UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Q.2111

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

Enmienda 2
(04/2002)

SERIE Q: CONMUTACIÓN Y SEÑALIZACIÓN

Red digital de servicios integrados de banda ancha (RDSI-BA) – Capa de adaptación del modo de transferencia asíncrono de señalización

---

Capa de adaptación del modo transferencia asíncrono de la RDSI-BA – Protocolo con conexión específico de servicio en un entorno multienlace y sin conexión

Enmienda 2: Interfaz de programa de aplicación para protocolo con conexión específico de servicio en un entorno multienlace y sin conexión por Ethernet

Recomendación UIT-T Q.2111 – Enmienda 2

RECOMENDACIONES UIT-T DE LA SERIE Q

CONMUTACIÓN Y SEÑALIZACIÓN

SEÑALIZACIÓN EN EL SERVICIO MANUAL INTERNACIONAL	Q.1–Q.3
EXPLOTACIÓN INTERNACIONAL SEMIAUTOMÁTICA Y AUTOMÁTICA	Q.4–Q.59
FUNCIONES Y FLUJOS DE INFORMACIÓN PARA SERVICIOS DE LA RDSI	Q.60–Q.99
CLÁUSULAS APLICABLES A TODOS LOS SISTEMAS NORMALIZADOS DEL UIT-T	Q.100–Q.119
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 4	Q.120–Q.139
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 5	Q.140–Q.199
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 6	Q.250–Q.309
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R1	Q.310–Q.399
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R2	Q.400–Q.499
CENTRALES DIGITALES	Q.500–Q.599
INTERFUNCIONAMIENTO DE LOS SISTEMAS DE SEÑALIZACIÓN	Q.600–Q.699
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 7	Q.700–Q.799
INTERFAZ Q3	Q.800–Q.849
SISTEMA DE SEÑALIZACIÓN DIGITAL DE ABONADO N.° 1	Q.850–Q.999
RED MÓVIL TERRESTRE PÚBLICA	Q.1000–Q.1099
INTERFUNCIONAMIENTO CON SISTEMAS MÓVILES POR SATÉLITE	Q.1100–Q.1199
RED INTELIGENTE	Q.1200–Q.1699
REQUISITOS Y PROTOCOLOS DE SEÑALIZACIÓN PARA IMT-2000	Q.1700–Q.1799
ESPECIFICACIONES DE LA SEÑALIZACIÓN RELACIONADA CON EL CONTROL DE LLAMADA INDEPENDIENTE DEL PORTADOR	Q.1900–Q.1999
RED DIGITAL DE SERVICIOS INTEGRADOS DE BANDA ANCHA (RDSI-BA)	Q.2000–Q.2999
Aspectos generales	Q.2000–Q.2099
Capa de adaptación del modo de transferencia asíncrono de señalización	**Q.2100–Q.2199**
Protocolos de red de señalización	Q.2200–Q.2299
Aspectos comunes de los protocolos de aplicación de la RDSI-BA para la señalización de acceso, la señalización de red y el interfuncionamiento	Q.2600–Q.2699
Protocolos de aplicación de la RDSI-BA para señalización de red	Q.2700–Q.2899
Protocolos de aplicación de la RDSI-BA para señalización de acceso	Q.2900–Q.2999

Para más información, véase la Lista de Recomendaciones del UIT-T.

Recomendación UIT-T Q.2111

Capa de adaptación del modo transferencia asíncrono de la RDSI-BA – Protocolo con conexión específico de servicio en un entorno multienlace y sin conexión

Enmienda 2
Interfaz de programa de aplicación para protocolo con conexión específico de servicio en un entorno multienlace y sin conexión por Ethernet

Resumen

En esta enmienda a la Rec. UIT-T Q.2111 se presenta una interfaz de programación de aplicación (API) para el protocolo con conexión específico de servicio en un entorno multienlace y sin conexión (SSCOPMCE) por Ethernet. Su introducción facilita la incorporación de SSCOPMCE en los sistemas de comunicación que utilizan Ethernet.

Orígenes

La enmienda 2 a la Recomendación UIT-T Q.2111, preparada por la Comisión de Estudio 11 (2001-2004) del UIT-T, fue aprobada por el procedimiento de la Resolución 1 de la AMNT el 13 de abril de 2002.

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Asamblea Mundial de Normalización de las Telecomunicaciones (AMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución 1 de la AMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Recomendación UIT-T Q.2111

Capa de adaptación del modo transferencia asíncrono de la RDSI-BA – Protocolo con conexión específico de servicio en un entorno multienlace y sin conexión

Enmienda 2
Interfaz de programa de aplicación para protocolo con conexión específico de servicio en un entorno multienlace y sin conexión por Ethernet

1) Cláusula 2.2 – Bibliografía

Añádase la siguiente referencia:

[23] ISO/CEI 8652: 1995, *Information technology – Programming languages – Ada*.

2) Cláusula 4 – Abreviaturas

Añádase la siguiente definición alfabéticamente:

API Interfaz de programación *(application programming interface)*

3) Cláusula 5.3 – Modos de funcionamiento

Añádase la siguiente oración al final del párrafo que sigue inmediatamente a la figura 2:

Además, en el anexo F se describe una interfaz de programación de aplicación (API, *application programming interface*) para SSCOPMCE por Ethernet.

4) Anexo F

Añádase un nuevo anexo F (API para SSCOPMCE por Ethernet) como sigue:

Anexo F

API para SSCOPMCE por Ethernet

1 Introducción

En el anexo E a esta Recomendación se especifica el despliegue de SSCOPMCE por encima del servicio sin conexión proporcionado por las redes Ethernet IEEE 802.3. El objetivo primordial de esta configuración es realizar un bus de datos de sistemas abiertos para sistemas de bucle cerrado.

Las aplicaciones pueden utilizar los siguientes servicios de SSCOPMCE a través del punto de acceso al servicio (SAP) ofrecido por la función de coordinación específica de servicio (SSCF) en la interfaz usuario-red (UNI) [12]:

- Transferencia de datos sin acuse de recibo.
- Transferencia de datos asegurada.
- Transparencia de la información transferida.
- Establecimiento y liberación de conexiones para la transferencia de datos asegurada.

En tanto que el cuerpo principal de la Recomendación y el anexo E contienen las especificaciones necesarias para desarrollar un producto basado en una tarjeta de interfaz de red Ethernet, este anexo especifica una interfaz de programación de aplicación (API) al SAP. La razón para especificar una API es incitar a los vendedores de herramientas de desarrollo y/o sistemas operativos en tiempo real a ofrecer una interfaz normalizada, abierta y familiar para que los desarrolladores de soporte lógico se beneficien de las capacidades de red ofrecidas por un bus de datos basado en Ethernet.

2 Objetivos de la API de bus de datos Ethernet

La API de bus de datos Ethernet es relativamente pequeña y autónoma, y permite a un programador acceder a los servicios SSCOPMCE cuando estos servicios funcionan por una capa de enlace de datos Ethernet. En el diseño de la API se consideraron dos objetivos:

- La API debería basarse en la noción de zócalos (conectores lógicos), los cuales se han utilizado ampliamente en la mayoría de las API de las redes existentes para sistemas de sobremesa y sistemas operativos en tiempo real. Los zócalos tratan esencialmente cada conexión de red como un tren en el cual se pueden escribir o leer octetos, permitiéndoles constituir una ampliación de los conceptos familiares de entrada/salida (I/O) de ficheros.
- La API debería incluir disposiciones para el tratamiento de las excepciones, a fin de gestionar los errores que aparecen en curso de ejecución.

3 Visión general de la implementación de la API de bus de datos Ethernet

La API de bus de datos Ethernet está escrita en el lenguaje de programación Ada 95 [3]. La selección de Ada está basada en su amplia utilización en sistemas aeroespaciales y de defensa, uno de los sectores de aplicación que motivaron la especificación del anexo E/Q.2111. En consecuencia, una API basada en Ada permitirá la migración de las arquitecturas de sistema existentes hacia un bus de datos basado en Ethernet. Además, las nuevas arquitecturas de sistema se pueden basar en dicha interfaz. Tal API ofrecerá además una interfaz de programación normalizada para uso con un bus de datos basado en Ethernet.

La API basada en Ada define los siguientes tipos (objetos):

- **EtherAddress**: Representa una dirección Ethernet.
- **EtherSocket**: Implementa un zócalo en el lado cliente que utiliza las capacidades de transferencia de datos asegurada de SSCOPMCE. Los datos se transportan en una o más PDU con datos secuenciados (SD, *sequenced-data*) dentro de tramas Ethernet.
- **EtherTag**: Contiene los atributos asociados con el tipo tag (rótulo) 802.1 [22].
- **EtherServerSocket**: Implementa un zócalo en el lado servidor que utiliza las capacidades de transferencia de datos asegurada de SSCOPMCE. Los datos se transportan en una o más PDU con datos secuenciados (SD) dentro de tramas Ethernet.
- **Datagram**: Crea un datagrama haciendo referencia una PDU de datos de usuario (UD, *user data*) no numerados.
- **DatagramSocket**: Crea un zócalo para enviar o recibir un datagrama.
- **MulticastSocket**: Crea un zócalo multidifusión para enviar o recibir un datagrama. Los datos se transportan en una o más PDU de datos de usuario no numerados (UD). El funcionamiento en multidifusión se basa en el protocolo de registro multidifusión GARP (GMRP, *GARP multicast registration protocol*) [21].

La razón por la que se han definido sólo pocos tipos de objetos se basa en gran parte en la correspondencia muy racionalizada de las capas de protocolo permitidas en el anexo E/Q.2111. Desde la perspectiva de una definición, estos tipos, y las operaciones asociadas con estos tipos, están contenidos en el lote bus de datos Ethernet. Un piloto asociado con una tarjeta de interfaz de

red debe ser conforme con dicho lote. Desde la perspectiva de la implementación, estos tipos se diseñan como privados, y, como la especificación de operaciones asociadas, están fuera del alcance de esta Recomendación. Se ha efectuado todo esto para dar flexibilidad a la implementación y evolución de la API.

4 Resumen de la definición del lote Ada

Lo siguiente es un resumen del lote Bus de datos Ethernet:

```
package Ethernet Databus is

     type EtherAddress is private;
     type EtherAddresses is (POSITIVE range <>) of  EtherAddress;

     type EtherTag is private;
     type COS_TYPE is mod 2**3;
     type VLAN_TYPE is mod 2**12;

     type EtherSocket is private;
     type PORT_TYPE is mod 2**16;

     type EtherServerSocket is private;

     type Datagram is private;
     type BYTE is mod 2**8;
     type BYTE_ARRAY is array (POSITVE range <>) of BYTE;

     type DatagramSocket is private;

     type MulticastSocket is private;

     -- EtherAddress

     function getAddress(addr: EtherAddress) return STRING;
     function getOUI(addr: EtherAddress) return STRING;
     function getLocal(addr: EtherAddress) return STRING;
     function isGroupAddress(addr: EtherAddress) return BOOLEAN;
     function getLocalAddress return EtherAddress;
     function getLocalAddresses return EtherAddresses;

     -- EtherTag

     procedure makeEtherTag(cos: in COS_TYPE) return EtherTag;
     procedure makeEtherTag(vlan: in VLAN_TYPE) return EtherTag;
     procedure makeEtherTag(cos: in COS_TYPE;
                            cfi: in BOOLEAN;
                            vlan: in VLAN_TYPE)
                            return EtherTag;
     function get_cos(tag: EtherTag) return COS_TYPE;
     function get_cfi(tag: EtherTag) return BOOLEAN;
     function get_vlan(tag: EtherTag) return VLAN_TYPE;

     -- EtherSocket

     function makeethersocket(host: etheraddress;
                            port: port_type)
                            return ethersocket;
     function makeEtherSocket(host: EtherAddress;
                            tag: EtherTag;
                            port: PORT_TYPE)
                            return  EtherSocket;
     function makeEtherSocket(host: EtherAddress;
                            port: PORT_TYPE;
                            interface: EtherAddress;
                            localPort: PORT_TYPE)
                            return EtherSocket;
     function makeEtherSocket(host: EtherAddress;
                            port: PORT_TYPE;
                            tag: EtherTag;
```

```
                        interface: EtherAddress;
                        localPort: PORT_TYPE)
                                return EtherSocket;
function getEtherAddress(socket: EtherSocket)
                        return EtherAddress;
function getPort(socket: EtherSocket)  return PORT_TYPE;
function getLocalPort(socket: EtherSocket)  return PORT_TYPE;
function getlocaladdress(socket: ethersocket)
                        return etheraddress;
function getInputStream(socket: EtherSocket)
                        return STREAM_ACCESS;
function getOutputStream(socket: EtherSocket)
                        return STREAM_ACCESS;
procedure close(socket: in EtherSocket);

-- EtherServerSocket

function makeEtherServerSocket(port: PORT_TYPE)
                                return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                tag: EtherTag)
                                return EtherServerSocket;
function  makeEtherServerSocket(port: PORT_TYPE;
                                queueLength: POSITIVE)
                                return EtherServerSocket;
function  makeEtherServerSocket(port: PORT_TYPE;
                                queueLength: POSITIVE;
                                tag: EtherTag)
                                return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                queueLength: POSITIVE;
                                bindAddress: EtherAddress)
                                return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                queueLength: POSITIVE;
                                tag: EtherTag;
                                bindAddress: EtherAddress)
                                return EtherServerSocket;
function accept(socket: EtherServerSocket)
                                return EtherSocket;
procedure close(socket: in EtherServerSocket);
function getEtherAddress(socket: EtherServerSocket)
                                return EtherAddress;
function getLocalPort(socket: EtherServerSocket)
                                return PORT_TYPE;
function getTag(socket: EtherServerSocket) return EtherTag;

-- DATAGRAM

-- for receiving datagrams

function makeDatagram (buffer: BYTE_ARRAY;
                       length: POSITIVE)
                       return Datagram;

function makeDatagram (buffer: BYTE_ARRAY;
                       offset: NATURAL;
                       length: POSITIVE)
                       return Datagram;

-- for sending datagrams

function makeDatagram (data: BYTE_ARRAY;
                       offset: NATURAL;
                       length: POSITIVE)
                       return Datagram;

function makeDatagram (data: BYTE_ARRAY;
                       length: POSITIVE;
                       destination: EtherAddress;
```

```
                        port: PORT_TYPE)
                        return Datagram;

function getAddress(d: Datagram) return EtherAddress;
function getPort(d: Datagram) return PORT_TYPE;
function getData(d: Datagram) return BYTE_ARRAY;
function getLength(d: Datagram) return POSITIVE;
function getOffset(d: Datagram) return NATURAL;
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY);
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY;
                  offset: in NATURAL;
                  length: in POSITIVE);
procedure setAddress(d: in Datagram;
                     remote: in EtherAddress);
procedure setPort(d: in Datagram ;
                  port: in PORT_TYPE);
procedure setLength(d: in Datagram;
                    length: in POSITIVE);

-- DatagramSocket

function makeDatagramSocket return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                            tag: EtherTag)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                     address: EtherAddress)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                             tag: EtherTag;
                             address: EtherAddress;
                             return DatagramSocket;

procedure send(socket: in DatagramSocket;
               d: in Datagram);
procedure receive(socket: in DatagramSocket;
                  d: in out Datagram);
procedure close(socket: in DatagramSocket);
function getLocalPort(socket: DatagramSocket) return PORT_TYPE;
procedure connect(socket: in DatagramSocket;
             host: in EtherAddress;
                  port: in PORT_TYPE);
procedure disconnect(socket: in DatagramSocket);
function getPort(socket: DatagramSocket) return PORT_TYPE;
function getEtherAddress(socket: DatagramSocket)
                         return EtherAddress;
function getTag(socket: DatagramSocket) return EtherTag;

-- MulticastSocket

function makeMulticastSocket return MulticastSocket;
function makeMulticastSocket(port: PORT_TYPE)
                             return MulticastSocket;
function makeMulticastSocket(port: PORT_TYPE;
                             tag: EtherTag)
                             return MulticastSocket;

procedure joinGroup(socket: in MulticastSocket;
                    address: in EtherAddress);
procedure leaveGroup(socket: in MulticastSocket;
                     address: in EtherAddress);
procedure setInterface(socket: in MulticastSocket;
                    address: in EtherAddress);
```

```
     function getInterface(socket: MulticastSocket)
                          return EtherAddress;

     procedure send(socket: in MulticastSocket;
                    d: in Datagram);
     procedure receive(socket: in MulticastSocket;
                       d: in out Datagram);
     procedure close(socket: in MulticastSocket);
     function getLocalPort(socket: MulticastSocket)
                          return PORT_TYPE;
     procedure connect(socket: in MulticastSocket;
                       host: in EtherAddress;
                       port: in PORT_TYPE);
     procedure disconnect(socket: in MulticastSocket);
     function getPort(socket: MulticastSocket) return PORT_TYPE;
     function getEtherAddress(socket: MulticastSocket)
                              return EtherAddress;
     function getTag(socket: MulticastSocket) return EtherAddress;

     -- Exceptions

     UnknownHostException: exception;
     IlegalArgumentException: exception;
     BindException: exception;
     IOException: exception;
     SocketException: exception;

private

     -- Implementation dependent

end Ethernet Databus;
```

5 Descripción de la definición del paquete Ada

Lo siguiente es una descripción detallada de cada una de las subrutinas:

5.1 Subrutinas EtherAddress

getAddress

```
function getAddress(addr: EtherAddress) return STRING;
```

Retorna una dirección Ethernet completa de 48 bits.

Parámetros:

addr – dirección Ethernet

Retorna:

una sola dirección Ethernet, como una cadena que describe los octetos en notación hexadecimal, por ejemplo, "3407A4CE0000".

getOUI

```
function getOUI(addr: EtherAddress) return STRING;
```

Retorna los primeros tres octetos de una dirección Ethernet: el Identificador Universal de Organización.

Parámetros:

addr – dirección Ethernet

Retorna:

la parte OUI de la dirección, como una cadena que describe los octetos en notación hexadecimal, por ejemplo, "3407A4".

getLocal

```
function getLocal(addr: EtherAddress) return STRING;
```

Retorna los últimos tres octetos de una dirección Ethernet: la parte asignada localmente.

Parámetros:

addr – dirección Ethernet

Retorna:

la parte asignada localmente de la dirección, como una cadena que describe los octetos en notación hexadecimal, por ejemplo, "CE0000".

isGroupAddress

```
function isGroupAddress(addr: EtherAddress) return BOOLEAN;
```

Determina si la dirección Ethernet es una dirección de grupo, si el primer bit del octeto de orden superior es cero.
Parámetros:
addr – dirección Ethernet
Retorna:
true (verdadero) si la dirección es una dirección de grupo, de lo contrario, false (falso).

getLocalAddress

```
function getLocalAddress(addr: EtherAddress) return STRING;
```

Retorna la dirección asociada con el anfitrión local.
Parámetros:
addr – dirección Ethernet local
Retorna:
Una dirección Ethernet
Opone la excepción: UnknownHostException
si no se pudo encontrar una dirección Ethernet para el anfitrión.

getAllHostAddresses

```
function getLocalAddresses return EtherAddresses;
```

Retorna un array de las direcciones asociadas a un anfitrión con múltiples regresos al punto de partida (*multi-homed*).
Retorna:
Un array de direcciones Ethernet
Opone la excepción: UnknownHostException
si no se pudo encontrar una dirección Ethernet para el anfitrión.

5.2 Subrutinas EtherTag

makeEtherTag

```
function makeEtherTag(cos: in COS_TYPE) return EtherTag;
```

Establece el campo CoS del rótulo 802.1. El campo VLAN se fija a un valor por defecto de todos ceros. El campo CFI se fija a un valor por defecto de cero.
Parámetros:
cos – clase de servicio

makeEtherTag

```
function makeEtherTag(vlan: in VLAN_TYPE) return EtherTag;
```

Establece el campo VLAN del rótulo 802.1. El campo CoS se fija a un valor por defecto de todos ceros. El campo CFI se fija a un valor por defecto de cero.
Parámetros:
vlan – identificador vlan

makeEtherTag

```
function makeEtherTag(cos: in COS_TYPE;
                      cfi: in BOOLEAN;
                           vlan: in VLAN_TYPE)
                              return EtherTag;
```

Establece todos los campos del rótulo 802.1.
Parámetros:
cos – clase de servicio
cfi – identificador de formato canónico
vlan – identificador de LAN virtual

get_cos

```
function get_cos(tag: EtherTag) return COS_TYPE;
```

Retorna el valor del campo CoS en el rótulo 802.1.
Parámetros:
tag. – rótulo (tag) 802.1
Retorna:
la clase de servicio

get_cos

```
function get_cfi(tag: EtherTag) return BOOLEAN;
```

Retorna el valor del campo CFI en el rótulo 802.1.

Parámetros:
tag – rótulo (tag) 802.1
Retorna:
el identificador de formato canónico

get_vlan

```
function get_vlan(tag: EtherTag) return VLAN_TYPE;
```

Retorna el valor del campo VLAN en el rótulo 802.1.
Parámetros:
tag – rótulo (tag) 802.1
Retorna:
el identificador vlan

5.3 Subrutinas EtherSocket

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         port: PORT_TYPE)
                        return EtherSocket;
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión.
Parámetros:
host – dirección del anfitrión de destino
port – puerto de destino
Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el zócalo.

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         tag: EtherTag;
                        port: PORT_TYPE)
                        return EtherSocket;
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión.
Parámetros:
host – dirección del anfitrión de destino
tag – rótulo (tag) 802.1
port – puerto de destino
Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el zócalo.

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                              port: PORT_TYPE;
                              interface: EtherAddress;
                              localPort: PORT_TYPE)
                              return EtherSocket;
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión. Se conecta al anfitrión y puerto especificados en los primeros dos argumentos, y desde la interfaz de red local y el puerto especificados en los últimos dos argumentos.
Parámetros:
host – dirección del anfitrión de destino
port – puerto de destino
interface – dirección local
localPort – puerto local
Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el zócalo.

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                              port: PORT_TYPE,
                              tag: EtherTag;
                               interface: EtherAddress;
```

```
                                localPort: PORT_TYPE)
                                returns EtherSocket;
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión. Se conecta al anfitrión y al puerto especificados en los primeros dos argumentos, y de la interfaz de la red local y el puerto especificados en los últimos dos argumentos.

Parámetros:
host – dirección del anfitrión de destino
port – puerto de destino
tag – rótulo (tag) 802.1
interface – dirección local
localPort – puerto local

Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el zócalo.

getEtherAddress

```
function getEtherAddress(socket: EtherSocket)
                                return EtherAddress;
```

Retorna el anfitrión distante a que está conectado el zócalo o, si la conexión está ahora cerrada, a qué anfitrión estaba conectado el zócalo cuando existía la conexión.

Parámetros:
socket – zócalo Ethernet

Retorna:
la dirección Ethernet distante a la cual está conectado el zócalo.

getPort

```
function getPort(socket: EtherAddress) return PORT_TYPE;
```

Retorna el puerto a que está, estaba o estará conectado el zócalo en el anfitrión distante.

Parámetros:
socket – zócalo Ethernet

Retorna:
el puerto a que está conectado en el anfitrión distante.

getLocalPort

```
function getLocalPort(socket: EtherAddress) return PORT_TYPE;
```

Retorna el número de puerto para el anfitrión local.

Parámetros:
socket – zócalo Ethernet

Retorna:
el número de puerto local.

getLocalAddress

```
function getLocalAddress(socket: EtherAddress)
                                return EtherAddress;
```

Obtiene la dirección local a la cual está vinculado el zócalo.

Parámetros:
socket – zócalo Ethernet

Retorna:
la dirección local.

getInputStream

```
function getInputStream(socket: EtherSocket)
                               return STREAM_ACCESS;
```

Retorna un tren de entrada para este zócalo.

Parámetros:
socket – zócalo Ethernet

Retorna:
una referencia a un tren de entrada para leer octetos desde esté zócalo.

Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el tren de salida.

getOutputStream

```
function getOutputStream(socket: EtherSocket)
                                return STREAM_ACCESS;
```

Retorna un tren de salida para este zócalo.

Parámetros:
socket – zócalo Ethernet

Retorna:
una referencia a un tren de salida para escribir octetos en este zócalo.
Opone la excepción: IOException
si aparece un error de entrada/salida mientras se crea el tren de salida.

close

```
procedure close(socket: in EtherSocket);
```

Cierra el zócalo.
Parámetros:
socket – zócalo Ethernet
Opone la excepción: IOException
si aparece un error de entrada/salida mientras se cierra el zócalo.

5.4 Subrutinas EtherServerSocket

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE)
                                  return EtherServerSocket;
```

Crea un zócalo servidor en el puerto especificado por el argumento.
Parámetros:
port – puerto local
Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                                 tag: EtherTag)
                               return EtherServerSocket;
```

Crea un zócalo servidor, en el puerto, y con base en el rótulo, especificados por los argumentos.
Parámetros:
port – puerto local
tag – rótulo (tag) 802.1
Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                           queueLength: POSITIVE)
                                 return EtherServerSocket;
```

Crea un zócalo servidor, en el puerto especificado con la longitud de cola especificada (en octetos) para peticiones de conexión entrantes.
Parámetros:
port – puerto local
queueLength – longitud de cola
Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

makeEtherServerSocket

```
function  makeEtherServerSocket(port: PORT_TYPE;
                            queueLength: POSITIVE;
                                    tag: EtherTag)
                                  return EtherServerSocket;
```

Crea un zócalo servidor en el puerto especificado con la longitud de cola especificada (en octetos) para peticiones de conexión entrantes.
Parámetros:
port – puerto local
queueLength – longitud de cola
tag – rótulo (tag) 802.1
Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                               queueLength: POSITIVE;
                                indAddress: EtherAddress)
                                      return EtherServerSocket;
```

Crea un zócalo servidor en el puerto especificado con la longitud de cola especificada para retener las peticiones de conexión entrantes; el zócalo vincula solamente a la dirección Ethernet especificada.

Parámetros:
port – puerto local
queueLength – longitud de cola
bindAddress – dirección a que se vincula

Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                               queueLength: POSITIVE;
                                       tag: EtherTag;
                               bindAddress: EtherAddress)
                                      return EtherServerSocket;
```

Crea un zócalo servidor en el puerto especificado con la longitud de cola y el rótulo especificados para retener las peticiones de conexión entrantes; el zócalo vincula solamente a la dirección Ethernet especificada.

Parámetros:
port – puerto local
queueLength – longitud de cola
bindAddress – dirección a que se vincula
tag – rótulo (tag) 802.1

Opone la excepción: BindException
si el zócalo no se puede crear y vincular al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

accept

```
function accept(socket: EtherServerSocket) return EtherSocket;
```

Estar a la escucha de que se efectúe una conexión a este zócalo y aceptarla. El método bloquea hasta que se efectúa una conexión.

Parámetros:
socket – zócalo servidor Ethernet

Opone la excepción: IOException
si aparece un error de entrada/salida mientras se espera una conexión.

close

```
procedure close(socket: in EtherServerSocket);
```

Cierra este zócalo.

Parámetros:
socket – zócalo servidor Ethernet

Opone la excepción: IOException
si aparece un error de entrada/salida mientras se cierra el zócalo.

getEtherAddress

```
function getEtherAddress(socket: EtherServerSocket)
 return EtherAddress;
```

Retorna la dirección local de este zócalo servidor.

Parámetros:
socket – zócalo servidor Ethernet

Retorna:
la dirección local.

getLocalPort

```
function getLocalPort(socket: EtherServerSocket) return PORT_TYPE;
```

Determina el puerto local que se está escuchando.

Parámetros:
socket – zócalo servidor Ethernet

Retorna:
el número de puerto local.

getTag

```
function getTag(socket: EtherServerSocket) return EtherTag;
```

Retorna el rótulo de este zócalo servidor.
Parámetros:
socket – zócalo servidor Ethernet
Retorna:
El rótulo (tag); de lo contrario, se retorna "nulo" si ningún rótulo está asociado con este zócalo.

5.5 Subrutinas de datagrama

makeDatagram

```
function makeDatagram(buffer: BYTE_ARRAY;
          length: POSITIVE)
                        return Datagram;
```

Crea un objeto datagrama para la recepción de datos. Los datos de datagrama recibidos se almacenan en `buffer` hasta que se llena la PDU UD apropiada o hasta que se han escrito `length` octetos en la memoria tampón.
Parámetros:
buffer – array de octetos
length – número de octetos
Opone la excepción: IllegalArgumentException
si la longitud especificada desborda la memoria tampón.

makeDatagram

```
function makeDatagram(buffer: BYTE_ARRAY;
                  offset: NATURAL;
                     length: POSITIVE)
                              return Datagram;
```

Crea un objeto datagrama para la recepción de datos. Los datos de datagrama recibidos se almacenan en `buffer`, comenzando en `buffer[offset]`, hasta que se llena la PDU UD apropiada o hasta que se han escrito `length` octetos en la memoria tampón.
Parámetros:
buffer – array de octetos
offset – desplazamiento, en octetos
length – número de octetos
Opone la excepción: IllegalArgumentException
si la longitud especificada desborda la memoria tampón.

makeDatagram

```
function makeDatagram(data: BYTE_ARRAY;
          offset: NATURAL;
          length: POSITIVE)
                        return Datagram;
```

Crea un datagrama para el envío de datos. El datagrama se llena con `length` octetos de datos. La `destination` apunta al anfitrión al que se debe entregar el datagrama; el `port` es el puerto de destino en ese anfitrión.
Parámetros:
data – array de octetos
length – número de octetos
destination – dirección de destino
port – puerto de destino
Opone la excepción: IllegalArgumentException
si la longitud es mayor que el tamaño del array de datos.

makeDatagram

```
function makeDatagram(data: BYTE_ARRAY;
          length: POSITIVE;
           destination: EtherAddress;
                      port: PORT_TYPE)
                        return Datagram;
```

Crea un datagrama para el envío de datos. El datagrama se llena con `length` octetos de datos comenzando en `offset`. La `destination` apunta al anfitrión al que se debe entregar el datagrama; el `port` es el puerto de destino en ese anfitrión.
Parámetros:
data – array de octetos
offset – desplazamiento, en octetos
length – número de octetos
destination – dirección de destino
port – puerto de destino
Opone la excepción: IllegalArgumentException
si la longitud es mayor que el tamaño del array de datos.

getAddress

```
function getAddress(d: Datagram) return EtherAddress;
```

Retorna la dirección del anfitrión distante del que se recibió el datagrama.
Parámetros:
d – datagrama
Retorna:
la dirección del anfitrión distante.

getPort

```
function getPort(d: Datagram) return PORT_TYPE;
```

Retorna el puerto distante del que se recibió el datagrama.
Parámetros:
d – Datagrama
Retorna:
el número de puerto distante.

getData

```
function getData(d: Datagram) return BYTE_ARRAY;
```

Retorna un array de octetos que contiene los datos del datagrama.
Parámetros:
d – datagrama
Retorna:
array de octetos.

getLength

```
function getLength(d: Datagram) return POSITIVE;
```

Retorna el número de octetos en el datagrama.
Parámetros:
d – datagrama
Retorna:
número de octetos.

getOffset

```
function getOffset(d: Datagram) return NATURAL;
```

Retorna el punto, en el array retornado por `getData`, en el que comienzan los datos del datagrama.
Parámetros:
d – datagrama
Retorna:
punto, en el array, en el que comienzan los datos.

setData

```
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY);
```

Cambia la cabida útil del datagrama.
Parámetros:
d – datagrama
data – array de octetos

setData

```
procedure setData(d: in Datagram
            data: in BYTE_ARRAY;
            offset: in NATURAL;
            length: in POSITIVE);
```

Envía datos en piezas de tamaño comenzando en `offset`.

Parámetros:
d – datagrama
data – array de octetos
offset – desplazamiento
length: – tamaño de la pieza ("trozo") de datos.

setAddress

```
procedure setAddress(d: in Datagram;
                     remote: in EtherAddress);
```

Cambia la dirección de destino de un datagrama.
Parámetros:
d – datagrama
remote – dirección Ethernet distante

setPort

```
procedure setPort(d: in Datagram;
                  port: in PORT_TYPE);
```

Cambia el puerto al que está dirigido un datagrama.
Parámetros:
d – datagrama
port – puerto de destino

setLength

```
procedure setLength(d: in Datagram;
                    length: in POSITIVE);
```

Cambia el número de octetos en la memoria tampón interna de manera que los datagramas no sean truncados entre recepciones.
Parámetros:
d – datagrama
lengh – longitud en octetos

5.6 Subrutinas DatagramSocket

makeDatagramSocket

```
function makeDatagramSocket return DatagramSocket;
```

Crea un zócalo vinculado a un puerto anónimo. Para recibir datagramas se puede utilizar el mismo zócalo utilizado por un servidor para devolverlos.
Opone la excepción: SocketException
si no se puede crear el zócalo.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE)
                                  return DatagramSocket;
```

Crea un zócalo que está a la escucha de los datagramas entrantes en un determinado puerto especificado por el argumento port.
Parámetros:
port – puerto a la escucha
Opone la excepción: SocketException
si no se puede crear el zócalo.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                            tag: EtherTag)
                            return DatagramSocket;
```

Crea un zócalo que está a la escucha de los datagramas entrantes en un determinado puerto, especificado por el argumento port, y un determinado rótulo, especificado por el argumento tag.
Parámetros:
port – puerto a la escucha
tag – rótulo 802.1
Opone la excepción: SocketException
si no se puede crear el zócalo.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                         address: EtherAddress)
                            return DatagramSocket;
```

Crea un zócalo que está a la escucha de los datagramas entrantes en un puerto y una interfaz de red específicos. Este constructor es particularmente útil para un anfitrión con múltiples regresos al punto de partida.
Parámetros:
port – puerto a la escucha
address – dirección Ethernet del anfitrión
Opone la excepción: SocketException
si no se puede crear el zócalo.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                  tag: EtherTag;
                          address: EtherAddress)
                             return DatagramSocket;
```

Crea un zócalo que está a la escucha de los datagramas entrantes en un puerto, un rótulo y una interfaz de red específicos. Este constructor es particularmente útil para un anfitrión con múltiples regresos al punto de partida.
Parámetros:
port – puerto a la escucha
tag – rótulo 802.1
address – dirección Ethernet del anfitrión
Opone la excepción: SocketException
si no se puede crear el zócalo.

send

```
procedure send(socket: DatagramSocket;
                       d: in Datagram);
```

Envía un solo datagrama dp por la red utilizando este zócalo de datagrama.
Parámetros:
socket – zócalo de datagrama
d – objeto datagrama
Opone la excepción: IOException
si el datagrama que se va a enviar es más grande que el que puede ser soportado por el software nativo.

receive

```
procedure receive(socket: in DatagramSocket;
                          d: in out Datagram);
```

Recibe un solo datagrama de la red y lo almacena en el datagrama d.
Parámetros:
socket – zócalo de datagrama
d – objeto datagrama
Opone la excepción: IOException
si surge un problema en la recepción de los datos.

close

```
procedure close(socket: in DatagramSocket);
```

Libera el puerto ocupado por el zócalo.
Parámetros:
socket – zócalo de datagrama.

getLocalPort

```
function getLocalPort(socket: DatagramSocket)
                          return PORT_TYPE;
```

Retorna el puerto local en el cual el zócalo está a la escucha.
Parámetros:
socket – zócalo de datagrama
Retorna:
el puerto local.

connect

```
procedure connect(socket: DatagramSocket;
                       host: in EtherAddress;
                       port: in PORT_TYPE);
```

Habilita la capacidad para enviar datagramas al anfitrión distante especificado, y recibir datagramas del anfitrión distante especificado, en el puerto distante especificado.

Parámetros:
socket – zócalo de datagrama
host – dirección Ethernet
port – puerto distante.

disconnect

```
procedure disconnect(socket: in DatagramSocket);
```

Inhabilita la capacidad del zócalo de manera que pueda enviar datagramas a cualquier anfitrión y puerto, y recibirlos de cualquiera de éstos.
Parámetros:
socket – zócalo de datagrama.

getPort

```
function getPort(socket: MulticastSocket) return PORT_TYPE;
```

Retorna el puerto distante al cual está conectado el zócalo.
Parámetros:
socket – zócalo de datagrama
Retorna:
el puerto distante utilizado por la conexión; de lo contrario, se retorna un nulo si el zócalo no está conectado.

getEtherAddress

```
function getEtherAddress(socket: MulticastSocket)
                              return EtherAddress;
```

Retorna la dirección del anfitrión distante al cual está conectado el zócalo.
Parámetros:
socket – zócalo de datagrama
Retorna:
la dirección del anfitrión distante; de lo contrario, se retorna un nulo si el zócalo no está conectado.

getTag

```
function getTag(socket: MulticastSocket) return EtherTag;
```

Retorna el rótulo (tag) asociado con el zócalo.
Parámetros:
socket – zócalo de datagrama
Retorna:
el rótulo; de lo contrario, se retorna un nulo si ningún rótulo está asociado con este zócalo.

5.7 Subrutinas MulticastSocket

makeMulticastSocket

```
function makeMulticastSocket return MulticastSocket;
```

Crea un zócalo multidifusión vinculado a un puerto anónimo. Un destinatario contesta al mismo puerto.
Opone la excepción: SocketException
si no se puede crear el zócalo.

makeMulticastSocket

```
function makeMulticastSocket(port: PORT_TYPE)
                                 return MulticastSocket;
```

Crea un zócalo multidifusión en un determinado puerto.
Parámetros:
port – puerto de origen
Opone la excepción: SocketException
si no se puede crear el zócalo, por ejemplo, si el puerto ya se está utilizando.

makeMulticastSocket

```
function makeMulticastSocket(port: PORT_TYPE;
                              tag: EtherTag)
                             return MulticastSocket;
```

Crea un zócalo multidifusión en un determinado puerto utilizando un rótulo especificado.
Parámetros:
port – puerto de origen
tag – rótulo 802.1
Opone la excepción: SocketException
si no se puede crear el zócalo, por ejemplo, si el puerto ya se está utilizando.

joinGroup

```
procedure joinGroup(socket: MuticastSocket;
                    address: in EtherAddress);
```

Una vez creado un zócalo multidifusión, este método permite incorporarlo a un grupo multidifusión
Parámetros:
socket – zócalo multidifusión
address – dirección Ethernet
Opone la excepción: IOException
si la dirección no es una dirección de grupo.

leaveGroup

```
procedure leaveGroup(socket: MutlicastSocket;
                     address: in EtherAddress);
```

Una vez que un zócalo multidifusión se ha incorporado a un grupo, puede separarse del grupo invocando este método.
Parámetros:
socket – zócalo multidifusión
address – dirección Ethernet
Opone la excepción: IOException
si la dirección no es una dirección de grupo.

setInterface

```
procedure setInterface(socket: MultiastSocket;
                       address: in EtherAddress);
```

Asocia una determinada interfaz de red para uso de multidifusión en un anfitrión con múltiples regresos al punto de partida.
Parámetros:
socket – zócalo multidifusión
address – dirección Ethernet
Opone la excepción: SocketException
si la dirección no existe en la máquina local.

getInterface

```
function getInterface(socket: MulticastSocket)
                       return EtherAddress;
```

Obtiene la dirección de la interfaz que se está utilizando.
Retorna:
la dirección que se está utilizando.

send

```
procedure send(socket: in MulticastSocket;
                     d: in Datagram);
```

Envía un solo datagrama dp por la red utilizando este zócalo de datagrama.
Parámetros:
socket – zócalo multidifusión
d – objeto datagrama
Opone la excepción: IOException
si el datagrama que se va a enviar es mayor que el que puede ser soportado por el software nativo.

receive

```
procedure receive(socket: in MulticastSocket;
                        d: in out Datagram);
```

Recibe un solo datagrama de la red y lo almacena en el datagrama d.
Parámetros:
d – objeto datagrama
Opone la excepción: IOException
si surge un problema en la recepción de los datos.

close

```
procedure close(socket: in MulticastSocket);
```

Libera el puerto ocupado por el zócalo.
Parámetros:
socket – zócalo multidifusión

getLocalPort

```
function getLocalPort(socket: MulticastSocket)
                       return PORT_TYPE;
```

Retorna el puerto local en el que el zócalo está a la escucha.
Parámetros:
socket – zócalo multidifusión
Retorna:
el puerto local.

connect

```
procedure connect(socket: in MulticastSocket;
                  host: in EtherAddress;
                  port: in PORT_TYPE);
```

Habilita la capacidad para enviar datagramas a los anfitriones distantes especificados, y recibir datagramas de los anfitriones distantes especificados, en el puerto distante especificado.
Parámetros:
socket – zócalo multidifusión
host – dirección de anfitrión distante
puerto – puerto distante

disconnect

```
procedure disconnect(socket: in MulticastSocket);
```

Inhabilita la capacidad del zócalo de manera que pueda enviar datagramas a cualquier anfitrión y puerto, y recibirlos de cualquiera de éstos.
Parámetros:
socket – zócalo multidifusión

getPort

```
function getPort(socket: MulticastSocket) return PORT_TYPE;
```

Retorna el puerto distante al cual está conectado el zócalo.
Parámetros:
socket – zócalo multidifusión
Retorna:
el puerto distante utilizado por la conexión; de lo contrario, retorna -1 si el zócalo no está conectado.

getEtherAddress

```
function getEtherAddress(socket: MulticastSocket)
                         return EtherAddress;
```

Retorna la dirección del anfitrión distante al cual está conectado el zócalo.
Parámetros:
socket – zócalo multidifusión
Retorna:
la dirección del anfitrión distante; de lo contrario, retorna nulo si el zócalo no está conectado.

getTag

```
function getTag(socket: MulticastSocket) return EtherTag;
```

Retorna el rótulo (tag) asociado con el zócalo.
Parámetros:
socket – zócalo multidifusión
Retorna:
el rótulo; de lo contrario, retorna nulo si ningún rótulo está asociado con este zócalo.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	**Conmutación y señalización**
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Ginebra, 2002