	void (*callback)(int ch, const char *if_name) ch Channel number of the network interface whose address status has changed if_name Name of the network interface whose address status has changed
Return value	None
Detail	This function is called when the address status of a network interface changes. The function name is arbitrary. It is called when a temporary address becomes a valid address or a persistent address, or when an address is deleted due to the expiration of its valid lifetime. This callback function is executed in the context of the TCP/IPv6 protocol stack. Do not issue system calls that may cause a transition to a waiting state, such as `slp_tsk`. Also, ensure that the processing time is kept as short as possible. For details on transitions to waiting states, please refer to "1.2 Task States" in the ecRTOS Version 4 User's Guide: Kernel Edition. Note that this callback function is defined in ecnet.h as follows: typedef void (*IPV6_ADDR_CHG_CALLBACK)(int ch, const char *if_name);

3.10 Limitations

ecTCP/IPv6 implements only the bare minimum of IPv6 specifications. The following features are not implemented in the current version of ecTCP/IPv6:

・Support for PPP over IPv6

・Route MTU discovery

- IPv6 jumbo packets
- Stateful address autoconfiguration
- DHCPv6
- Automation of Security Associations (SAs) and Key management for IPsec
- IPsec tunnel mode
- Tunneling to IPv4 packets

4 Configuration

4.1 Libraries

All libraries included in the LIB folder are designed with a dual-stack architecture supporting both IPv4 and IPv6. These files are built with the DUAL_STK macro defined. If built without the DUAL_STK macro, only IPv4 libraries will be generated. IPv6-only libraries cannot be generated.

4.2 Definitions

Configuration can be performed by defining the following constants before including the configuration header ecnetc.h. The values in parentheses are the default values used when no specific value is specified.

#define NEIGH_CACHE_CNT	(8)	Number of entries that can be registered in the neighbor cache
#define PREFIX_LIST_CNT	(8)	Maximum number of prefixes that can be registered
#define DFLT_ROUTER_CNT	(2)	Maximum number of routers that can be registered in the router cache
#define DST_CACHE_CNT	(8)	Next-hop address counter
#define DAD_TMO	(1000)	Duplicate address detection timeout (milliseconds)
#define IP6F_REASM_TMO	(2)	Timeout for IPv6 fragment packet reassembly (seconds)
#define IP6HASH_QSZ	(2)	Size of hash queue for fragment packet reassembly
#define MAX_IPSEC_ENTRIES	(15)	Maximum number of entries in the IPsec database

Note
When using IPsec with a software encryption library, the stack size used by IP transmission and reception tasks may be insufficient. Therefore, be sure to define the IPSEC macro before including nonetc.h. This will automatically adjust the stack size internally.
Example)
#define IPSEC
#include "nonetc.h"

5 IPSec

To use IPsec, you must link the IPsec library. At the same time, the application must link the security database file and the file containing functions for encryption/decryption and hashing algorithms.

5.1 IPsec Database

IPsec is implemented based on the concept of Security Associations (SAs). In IPsec for ecTCP/IPv6, SAs must be managed manually. An SA is a unidirectional connection that secures specific traffic, specifying the security protocol, mode, endpoint addresses, and optional services. SAs utilize the Security Policy Database (SPD) and the Security Association Database (SAD). When transmitting IP packets, the SPD, which stores security policies, is referenced. The SAD contains parameters related to the SA.

5.1.1 Security Policy Database (SPD)

The Security Policy Database (SPD) is defined in `ipsec_db.c` as `T_SPD_ENTRY spd_ref[]`. The following items are configured here:

Item	Description
index	Database record index number
rmt_addr	Remote host IPv6 address
local_addr	Local host IPv6 address
transport	Transport layer protocol
rmt_port	Port number of the remote host application
local_port	Port number of the local host application
action	Application action policy
direction	Traffic direction
sad_ptr	Starting index number of the associated SA entry

index - Specifies a unique index number for the SPD record, starting from 1. This value must be incremented for each entry.

rmt_addr & local_addr - These fields contain the addresses of the respective hosts communicating via IPsec. To communicate with all hosts, define IPV6_ADDRANY.

transport - Defines the transport layer protocol using IPsec (PROT_TCP, PROT_UDP, PROT_ICMPV6). Specify '*' to use all protocols.

rmt_port & **local_port** - Specify the port numbers for the remote host and local host. Specify '*' to communicate with all ports.

action - Specify the application's behavior policy from the following options:
IPSEC_APPLY - Apply IPsec
IPSEC_DISCARD - Discard
IPSEC_BYPASS - Bypass IPsec
direction - Sets the traffic direction. The value is fixed at TRAFFIC_BIDIR.
sad_ptr - Specifies the starting index number of the associated SA entry. This is not used if a value other than IPSEC_APPLY is specified for action.

* Communication via IPsec is not possible with nodes not defined in the SPD. You must register information for all nodes with which you intend to communicate via IPsec. If you wish to communicate with other nodes without using security, you must add the following definitions at the end of the SPD:
Set rmt-addr and local_addr to IPV6_ADDRANY
Set transport, rmt_port, and local_port to '*'
Set action to IPSEC_BYPASS

5.1.2 Security Association Database (SAD)

The Security Association Database (SAD) is defined in `ipsec_db.c` within the `T_SAD_ENTRY` structure's `sad_ref[]` array. The following fields are defined here:

Field	Description
index	Index number of the database record
seqcntr	Sequence number used in the AH or ESP header
spi	Security Parameter Index
dst_addr	Destination IPv6 address
proto	Security protocol used (AH or ESP)
mode	Security mode
auth_alg	Authentication algorithm
antireply	Anti-replay protection flag
auth_key	Secret key used for authentication
auth_key_len	Length of the secret key used for authentication
encr_alg	Encryption algorithm
esp_ah	Flag indicating whether to include an authentication data field in the ESP
encr_key	Private key used for encryption
encr_key_len	Length of the private key used for encryption

iv	Initialization vector used in the ESP
direction	Direction of the security association
bundle_ptr	Index to the next SA entry
spd_ptr	Starting index number of the associated SP entry

index - Specifies a unique index number for the record, starting from 1. This value must be incremented for each entry.

seqcntr - The sequence number used in the AH or ESP header. Specify a 32-bit random value.

spi - Specifies the Security Parameter Index. This value must match that of the remote host with which you are communicating.

dst_addr - Destination IPv6 address. Use the remote address when sending data, and the local address when receiving data.

proto - Specifies the security protocol. Specify PROT_AH when using the AH protocol, and PROT_ESP when using the ESP protocol.

mode - Specifies the security mode. This value is fixed at MODE_TRANSPORT.

auth_alg - Specifies the authentication algorithm.
You can set this to AUTH_NULL (NULL), AUTH_HMAC_SHA1_96 (SHA1), or AUTH_HMAC_MD5_96 (MD5).

antireply - Configures the replay prevention feature. This value is fixed at '0'.

auth_key and auth_key_len - Specify the secret key and key length used for authentication. You can set HMAC-SHA1-96 (SHA1), which uses a 20-byte key, or HMAC-MD5-96 (MD5), which uses a 16-byte key.

encr_alg - Specifies the encryption algorithm. You can set ESP_NULL (NULL), ESP_DES_CBC (DES), or ESP_3DES_CBC (3DES).

esp_ah - A flag indicating whether to include an authentication data field in ESP. If set to '0', authentication data is not included. If set to '1', authentication data is included.

encr_key and encr_key_len - Specify the secret key and key length used for encryption. The key can be set to 8 bytes for DES-CBC or 24 bytes for 3DES-CBC.

iv - Sets the initialization vector used by ESP. Set an 8-byte random value here.

direction - This member should be specified as TRAFFIC_INBOUND for incoming packet and TRAFFIC_OUTBOUND for outgoing packet.

bundle_ptr - Index to the next SA entry. Set this value to '0'.

spd_ptr - Specifies the starting index number of the associated SPD entry.

※ Note: Since an SA is a unidirectional connection, the host performing secure

communication must have two SAs-one for the sender and one for the receiver-registered in the SAD.

※ Note: If "action" is set to a value other than IPSEC_APPLY in the SPD entry, registration in the SAD is not required.

5.2 Encryption/Decryption and Hash Algorithms

ecTCP/IPv6's IPsec supports MD5 and SHA-1 as hash algorithms and DES-CBC and 3DES-CBC as encryption/decryption algorithms by default. These functions are included in the ecryptxxx.lib encryption library.

Performing these operations in software reduces communication speed to a fraction of its original rate. To achieve high-speed communication using IPsec, we recommend using a hardware accelerator for encryption/decryption and hashing operations. The following interface functions are called from within IPsec. By implementing these functions, you can utilize hardware accelerators with different specifications.

Func	MD5 Hash Calculation
Syntax	ER ipsec_hmac_md5(UB* in_data, UW in_size, UB* secret_key, UW key_len, UB *output, UW req_len); in_data Pointer to the buffer containing the input data in_size Length of the input data secret_key Pointer to the buffer containing the secret key key_len Length of the secret key output Pointer to the buffer that will store the calculated data req_len Size of the buffer to be used
Return value	Length of the stored hash
Detail	Performs an MD5 hash calculation.

Func	SHA-1 hash calculation
Syntax	ER ipsec_hmac_sha1(UB* in_data, UW in_size, UB* secret_key, UW key_len, UB *output, UW req_len); in_data Pointer to the buffer containing the input data in_size Length of the input data secret_key Pointer to the buffer containing the secret key key_len Length of the secret key output Pointer to the buffer that will store the calculated data

	req_len Size of the buffer to be used
Return value	Length of the stored hash
Detail	Performs an SHA-1 hash calculation.

6 Changes from the IPv4 stack

6.1 T_IPEP

The structure `T_IPV4EP`, used to specify addresses for creating communication endpoints and sockets, has been extended to support IPv6 addresses and renamed `T_IPEP`.

```
typedef struct t_ipep
{
  UW            ipaddr;         /* IPv4 address */
  UH            portno;         /* Port number */
  UW            ip6addr[4];     /* IPv6 address */
  BOOL          type;           /* Address type to use */
} T_IPEP;
```

For `type`, specify `IPV4_ADDR` when specifying an IPv4 address and `IPV6_ADDR` when specifying an IPv6 address.

The T_TCP_CREP and T_UDP_CCEP functions, which used the T_IPV4EP structure, have been changed to use T_IPEP.

6.2 Specifying the Local IP Address

In functions that set the local IP address, IPV6_ADDRANY can be used in the same way as the previously supported IPV4_ADDRANY. If IPV6_ADDRANY is specified for the IP address, an automatically generated link-local address is set by the protocol stack. You should normally use IPV6_ADDRANY.

6.3 Byte Order Conversion

The following function has been added to perform byte order conversion for 128-bit IPv6 addresses used in IPv6.

ntohl6

【Function】Converts from network order to host order

【Syntax】 UW *ntohl6(UW *hl, UW *nl);

hl Pointer to the IPv6 address converted to host order

nl Pointer to the IPv6 address in network byte order

【Return Value】Pointer to the IPv6 address converted to host order

htonl6

【Function】Converts a host-order IPv6 address to network-order

【Syntax】UW *htohl6(UW *nl, UW *hl);

nl	Pointer to an IPv6 address in network-order
hl	Pointer to an IPv6 address converted to host-order

【Return Value】Pointer to an IPv6 address converted to network-order

6.4 ICMPv6

Functions for ICMPv6 have been added.

By registering a user-defined callback function using `icmpv6_def_cbk`, you can receive packets other than ECHO packets. Use `icmpv6_snd_dat` for transmission.

icmpv6_def_cbk

【Function】Registering an ICMP Receive Callback Function

【Syntax】ER icmpv6_def_cbk(T_ICMP6_CB *b, ICMP6_CALLBACK callback)

b	ICMPv6 control block
callback	Callback function

【Return Value】

E_OK	Successful completion
Negative value	Abnormal termination

【Details】You can register a function to be called when ICMPv6 packets other than Echo Request, Neighbor Solicitation, Neighbor Advertisement, Redirect, MLD Query, MLD Report and Router Advertisement are received.
ICMPv6 control blocks are managed in a chained structure, allowing multiple callback functions to be registered.
The ICMP control block area requires no initialization; the user simply needs to allocate this area as a variable and pass it to the function.

Callback

【Function】ICMP Receive Callback

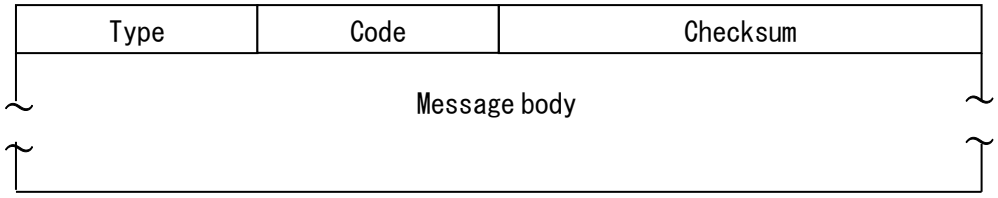
【Syntax】VP *callback(T_ICMP6_CB *b, T_IP6 *ip, T_ICMP6_MSG *icmp, INT len);

b	Pointer to the ICMPv6 control block (not normally used)
ip	Pointer to the IPv6 packet
icmp6	Pointer to the ICMPv6 message
len	Packet length

```
typedef struct t_icmp6_msg {
```

```
        T_ICMPV6_HEADER icmpv6;
        UB      data[IP6_HEADER_SZ+8];/* Option Data (Variable Size) */
} T_ICMP6_MSG;
```

【Details】This is the function specified for the ICMPv6 receive callback. The function name can be anything. This function is called when an ICMPv6 packet other than an Echo Request, Neighbor Solicitation, Neighbor Advertisement, Redirect, MLD Query, MLD Report, or Router Advertisement is received. The ICMPv6 message format is as follows. The data following the first 32 bits varies depending on the message.



icmpv6_snd_dat

【Function】Send ICMP Packet

【Syntax】ER icmpv6_snd_dat(UW *src6addr, UW *dst6addr, T_ICMP6_HEADER *icmpv6, VP data, INT len);

src6addr	Local IPv6 address
dst6addr	Destination IPv6 address
icmpv6	Pointer to the ICMPv6 header to be sent
data	Pointer to the ICMPv6 data to be sent
len	Data length

【Return Value】

E_OK	Successful completion
Negative value	Abnormal termination

【Details】You can send arbitrary ICMPv6 packets. This function can also be called from within the callback functions described above. See ping6_command for an example.

7 IPv6 Sample Application

ecTCP/IPv6 includes the following sample applications for evaluating IPv6.

Ping6 (ecRTOS¥NETSMP¥SRC¥ecnping.c)

This is the IPv6 version of the Ping command. It uses the ICMPv6 echo function.

IPv6 FTP Server/Client (ecRTOS¥NETSMP¥SRC¥ecnftpd.c/ecnftpc.c)

These are IPv6 versions of an FTP server and client.

IPv6 Telnet Server/Client (ecRTOS¥NETSMP¥SRC¥ecnteld.c/ecntelc.c)

This is an IPv6 Telnet server. It can connect to IPv6-compatible Telnet clients and servers.

IPv6 ECHO Server/Client (ecRTOS¥NETSMP¥SRC¥ecnechc.c/ecnecho.c)

An echo program (port 7) using TCP and UDP.

IPv6 DNS Resolver (ecRTOS¥NETSMP¥SRC¥ecnedns.c)

A function for converting hostnames to IP addresses.

ecRTOS

IEEE 2050-2018 compliant Real Time OS

www.ecrtos.com

General inquiry / Technical support request: support@ecrtos.com