UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Q.2111

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

Enmienda 3
(10/2003)

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

Enmienda 3: Interfaz de programación de aplicaciones para SSCOPMCE por Ethernet y número de puerto UDP

Recomendación UIT-T Q.2111 (1999) – Enmienda 3

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 DE LOS SISTEMAS DE SEÑALIZACIÓN N.° 4, 5, 6, R1 Y R2	Q.120–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 de transferencia asíncrono de la RDSI-BA – Protocolo con conexión específico de servicio en un entorno multienlace y sin conexión

Enmienda 3

Interfaz de programación de aplicaciones para SSCOPMCE por Ethernet y número de puerto UDP

Resumen

En esta enmienda se describe una interfaz de programación de aplicaciones (API) en lenguaje de programación C++ para el dispositivo de protocolo descrito en el anexo E de la Rec. UIT-T Q.2111 (SSCOP en un entorno multienlace y sin conexión cuando funciona por Ethernet).

Además, se asigna el número de puerto UDP que se utilizará con SSCOPMCE por encima de UDP en configuraciones como las especificadas en los anexos C y D. Este número de puerto UDP puede utilizarse en conjunto con un número de puerto fuera de la escala de valores dinámica/privada (valores de 49152 a 65535).

Orígenes

La enmienda 3 a la Recomendación UIT-T Q.2111 (1999) fue aprobada el 14 de octubre de 2003 por la Comisión de Estudio 11 (2001-2004) del UIT-T por el procedimiento de la Recomendación UIT-T A.8.

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.

La observancia de esta Recomendación es voluntaria. Ahora bien, la Recomendación puede contener ciertas disposiciones obligatorias (para asegurar, por ejemplo, la aplicabilidad o la interoperabilidad), por lo que la observancia se consigue con el cumplimiento exacto y puntual de todas las disposiciones obligatorias. La obligatoriedad de un elemento preceptivo o requisito se expresa mediante las frases "tener que, haber de, hay que + infinitivo" o el verbo principal en tiempo futuro simple de mandato, en modo afirmativo o negativo. El hecho de que se utilice esta formulación no entraña que la observancia se imponga a ninguna de las partes.

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

Página

1) Cláusula 2.2 – Bibliografía 1

2) Cláusula 5.3 – Modos de funcionamiento 1

3) Cláusula C.3.2.1 – Descripción de la interfaz superior UDP 1

4) Cláusula D.3.2.1 – Descripción de la interfaz superior UDP 1

5) Anexo G 2

Anexo G – API C++ para SSCOPMCE por Ethernet 2

G.1 Introducción 2

G.2 Objetivos de la API de bus de datos Ethernet 2

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

G.4 Resumen de la definición de la colección de programas C++ 3

G.5 Descripción de la definición de la colección de programas C++ 5

Recomendación UIT-T Q.2111

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

Enmienda 3

Interfaz de programación de aplicaciones para SSCOPMCE por Ethernet y número de puerto UDP

1) Cláusula 2.2 – Bibliografía

Añádase la siguiente referencia:

[24] ISO/CEI 14882:1998, *Programming languages – C++.*

2) Cláusula 5.3 – Modos de funcionamiento

Modifíquese la última oración al final del párrafo inmediatamente a continuación de la figura 2 y añádase otra oración para que resulte lo siguiente:

Además, en el anexo F se propone una interfaz de programación de aplicaciones (API) para SSCOPMCE por Ethernet, especificada en Ada. Asimismo, en el anexo G se propone una interfaz de programación de aplicaciones (API, *application programming interface*) para SSCOPMCE por Ethernet, especificada en C++.

3) Cláusula C.3.2.1 – Descripción de la interfaz superior UDP

i) Modifíquese la parte final de "Puerto de origen" y añádase otra oración para que resulte lo siguiente:

Cuando se incluya, el valor numérico de SSCOPMCE por encima de UDP es un "VALOR QUE DEBE SER ASIGNADO POR IANA" o un valor de la escala de valores dinámicos/privados (valores de 49152 a 65535), conforme al entorno donde se utilice SSCOPMCE.

ii) Modifíquese el final de "Puerto de destino" y añádase otra oración para que resulte lo siguiente:

El valor numérico de SSCOPMCE por encima de UDP es un "VALOR QUE DEBE SER ASIGNADO POR IANA" o un valor de la escala de valores dinámicos/privados (valores de 49152 a 65535), conforme al entorno donde se utilice SSCOPMCE.

4) Cláusula D.3.2.1 – Descripción de la interfaz superior UDP

i) Modifíquese el final de "Puerto de origen" y añádase otra oración para que resulte lo siguiente:

Si está incluido, el valor numérico de SSCOPMCE por encima de UDP es un "VALOR QUE DEBE SER ASIGNADO POR IANA" o un valor de la escala de valores dinámicos/privados (valores de 49152 a 65535), conforme al entorno donde se utilice SSCOPMCE.

ii) Modifíquese el final de "Puerto de destino" y añádase otra oración para que resulte lo siguiente:

El valor numérico de SSCOPMCE por encima de UDP es un "VALOR QUE DEBE SER ASIGNADO POR IANA" o un valor de la escala de valores dinámicos/privados (valores de 49152 a 65535), conforme al entorno donde se utilice SSCOPMCE.

5) Anexo G

Añádase un nuevo anexo G (API para SSCOPMCE por Ethernet) de la siguiente manera:

Anexo G

API C++ para SSCOPMCE por Ethernet

G.1 Introducción

En el anexo E se describe 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 la Recomendación Q.2111 y su 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 aplicaciones (API) al SAP. La razón para especificar una API es instar a los proveedores de herramientas de desarrollo y/o sistemas operativos en tiempo real a ofrecer una interfaz normalizada, abierta y familiar para que los desarrolladores de programas informáticos se beneficien de las capacidades de red ofrecidas por un bus de datos basado en Ethernet.

G.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 los programadores 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 (sockets), los cuales se han utilizado ampliamente en la mayoría de las API de las redes existentes para ordenadores personales 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 mecanismos para el tratamiento de las excepciones, a fin de gestionar los errores que aparecen en curso de ejecución.

G.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 C++. La selección de C++ está basada en su creciente utilización en sistemas aeroespaciales y de defensa, uno de los sectores de aplicación que motivaron la especificación del anexo E. Por consiguiente, una API basada en C++ 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 API que ofrecerá además una interfaz de programación normalizada para uso con un bus de datos basado en Ethernet.

La API basada en C++ 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 de 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 a 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 (UD) no numerados. 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 que se señala en el anexo E. 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 controlador asociado con una tarjeta de interfaz de red debe ser conforme a dicho lote. Desde la perspectiva de la implementación, estos tipos se designan 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.

G.4 Resumen de la definición de la colección de programas C++

A continuación se presenta un resumen de la definición de la colección de programas del bus de datos Ethernet:

Library <EthernetDatabus>

```
class EtherAddress {

public:
        char *addr;
        char *getOUI();
        char *getLocal();
        boolean isGroupAddress();
        char *getHostAddress();
        char **getAllHostAddresses();
        };
```

```
class EtherTag {

public:
      EtherTag(int cos);
      EtherTag(double vlan);
      EtherTag(int cos, boolean cfi, double vlan);
      int get_cos();
      boolean get_cfi();
      double get_vlan();
        };


class EtherSocket {

public:
        EtherSocket(const EtherAddress &host, int port);
        EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherAddress &interface, int localPort);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherTag &tag, const EtherAddress &interface,
                    int localPort);
        EtherAddress getEtherAddress();
        int getPort();
        int getLocalPort();
        EtherAddress getLocalAddress();
        istream &getInputStream();
        ostream &getOutputStream();
        void close();
        };


class EtherServerSocket{

public:
        EtherServerSocket(int port);
        EtherServerSocket(int port, int queueLength);
        EtherServerSocket(int port, int queueLength, const EtherAddress
                          &bindAddress);
        EtherSocket accept();
        void close();
        EtherAddress getEtherAddress();
        int getLocalPort();
        };

class Datagram {

public:

        // for receiving datagrams
        Datagram(unsigned char *buffer[ ], int length);
        Datagram(unsigned char *buffer[ ], int offset, int length);

        // for sending datagrams
        Datagram(unsigned char *data[ ], int length, const EtherAddress
                 &destination, int port);
        Datagram(unsigned char *data[ ], int offset, int length,
                 const EtherAddress &destination, int port);
```

```
        EtherAddress getAddress();
        int getPort();
        unsigned char **getData();
        int getLength();
        int getOffset();
        setData(unsigned char **data);
        setData(unsigned char **data, int offset, int length);
        void setAddress(const EtherAddress &remote);
        void setPort(int port);
        void setLength(int length);
        };

class DatagramSocket {

public:
        DatagramSocket();
        DatagramSocket(int port);
        DatagramSocket(int port, const EtherAddress &address);
        DatagramSocket(int port, const EtherAddress &address,
        const EtherTag &tag);

        void send(const Datagram &d);
        void receive(const Datagram &d);
        void close();
        void getLocalPort();
        void connect(const EtherAddress &host, int port);
        void disconnect();
        int getPort();
        EtherAddress getEtherAddress();
        };

class MulticastSocket:  public DatagramSocket {

public:
        MulticastSocket();
        MulticastSocket(int port);
        void joinGroup(const EtherAddress &address);
        void leaveGroup(const EtherAddress &address);
        void send(const Datagram &d);
        void setInterface(const EtherAddress &address);
        EtherAddress getInterface();
        };
```

G.5 Descripción de la definición de la colección de programas C++

A continuación se presenta una descripción detallada de cada una de las clases y de las funciones de sus miembros asociados:

G.5.1 Clase EtherAddress

Esta clase representa una dirección Ethernet.

```
class EtherAddress {

public:
       char *addr;
       char *getOUI();
       char *getLocal();
       boolean isGroupAddress();
       char *getHostAddress();
       char **getAllHostAddresses();
       };
```

Variables

addr

```
char *addr;
```

Los seis bytes de una dirección Ethernet, con el byte de orden superior en primer lugar. El formato de la dirección es una cadena que describe los bytes en notación hexadecimal, por ejemplo, "347B0046A8CE".

Métodos

getOUI

```
char *getOUI();
```

Devuelve los tres primeros bytes de una dirección Ethernet: identificador universal de organización.
Devuelve:
La parte OUI de la dirección, como una cadena que describe los bytes en notación hexadecimal, por ejemplo, "347B00".

getLocal

```
char *getLocal();
```

Devuelve los últimos tres bytes de una dirección Ethernet: parte asignada localmente.
Devuelve:
La parte de la dirección asignada localmente, como una cadena que describe los bytes en notación hexadecimal, por ejemplo, "46A8CE".

isGroupAddress

```
boolean isGroupAddress();
```

Determina si la dirección Ethernet es una dirección de grupo, si el primer bit del byte de orden superior es cero.
Devuelve:
Verdadero, si se trata de una dirección de grupo, falso en caso contrario.

getHostAddress

```
char *getHostAddress();
```

Devuelve la dirección Ethernet numérica asociada con un anfitrión.
Devuelve:
Una dirección Ethernet simple, como una cadena que describe los bytes en notación hexadecimal, por ejemplo, "3487A8CE".
Opone la excepción: UnknownHostException
si no se pudo encontrar una dirección Ethernet para el anfitrión.

getAllHostAddresses

```
char **getAllHostAddresses();
```

Devuelve un conjunto de direcciones asociadas con un anfitrión multienlaces.
Devuelve:
Un conjunto de direcciones Ethernet, con un puntero a la primera dirección.
Opone la excepción: UnknownHostException
si no se pudo encontrar una dirección Ethernet para el anfitrión.

G.5.2 Clase EtherTag

Esta clase incluye los atributos asociados con el tipo de rótulo 802.1.

```
class EtherTag {

public:
      EtherTag(int cos);
      EtherTag(double vlan);
      EtherTag(int cos, boolean cfi, double vlan);
      int get_cos();
```

```
boolean get_cfi();
double get_vlan();
  };
```

Constructor:

EtherTag

```
EtherTag(int cos);
```

Fija el campo CoS del rótulo 802.1. El campo VLAN se pone a un valor por defecto todos ceros. El campo CFI se pone al valor por defecto cero.
Parámetros:
cos – clase de servicio.
Opone la excepción: IllegalArgumentException
si uno o más campos se fijaron inadecuadamente.

EtherTag

```
EtherTag(double vlan);
```

Fija el campo CoS del rótulo 802.1. El campo VLAN se pone a un valor por defecto todos ceros. El campo CFI se pone a un valor por defecto cero.
Parámetros:
vlan – identificador vlan
Opone la excepción: IllegalArgumentException
si uno o más campos se fijan inadecuadamente.

EtherTag

```
EtherTag(double vlan);
```

Fija el campo VLAN del rótulo 802.1. El campo CoS se pone a un valor por defecto todos ceros. El campo CFI se pone a un valor por defecto cero.
Parámetros:
vlan – identificador vlan
Opone la excepción: IllegalArgumentException
si uno o más campos se fijan inadecuadamente.

EtherTag

```
EtherTag(int cos, boolean cfi, double vlan);
```

Fija todos los campos del rótulo 802.1.
Parámetros:
cos – clase de servicio
cfi – identificador de formato canónico
vlan – identificador LAN virtual
Opone la excepción: IllegalArgumentException
si uno o más campos se fijaron inadecuadamente.

Métodos

get_cos

```
int get_cos();
```

Devuelve el valor del campo CoS en el rótulo 802.1.
Devuelve:
la clase de servicio.

get_cfi

```
int get_cfi();
```

Devuelve el valor del campo CFI en el rótulo 802.1.
Devuelve:
el identificador de formato canónico.

get_cfi

```
int get_vlan();
```

Devuelve el valor del campo VLAN en el rótulo 802.1.
Devuelve:
el identificador vlan.

G.5.3 Clase EtherSocket

Esta clase crea zócalos que utilizan las capacidades de la transferencia de datos asegurada de SSCOPMCE. Los datos se transportan en una o más PDU de datos secuenciados (SD) dentro de tramas Ethernet.

```
class EtherSocket {

public:
        EtherSocket(const EtherAddress &host, int port);
        EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherAddress &interface, int localPort);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherTag &tag, const EtherAddress &interface,
                    int localPort);
        EtherAddress getEtherAddress();
        int getPort();
        int getLocalPort();
        EtherAddress getLocalAddress();
        istream &getInputStream();
        ostream &getOutputStream();
        void close();
        };
```

Constructores

EtherSocket

```
EtherSocket(const EtherAddress &host, int port);
```

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 se produce un error de I/O durante la creación del zócalo.

EtherSocket

```
EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión.
Parámetros:
host – dirección del anfitrión de destino
tag – rótulo 802.1
port – puerto de destino
Opone la excepción: IOException
si se produce un error I/O durante la creación del zócalo.

EtherSocket

```
EtherSocket(const EtherAddress &host, int port,
        const EtherAddress &interface, int localPort);
```

Crea un zócalo al puerto especificado en el anfitrión especificado e intenta la conexión. Establece la conexión al anfitrión y puerto especificados en los dos primeros argumentos, y desde la interfaz y puerto de red local especificados en los dos últimos argumentos.
Parámetros:
host – dirección del anfitrión de destino
port – puerto de destino
interface – dirección local
localPort – puerto local
Opone la excepción: IOException
si se produce un error de I/O durante la creación del zócalo.

EtherSocket

```
EtherSocket(const EtherAddress &host, int port,
            const EtherTag &tag, const EtherAddress &interface,
            int localPort);
```

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

Parámetros:

host – dirección del anfitrión de destino

port – puerto de destino

tag – rótulo 802.1

interface – dirección local

localPort – puerto local

Opone la excepción: IOException

si se produce un error de I/O durante la creación del zócalo.

Métodos

getEtherAddress

```
EtherAddress getEtherAddress();
```

Devuelve el anfitrión distante al que está conectado el zócalo o si la conexión ya está cerrada, a qué anfitrión estaba conectado el zócalo.

Devuelve:

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

getPort

```
int getPort();
```

Devuelve el puerto en el que está conectado el zócalo a un anfitrión distante, o estaba conectado o estará conectado al mismo.

Devuelve:

el puerto de conexión en el anfitrión distante.

getLocalPort

```
int getLocalPort();
```

Devuelve el número de puerto del anfitrión local.

Devuelve:

el número de puerto local.

getLocalAddress

```
EtherAddress getLocalAddress();
```

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

Devuelve:

la dirección local.

getInputStream

```
istream &getInputStream();
```

Devuelve un tren de entrada para este zócalo.

Devuelve:

una referencia a un tren de entrada para poder leer bytes de este zócalo.

Opone la excepción: IOException

si se produce un error de I/O durante la creación del tren de salida.

getOutputStream

```
ostream &getOutputStream();
```

Devuelve un tren de salida para este zócalo.

Devuelve:

una referencia a un tren de salida para poder escribir bytes en este zócalo.

Opone la excepción: IOException

si se produce un error de I/O durante la creación del tren de salida.

close

```
void close();
```

Cierra el zócalo.
Opone la excepción: IOException
si se produce un error de I/O durante el cierre del zócalo.

G.5.4 Clase EtherServerSocket

Esta clase implementa zócalos de servidor.

```
class EtherServerSocket{

public:
        EtherServerSocket(int port);
        EtherServerSocket(int port, int queueLength);
        EtherServerSocket(int port, int queueLength, const EtherAddress
                          &bindAddress);
        EtherSocket accept();
        void close();
        EtherAddress getEtherAddress();
        int getLocalPort();
        };
```

Constructores

EtherServerSocket

```
EtherServerSocket(int port);
```

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

EtherServerSocket

```
EtherServerSocket(int port, int queueLength);
```

Crea un zócalo servidor en el puerto especificado con la longitud de cola especificada para las solicitudes de conexión entrantes.
Opone la excepción: BindException
si no se puede crear y vincular el zócalo al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

EtherServerSocket

```
EtherServerSocket(int port, int queueLength, const EtherAddress
                  &bindAddress);
```

Crea un zócalo servidor en el puerto especificado con la longitud de cola especificada para retener las solicitudes de conexión entrantes; el zócalo se vincula únicamente a la dirección Ethernet especificada.
Opone la excepción: BindException
si no se puede crear y vincular el zócalo al puerto solicitado, o si otro zócalo servidor ya está utilizando el puerto solicitado.

Métodos

accept

```
EtherSocket accept();
```

Supervisa que se efectúe una conexión a este zócalo y la acepta. El método se bloquea hasta que se efectúa una conexión
Opone la excepción: IOException
si se produce un error de I/O durante la espera de la conexión.

close

```
void close();
```

Cierra este zócalo.
Opone la excepción: IOException
si se produce un error de I/O durante el cierre del zócalo.

getEtherAddress

```
EtherAddress getEtherAddress();
```

devuelve la dirección local de este zócalo servidor.
Devuelve:
la dirección local.

getLocalPort

```
int getLocalPort();
```

Determina el puerto local que se supervisa.
Devuelve:
el número de puerto local.

G.5.5 Clase Datagrama

Esta clase crea el datagrama referido a una PDU de datos de usuario (UD) no numerados.

```
class Datagram {

public:

        // for receiving datagrams
        Datagram(unsigned char *buffer[ ], int length);
        Datagram(unsigned char *buffer[ ], int offset, int length);

        // for sending datagrams
        Datagram(unsigned char *data[ ], int length, const EtherAddress
                 &destination, int port);
        Datagram(unsigned char *data[ ], int offset, int length,
                 const EtherAddress &destination, int port);

        EtherAddress getAddress();
        int getPort();
        unsigned char **getData();
        int getLength();
        int getOffset();
        setData(unsigned char **data);
        setData(unsigned char **data, int offset, int length);
        void setAddress(const EtherAddress &remote);
        void setPort(int port);
        void setLength(int length);
        };
```

Constructores

Datagram

```
Datagram(unsigned char *buffer[ ], int length);
```

Crea un objeto datagrama para recibir datos. Los datos del datagrama recibido se almacenan en `buffer` hasta que se llena la PDU UD apropiada o hasta que se han escrito `length` bytes en la memoria intermedia.
Parámetros:
buffer – conjunto de bytes
length – número de bytes
Opone la excepción: IllegalArgumentException
si la longitud especificada desborda la memoria intermedia.

Datagram

```
Datagram(unsigned char *buffer[ ], int offset, int length);
```

Crea un objeto datagrama para recibir datos. Los datos del datagrama recibidos se almacena en `buffer`, comenzando en `buffer[offset]` hasta que se llena la PDU UD apropiada o hasta que se han escrito `length` bytes en la memoria intermedia.

Parámetros:
buffer – conjunto de bytes
offset – desplazamiento, en bytes
length –número de bytes

Opone la excepción: IllegalArgumentException
si el número de bytes especificado desborda la memoria intermedia.

Datagram

```
Datagram(unsigned char *data[ ], int length, const EtherAddress
    &destination, int port);
```

Crea un datagrama para enviar datos. El datagrama se llena con `length` bytes de datos. `destination` señala el anfitrión al que se va a entregar el datagrama; `port` es el puerto de destino de ese anfitrión.

Parámetros:
data – conjunto de bytes
length – número de bytes
destination – dirección de destino
port – puerto de destino

Opone la excepción: IllegalArgumentException
si el número de bytes es mayor que el tamaño del conjunto de datos.

Datagram

```
Datagram(unsigned char *data[ ], int offset, int length, const
    EtherAddress &destination, int port);
```

Crea un datagrama para enviar datos. El datagrama se llena con `length` bytes de datos comenzando en `offset` (desplazamiento). `destination` señala el anfitrión al que se va a entregar el datagrama; `port` es el puerto de destino de ese anfitrión.

Parámetros:
data – conjunto de bytes
offset – desplazamiento, en bytes
length – número de bytes
destination – dirección de destino
port – puerto de destino

Opone la excepción: IllegalArgumentException
si el número de bytes es mayor que el tamaño del conjunto de datos.

Métodos:

getAddress

```
EtherAddress getAddress();
```

Devuelve la dirección del anfitrión distante del cual se recibió el datagrama.

Devuelve:
la dirección del anfitrión distante.

getPort

```
int getPort();
```

Devuelve el puerto distante del que se recibió el datagrama.

Devuelve:
el número de puerto distante.

getData

```
unsigned char **getData();
```

Devuelve un conjunto de bytes que incluye los datos del datagrama.

Devuelve:
conjunto de bytes.

getLength

```
int getLength();
```

Devuelve el número de bytes en el datagrama.
Devuelve:
número de bytes.

getOffset

```
int getOffset();
```

Regresa el punto en el conjunto devuelto por `getData()` donde comienzan los datos del datagrama.
Devuelve:
punto en el conjunto donde comienzan los datos.

setData

```
setData(unsigned char **data);
```

Cambia la cabida útil del datagrama.

setData

```
setData(unsigned char **data, int offset, int length);
```

Envía datos en `length` trozos comenzando en `offset` (desplazamiento).

setAddress

```
void setAddress(const EtherAddress &remote);
```

Cambia la dirección de destino de un datagrama.

setPort

```
void setPort(int port);
```

Cambia el puerto al que está direccionado un datagrama.

setLength

```
void setLength(int length);
```

Cambia el número de bytes en la memoria intermedia interna de modo que no se trunquen los datagramas entre las recepciones.

G.5.6 clase DatagramSocket

Esta clase crea un zócalo para enviar o recibir un datagrama.

```
class DatagramSocket {

public:
        DatagramSocket();
        DatagramSocket(int port);
        DatagramSocket(int port, const EtherAddress &address);
        DatagramSocket(int port, const EtherAddress &address,
        const EtherTag &tag);

        void send(const Datagram &d);
        void receive(const Datagram &d);
        void close();
        void getLocalPort();
        void connect(const EtherAddress &host, int port);
        void disconnect();
        int getPort();
        EtherAddress getEtherAddress();
        };
```

Constructores:

DatagramSocket

```
DatagramSocket();
```

Crea un zócalo vinculado a un puerto anónimo. Puede utilizarse el mismo zócalo para recibir datagramas que le devuelve un servidor.
Opone la excepción: SocketException
si no se puede crear el zócalo.

DatagramSocket

```
DatagramSocket(int port);
```

Crea un zócalo que supervisa los datagramas entrantes en un puerto específico, determinado por el argumento `port`.
Parámetros:
port – puerto de supervisión o escucha
Opone la excepción: SocketException
si no se puede crear el zócalo.

DatagramSocket

```
DatagramSocket(int port, const EtherAddress &address);
```

Crea un zócalo que supervisa los datagramas entrantes en un puerto y una interfaz de red específicos. Este constructor es particularmente útil para un anfitrión con multienlaces.
Parámetros:
port – puerto supervisado
dirección – dirección Ethernet del anfitrión
Opone la excepción: SocketException
si no se puede crear el zócalo.

DatagramSocket

```
DatagramSocket(int port, const EtherAddress &address, const EtherTag
                &tagattr);
```

Crea un zócalo que supervisa los datagramas entrantes en un puerto e interfaz de red específicos y sobre una etiqueta especificada por el argumento `tag`. Este constructor es particularmente útil para un anfitrión con multienlaces.
Parámetros:
port – puerto supervisado
address – dirección Ethernet del anfitrión
tagattr –rótulo 802.1
Opone la excepción: SocketException
si no se puede crear el zócalo.

Métodos

send

```
void send(const Datagram &d);
```

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

receive

```
void receive(const Datagram &d);
```

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

close

```
void close();
```

Libera el puerto ocupado por el zócalo.

getLocalPort

```
void getLocalPort();
```

Devuelve el puerto local en el cual está supervisando el zócalo.
Devuelve:
el puerto local.

connect

```
void connect(const EtherAddress &host, int port);
```

Activa la capacidad para enviar datagramas al anfitrión distante especificado o para recibir datagramas del mismo en el puerto distante especificado.

disconnect

```
void disconnect();
```

Desactiva la capacidad del zócalo de modo que pueda enviar datagramas a cualquier anfitrión y puerto, o para que pueda recibir datagramas de los mismos.

getPort

```
int getPort();
```

Devuelve el puerto distante al cual está conectado el zócalo.
Devuelve:
el puerto distante utilizado por la conexión; de lo contrario, devuelve –1 si el zócalo no está conectado.

getEtherAddress

```
EtherAddress getEtherAddress();
```

Devuelve la dirección del anfitrión distante al cual está conectado el zócalo.
Devuelve:
la dirección del anfitrión distante; de lo contrario, devuelve un nulo si no está conectado el zócalo.

G.5.7 clase MulticastSocket

Esta clase crea un zócalo multidifusión. Los datos se transfieren utilizando las capacidades de transferencia de datos sin acuse de recibo de SSCOPMCE.

```
class MulticastSocket : public DatagramSocket {

public:
        MulticastSocket();
        MulticastSocket(int port);
        void joinGroup(const EtherAddress &address);
        void leaveGroup(const EtherAddress &address);
        void send(const Datagram &d);
        void setInterface(const EtherAddress &address);
        EtherAddress getInterface( );
        };
```

Constructores

MulticastSocket

```
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.

MulticastSocket

```
MulticastSocket(int port);
```

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

joinGroup

```
void joinGroup(const EtherAddress &address);
```

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

leaveGroup

```
void leaveGroup(const EtherAddress &address);
```

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

send

```
void send(const Datagram &d);
```

Envía un datagrama por el zócalo multidifusión invocando este método heredado de la clase datagrama.
Parámetros:
address – dirección Ethernet
Opone la excepción: IOException
si el datagrama es más grande de lo que puede soportar el programa informático nativo.

setInterface

```
void setInterface(const EtherAddress &address);
```

Asocia una interfaz de red particular para uso multidifusión en un anfitrión con multienlaces.
Parámetros:
address – dirección Ethernet
Opone la excepción: SocketException
si la dirección no existe en la máquina local.

getInterface

```
EtherAddress getInterface( );
```

Obtiene la dirección de la interfaz en uso.
Devuelve:
la dirección que se está utilizando.

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, aspectos del protocolo Internet y Redes de la próxima generación
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Impreso en Suiza
Ginebra, 2004