UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T Q.2111

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

Amendement 3
(10/2003)

SÉRIE Q: COMMUTATION ET SIGNALISATION

RNIS à large bande – Couche d'adaptation ATM de signalisation (SAAL)

Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

Amendement 3: Interface API pour le protocole SSCOPMCE sur réseau Ethernet avec numéro d'accès UDP

Recommandation UIT-T Q.2111 (1999) – Amendement 3

RECOMMANDATIONS UIT-T DE LA SÉRIE Q

COMMUTATION ET SIGNALISATION

SIGNALISATION DANS LE SERVICE MANUEL INTERNATIONAL	Q.1–Q.3
EXPLOITATION INTERNATIONALE AUTOMATIQUE ET SEMI-AUTOMATIQUE	Q.4–Q.59
FONCTIONS ET FLUX D'INFORMATION DES SERVICES DU RNIS	Q.60–Q.99
CLAUSES APPLICABLES AUX SYSTÈMES NORMALISÉS DE L'UIT-T	Q.100–Q.119
SPÉCIFICATIONS DES SYSTÈMES DE SIGNALISATION N° 4, 5, 6, R1 ET R2	Q.120–Q.499
COMMUTATEURS NUMÉRIQUES	Q.500–Q.599
INTERFONCTIONNEMENT DES SYSTÈMES DE SIGNALISATION	Q.600–Q.699
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 7	Q.700–Q.799
INTERFACE Q3	Q.800–Q.849
SYSTÈME DE SIGNALISATION D'ABONNÉ NUMÉRIQUE N° 1	Q.850–Q.999
RÉSEAUX MOBILES TERRESTRES PUBLICS	Q.1000–Q.1099
INTERFONCTIONNEMENT AVEC LES SYSTÈMES MOBILES À SATELLITES	Q.1100–Q.1199
RÉSEAU INTELLIGENT	Q.1200–Q.1699
PRESCRIPTIONS ET PROTOCOLES DE SIGNALISATION POUR LES IMT-2000	Q.1700–Q.1799
SPÉCIFICATIONS DE LA SIGNALISATION RELATIVE À LA COMMANDE D'APPEL INDÉPENDANTE DU SUPPORT	Q.1900–Q.1999
RNIS À LARGE BANDE	Q.2000–Q.2999
Aspects généraux	Q.2000–Q.2099
Couche d'adaptation ATM de signalisation (SAAL)	**Q.2100–Q.2199**
Protocoles du réseau sémaphore	Q.2200–Q.2299
Aspects communs des protocoles d'application du RNIS-LB pour la signalisation d'accès, la signalisation de réseau et l'interfonctionnement	Q.2600–Q.2699
Protocoles d'application du RNIS-LB pour la signalisation de réseau	Q.2700–Q.2899
Protocoles d'application du RNIS-LB pour la signalisation d'accès	Q.2900–Q.2999

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

Recommandation UIT-T Q.2111

Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

Amendement 3

Interface API pour le protocole SSCOPMCE sur réseau Ethernet avec numéro d'accès UDP

Résumé

Le présent amendement définit une interface de programmation d'application (API, *application programming interface*) pour la machine-protocole définie dans l'Annexe E de la Rec. UIT-T Q.2111 (protocole en mode connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE, *service specific connection oriented protocol in a multi-link and connectionless environment* fonctionnant sur réseau Ethernet).

Par ailleurs, il attribue le numéro d'accès UDP à utiliser avec le protocole SSCOPMCE au-dessus du protocole UDP dans les configurations définies dans les Annexes C et D. Ce numéro de port UDP peut être utilisé conjointement avec un numéro de port n'appartenant pas à la gamme des valeurs dynamiques/privées (de 49152 à 65535).

Source

L'Amendement 3 de la Recommandation Q.2111 (1999) de l'UIT-T a été approuvé le 14 octobre 2003 par la Commission d'études 11 (2001-2004) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT avait été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Recommandation UIT-T Q.2111

Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

Amendement 3

Interface API pour le protocole SSCOPMCE sur réseau Ethernet avec numéro d'accès UDP

1) Paragraphe 2.2 – Bibliographie

Ajouter la référence suivante:

[24] ISO/CEI 14882:2003, – *Langages de programmation – C++*.

2) Paragraphe 5.3 – Modes de fonctionnement

Modifier la fin de la dernière phrase de l'alinéa qui suit immédiatement la Figure 2 et ajouter une nouvelle phrase comme indiqué ci-après:

De plus, l'Annexe F définit une interface de programmation d'application (API, *application programming interface*) pour le protocole SSCOPMCE sur Ethernet, spécifiée en Ada. Par ailleurs, l'Annexe G définit une interface de programmation d'application (API) pour le protocole SSCOPMCE sur Ethernet, spécifiée en C++.

3) Paragraphe C.3.2.1 – Description de l'interface UDP supérieure

i) Modifier la fin de l'alinéa "Port source" du paragraphe C.3.2.1 et ajouter la nouvelle phrase suivante:

S'il est présent, la valeur numérique pour le protocole SSCOPMCE au-dessus du protocole UDP est soit "VALEUR A ATTRIBUER PAR IANA" ou valeur autre que celles de la gamme de valeurs dynamiques/privées (de 49152 à 65535), selon l'environnement dans lequel le protocole SSCOPMCE est utilisé.

ii) Modifier la fin de l'alinéa "Port de destination" du paragraphe C.3.2.1 et ajouter la nouvelle phrase suivante:

La valeur numérique pour le protocole SSCOPMCE au-dessus du protocole UDP est soit "VALEUR A ATTRIBUER PAR IANA" ou valeur autre que celles de la gamme de valeurs dynamiques/privées (de 49152 à 65535), selon l'environnement dans lequel le protocole SSCOPMCE est utilisé.

4) Paragraphe D.3.2.1 – Description de l'interface UDP supérieure

i) Modifier la fin de l'alinéa "Accès d'origine" du paragraphe D.3.2.1 et ajouter la nouvelle phrase suivante:

S'il est présent, la valeur numérique pour le protocole SSCOPMCE au-dessus du protocole UDP est soit "VALEUR A ATTRIBUER PAR IANA" ou valeur autre que celles de la gamme de valeurs dynamiques/privées (de 49152 à 65535), selon l'environnement dans lequel le protocole SSCOPMCE est utilisé.

ii) Modifier la fin de l'alinéa "Accès de destination" du paragraphe D.3.2.1 et ajouter la nouvelle phrase suivante:

La valeur numérique pour le protocole SSCOPMCE au-dessus du protocole UDP est soit "VALEUR A ATTRIBUER PAR IANA" ou valeur autre que celles de la gamme de valeurs dynamiques/privées (de 49152 à 65535), selon l'environnement dans lequel le protocole SSCOPMCE est utilisé.

5) Annexe G

Ajouter la nouvelle Annexe G (interface API pour le protocole SSCOPMCE sur réseau Ethernet) suivante:

Annexe G

Langage de programmation C++ – Interface API pour le protocole SSCOPMCE sur réseau Ethernet

G.1 Introduction

L'Annexe E de la présente Recommandation spécifie la mise en place du protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE) au sommet du service sans connexion assuré par les réseaux Ethernet IEEE 802.3. Il s'agit essentiellement pour la configuration de réaliser un bus de données pour systèmes ouverts destiné aux systèmes à boucle fermée.

Les applications peuvent utiliser les services SSCOPMCE suivants via les points SAP offerts par la fonction SSCF au niveau de l'interface UNI [12]:

- transfert de données sans accusé de réception;
- transfert de données garanti;
- transparence des informations transférées;
- établissement et libération des connexions pour le transfert de données garanti.

La Rec. UIT-T Q.2111 et son Annexe E contiennent les spécifications nécessaires à l'élaboration d'un produit fondé sur une carte d'interface réseau Ethernet, en revanche la présente annexe spécifie une interface de programmation d'application (API) avec le point SAP. La spécification d'une interface API a pour objet d'amener les fournisseurs de systèmes d'exploitation à proposer une interface normalisée, ouverte et familière aux concepteurs de logiciels afin de tirer parti des capacités de réseau offertes par un bus de données Ethernet.

G.2 Objectifs de l'interface API avec bus de données Ethernet

Cette interface API avec bus de données Ethernet est relativement compacte et autonome, elle permet au programmeur d'accéder aux services SSCOPMCE lorsque ces services opèrent sur une couche Liaison de données Ethernet. La conception d'une interface API devait répondre aux deux objectifs suivants:

1) elle devait être fondée sur la notion de connecteur qui est très utilisée dans la plupart des interfaces API de réseau pour systèmes d'exploitation en temps réel des ordinateurs de bureau. Dans les connecteurs chaque connexion de réseau est essentiellement considérée comme un flux dans lequel on peut écrire ou lire des octets, ce qui permet de les considérer comme une extension des concepts familiers d'E/S de fichiers;
2) elle devait permettre le traitement des exceptions afin de remédier aux erreurs de temps de traitement.

G.3 Implémentation de l'interface API avec bus de données Ethernet: aperçu général

L'interface API avec bus de données Ethernet est programmée en langage C++. Ce langage a été retenu en raison de son utilisation de plus en plus répandue dans les systèmes aérospatiaux et les systèmes de défense, applications qui ont, entre autres, influencé l'élaboration de l'Annexe E. Par conséquent, une interface API en C++ permettra le passage des architectures de systèmes existantes à une architecture à bus de données Ethernet. De plus, les nouvelles architectures de systèmes pourront être fondées sur ce bus de données. Ces interfaces API seront dotées d'une interface de programmation normalisée destinée à être utilisée avec un bus de données Ethernet.

L'interface API en C++ définit les types (objets) suivants:

- **EtherAddress**: représente une adresseEthernet.
- **EtherSocket**: implémente un connecteur côté client qui utilise les capacités de transfert garanti du protocole SSCOPMCE. Les données sont transportées dans une ou plusieurs unités PDU de données en séquence (SD, *sequenced-data*) dans les trames Ethernet.
- **EtherTag**: contient les attributs associés au type d'étiquette 802.1 [22].
- **EtherServerSocket**: implémente un connecteur côté serveur qui utilise les capacités de transfert garanti du protocole SSCOPMCE. Les données sont transportées dans une ou plusieurs unités PDU de données en séquence (SD) dans les trames Ethernet.
- **Datagram**: crée un datagramme renvoyant à une unité PDU de données d'utilisateur (UD, *user data*) numérotée.
- **DatagramSocket**: crée un connecteur pour envoyer et recevoir un datagramme.
- **MulticastSocket**: crée un connecteur de multidiffusion pour envoyer et recevoir un datagramme. Les données sont transportées dans une ou plusieurs PDU de données d'utilisateur (UD, *user data*) numérotées. L'opération de multidiffusion est fondée sur le protocole d'enregistrement de multidiffusion (GMRP, *GARP multicast registration protocol*).

Le faible nombre de types qui sont définis tient au nouveau mappage autorisé dans l'Annexe E. Du point de vue des définitions, ces types, ainsi que les opérations associées sur ces types, sont contenus dans le progiciel Ethernet Databus. Un pilote associé à une carte d'interface de réseau doit lui être conforme. Du point de vue de l'implémentation, ces types sont désignés comme privés et, tout comme la spécification des opérations associées, sortent du cadre de la présente Recommandation. On dispose ainsi d'une certaine souplesse d'implémentation et d'évolution de l'interface API.

G.4 Récapitulatif des définitions contenues dans la bibliothèque en C++

Ce qui suit est un récapitulatif des définitions contenues dans la bibliothèque Ethernet Databus:

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 Description des définitions continues dans la bibliothèque en C++

Ce qui suit est une description détaillée de chaque classe et des fonctions membre qui lui sont associées.

G.5.1 Classe EtherAddress

Cette classe représente une adresse Ethernet.

```
class EtherAddress {

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

Variables

addr

```
char *addr;
```

Les six octets d'une adresse Ethernet, dont le premier est l'octet d'ordre le plus élevé. L'adresse se présente sous la forme d'une chaîne décrivant les octets en notation hexadécimale, par exemple: "347B0046A8CE".

Méthodes

getOUI

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

Renvoie les trois premiers octets d'une adresse Ethernet, à savoir l'identificateur OUI (*organizationally universal identifier*).
Renvoie:
la partie OUI de l'adresse, sous forme d'une chaîne décrivant les octets en notation hexadécimale. Exemple: "347B00".

getLocal

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

Renvoie les trois derniers octets d'une adresse Ethernet, à savoir la partie assignée localement.
Renvoie:
partie de l'adresse assignée localement sous forme d'une chaîne décrivant les octets en notation hexadécimale. Exemple: "46A8CE".

isGroupAddress

```
boolean isGroupAddress();
```

Détermine si l'adresse Ethernet est une adresse de groupe, c'est-à-dire si le premier bit de l'octet d'ordre le plus élevé est zéro.
Renvoie:
"True" si l'adresse est une adresse de groupe, "false" dans le cas contraire.

getHostAddress

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

Renvoie l'adresse Ethernet numérique associée à un hôte.
Renvoie:
une seule adresse Ethernet, sous forme d'une chaîne décrivant les octets en notation hexadécimale, par exemple: "3487A8CE".
Envoie: UnknownHostException
lorsque aucune adresse Ethernet pour l'hôte n'a pu être trouvée.

getAllHostAddresses

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

Renvoie un tableau des adresses associées à un hôte fortement connecté.
Renvoie:
Un tableau d'adresses Ethernet, avec un pointeur désignant la première adresse.
Envoie: UnknownHostException
Lorsque aucune adresse Ethernet pour l'hôte n'a pu être trouvée.

G.5.2 Classe EtherTag

Cette classe contient les attributs associés au type d'étiquette 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();
        };
```

Constructeur

EtherTag

`EtherTag(int cos);`

Positionne le champ CoS de l'étiquette 802.1. Le champ VLAN prend la valeur par défaut qui est constituée de zéros uniquement. Le champ CFI prend la valeur par défaut qui est zéro.
Paramètres:
cos – classe de service
Envoie: IllegalArgumentException
si un ou plusieurs champs prennent une valeur inappropriée.

EtherTag

`EtherTag(double vlan);`

Positionne le champ CoS de l'étiquette 802.1. Le champ VLAN prend la valeur par défaut qui est constituée de zéros uniquement. Le champ CFI prend la valeur par défaut qui est zéro.
Paramètres:
vlan – identificateur de vlan
Envoie: IllegalArgumentException
si un ou plusieurs champs prennent une valeur inappropriée.

EtherTag

`EtherTag(double vlan);`

Positionne le champ VLAN de l'étiquette 802.1. Le champ CoS prend la valeur par défaut qui est constituée de zéros uniquement. Le champ CFI prend la valeur par défaut qui est zéro.
Paramètres:
vlan – identificateur de vlan
Envoie: IllegalArgumentException
si un ou plusieurs champs prennent une valeur inappropriée.

EtherTag

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

Positionne tous les champs d'étiquette 802.1.
Paramètres:
cos – classe de service
cfi – identificateur de format canonique
vlan – identificateur de LAN virtuel
Envoie: IllegalArgumentException
si un ou plusieurs champs prennent une valeur inappropriée.

Méthodes

get_cos

`int get_cos();`

Renvoie la valeur du champ CoS dans l'étiquette 802.1.
Renvoie:
la classe de service.

get_cfi

```
int get_cfi();
```

Renvoie la valeur du champ CFI dans l'étiquette 802.1.
Renvoie:
l'identificateur de format canonique.

get_cfi

```
int get_vlan();
```

Renvoie la valeur du champ VLAN dans l'étiquette 802.1.
Renvoie:
l'identificateur vlan.

G.5.3 Classe EtherSocket

Cette classe crée des connecteurs qui utilisent les capacités de transfert de données garanti du protocole SSCOPMCE. Les données sont transportées dans une ou plusieurs unités PDU de données en séquence (SD) dans des trames 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();
        };
```

Constructeurs

EtherSocket

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

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion.
Paramètres:
host – adresse de l'hôte de destination
port – port de destination
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur.

EtherSocket

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

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion.
Paramètres:
host – adresse de l'hôte de destination
tag – étiquette 802.1
port – port de destination
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur.

EtherSocket

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

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion. Il se connecte à l'hôte et au port spécifiés dans les deux premiers arguments, et depuis l'interface de réseau et le port locaux spécifiés dans les deux derniers arguments.

Paramètres:
host – adresse de l'hôte de destination
port – port de destination
interface – adresse locale
localPort – port local
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur.

EtherSocket

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

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion. Il se connecte à l'hôte et au port spécifiés dans les deux premiers arguments, et depuis l'interface de réseau et le port locaux spécifiés dans les deux derniers arguments.

Paramètres:
host – adresse de l'hôte de destination
port – port de destination
tag – étiquette 802.1
interface – adresse locale
localPort – port local
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur.

Méthodes

getEtherAddress

```
EtherAddress getEtherAddress();
```

Renvoie l'adresse de l'hôte auquel le connecteur est relié ou, si la connexion est maintenant fermée, l'adresse de l'hôte auquel le connecteur était relié lorsqu'il était relié.

Renvoie:
l'adresse distante Ethernet à laquelle le connecteur est relié.

getPort

```
int getPort();
```

Renvoie le port auquel le connecteur est, était ou sera relié sur l'hôte distant.

Renvoie:
le port auquel le connecteur est relié sur l'hôte distant.

getLocalPort

```
int getLocalPort();
```

Renvoie le numéro de port pour l'hôte local.

Renvoie:
le numéro de port local.

getLocalAddress

```
EtherAddress getLocalAddress();
```

Prend l'adresse locale auquel le connecteur est lié.

Renvoie:
l'adresse locale.

getInputStream

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

Renvoie un flux d'entrée pour ce connecteur.
Renvoie:
une référence à un flux d'entrée pour lire les octets à partir de ce connecteur.
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du flux de sortie.

getOutputStream

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

Renvoie un flux de sortie pour ce connecteur.
Renvoie:
une référence à un flux de sortie pour écrire les octets à partir de ce connecteur.
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du flux de sortie.

close

```
void close();
```

Ferme le connecteur.
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la fermeture du connecteur.

G.5.4 Classe EtherServerSocket

Cette classe implémente des connecteurs de serveur.

```
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();
        };
```

Constructeurs

EtherServerSocket

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

Crée un connecteur de serveur sur le port spécifié par l'argument.
Envoie: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé.

EtherServerSocket

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

Crée un connecteur de serveur sur le port spécifié avec la longueur de file d'attente spécifiée pour les demandes de connexion entrantes.
Envoie: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé.

EtherServerSocket

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

Crée un connecteur de serveur sur le port spécifié avec la longueur de file d'attente spécifiée pour retenir les demandes de connexion; le connecteur assure uniquement le lien avec l'adresse Ethernet spécifiée.
Envoie: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé.

Méthodes

accept

```
EtherSocket accept();
```

Se met en attente d'une connexion à réaliser vers ce connecteur et l'accepte. Cette méthode provoque un blocage jusqu'à ce que la connexion soit réalisée.
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant l'attente d'une connexion.

close

```
void close();
```

Ferme ce connecteur.
Envoie: IOException
lorsqu'une erreur d'E/S s'est produite pendant la fermeture du connecteur.

getEtherAddress

```
EtherAddress getEtherAddress();
```

Renvoie l'adresse locale de ce connecteur de serveur.
Renvoie:
l'adresse locale.

getLocalPort

```
int getLocalPort();
```

Détermine le port local sur lequel l'écoute a lieu.
Renvoie:
le numéro de port local.

G.5.5 Classe Datagram

Cette classe crée un datagramme se rapportant à une unité PDU de données non numérotées (UD, *unnumbered data*).

```
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);
};
```

Constructeurs:

Datagram

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

Crée un objet datagramme pour la réception des données. Les données du datagramme reçues sont stockées dans `buffer` jusqu'à ce que l'unité PDU UD appropriée ait été remplie ou jusqu'à ce que les octets `length` aient été écrits dans le tampon.

Paramètres:
buffer – tableau d'octets
length – nombre d'octets
Envoie: IllegalArgumentException
lorsque la longueur spécifiée provoque un débordement du tampon.

Datagram

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

Crée un objet datagramme pour la réception des données. Les données du datagramme reçues sont stockées dans `buffer`, en commençant à `buffer[offset]`, jusqu'à ce que l'unité PDU UD appropriée ait été remplie ou jusqu'à ce que les octets `length` aient été écrits dans le tampon.

Paramètres:
buffer – tableau d'octets
offset – décalage, en octets
length – nombre d'octets
Envoie: IllegalArgumentException
lorsque la longueur spécifiée provoque un débordement du tampon.

Datagram

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

Crée un datagramme pour l'envoi des données. Le datagramme contient des octets `length` des données. `destination` pointe vers le serveur auquel doit être remis le datagramme, le `port` est le `port` de destination sur ce serveur.

Paramètres:
data – tableau d'octets
length – nombre d'octets
destination – adresse de destination
port – port de destination
Envoie: IllegalArgumentException
lorsque la longueur est supérieure à la taille du tableau de données.

Datagram

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

Crée un datagramme pour l'envoi des données. Le datagramme contient des octets de `length` des données commençant à `offset`. La `destination` pointe vers le serveur auquel doit être remis le datagramme, le `port` est le port de destination sur ce serveur.

Paramètres:
data – tableau d'octets
offset – décalage en octets
length – nombre d'octets
destination – adresse de destination
port – port de destination

Envoie: IllegalArgumentException
lorsque la longueur est supérieure à la taille du tableau de données.

Méthodes

getAddress

```
EtherAddress getAddress();
```

Renvoie l'adresse du serveur distant d'où provient le datagramme reçu.

Renvoie:
l'adresse du serveur distant.

getPort

```
int getPort();
```

Renvoie le port distant d'où provient le datagramme reçu.

Renvoie:
le numéro du port distant.

getData

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

Renvoie un tableau d'octets contenant les données provenant du datagramme.

Renvoie:
le tableau d'octets.

getLength

```
int getLength();
```

Renvoie le nombre d'octets du datagramme.

Renvoie:
le nombre d'octets.

getOffset

```
int getOffset();
```

Renvoie le point du tableau renvoyé par `getData()` où les données extraites du datagramme commencent.

Renvoie:
point dans le tableau où les données commencent.

setData

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

Modifie la charge utile du datagramme.

setData

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

Envoie des données par tranches de `length` commençant à `offset`.

setAddress

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

Modifie l'adresse de destination d'un datagramme.

setPort

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

Modifie le port vers lequel le datagramme est envoyé.

setLength

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

Modifie le nombre d'octets dans le tampon interne de manière à ce que les datagrammes ne soient pas tronqués entre les réceptions.

G.5.6 Classe DatagramSocket

Cette classe crée un connecteur permettant d'envoyer ou de recevoir un datagramme.

```
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();
        };
```

Constructeurs

DatagramSocket

```
DatagramSocket();
```

Crée un connecteur lié à un port anonyme. Le même connecteur peut être utilisé pour recevoir des datagrammes renvoyés par un serveur.
Envoie: SocketException
lorsque le connecteur ne peut être créé.

DatagramSocket

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

Crée un connecteur qui écoute des datagrammes entrants sur un port spécifique, spécifié par l'argument `port`.
Paramètres:
port – port d'écoute
Envoie: SocketException
lorsque le connecteur ne peut être créé.

DatagramSocket

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

Crée un connecteur qui écoute des datagrammes entrants sur un port et une interface de réseau spécifiques. Cette structure est particulièrement utile pour un hôte fortement connecté.
Paramètres:
port – port d'écoute
tag – étiquette 802.1
Envoie: SocketException
lorsque le connecteur ne peut être créé.

DatagramSocket

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

Crée un connecteur qui écoute des datagrammes entrants sur un port et une interface de réseau spécifiques et sur une étiquette spécifiée par l'argument `tag`. Cette structure est particulièrement utile pour un hôte fortement connecté.

Paramètres:
port – port d'écoute
address – adresse Ethernet de l'hôte
`tagattr` – étiquette 802.1

Envoie: SocketException
lorsque le connecteur ne peut être créé.

Méthodes

send

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

Envoie un seul datagramme dp sur le réseau en utilisant le connecteur de datagramme.

Paramètres:
d – objet datagramme

Envoie: IOException
lorsque la taille du datagramme à envoyer est supérieure à celle que le logiciel spécifique peut prendre en charge.

receive

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

Reçoit un seul datagramme du réseau et le stocke dans l'objet datagramme dp.

Paramètres:
d – objet datagramme

Envoie: IOException
lorsqu'il y a problème de réception des données.

close

```
void close();
```

Libère le port occupé par le connecteur.

getLocalPort

```
void getLocalPort();
```

Renvoie le port local sur lequel le connecteur est en attente (écoute).

Renvoie:
le port local.

connect

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

Active la capacité d'envoi et de réception de datagrammes vers ou en provenance de l'hôte distant spécifié sur le port distant spécifié.

disconnect

```
void disconnect();
```

Désactive la capacité du connecteur de sorte qu'il peut envoyer et recevoir des datagrammes vers ou en provenance d'un hôte ou d'un port quelconque.

getPort

```
int getPort();
```

Renvoie le port distant auquel le connecteur est relié.

Renvoie:
le port distant utilisé par la connexion ou, si le connecteur n'est pas raccordé, la valeur –1.

getEtherAddress

```
EtherAddress getEtherAddress();
```

Renvoie l'adresse de l'hôte distant auquel le connecteur est relié.

Renvoie:
l'adresse de l'hôte distant ou, si le connecteur n'est pas raccordé, une information nulle.

G.5.7 Classe MulticastSocket

Cette classe crée un connecteur de multidiffusion. Les données sont transférées au moyen des capacités de transfert de données sans accusé de réception du protocole 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( );
        };
```

Constructeurs

MulticastSocket

`MulticastSocket( );`

Crée un connecteur de multidiffusion lié à un port anonyme. Le destinataire répond au même port.
Envoie: SocketException
en cas d'impossibilité de création du connecteur.

MulticastSocket

`MulticastSocket(int port);`

Crée un connecteur de multidiffusion sur un support spécifique en utilisant une étiquette spécifiée.
Paramètres:
port – port source
Envoie: SocketException
lorsque le connecteur ne peut être créé, par exemple, lorsque le port est déjà utilisé.

joinGroup

`void joinGroup(const EtherAddress &address);`

Après la création d'un connecteur de multidiffusion, cette méthode lui permet de se joindre à un groupe de multidiffusion.
Paramètres:
address – adresse Ethernet
Envoie: IOException
lorsque l'adresse n'est pas une adresse de groupe.

leaveGroup

`void leaveGroup(const EtherAddress &address);`

Après qu'un connecteur de multidiffusion se soit joint à un groupe, il peut le quitter en appelant cette méthode.
Paramètres:
address – adresse Ethernet
Envoie: IOException
lorsque l'adresse n'est pas une adresse de groupe.

send

`void send(const Datagram &d);`

Envoie un datagramme sur le connecteur de multidiffusion en appelant cette méthode héritée de la classe de datagramme.
Paramètres:
address – adresse Ethernet
Envoie: IOException
lorsque la taille du datagramme est supérieure à celle que le logiciel spécifique peut prendre en charge.

setInterface

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

Associe une interface de réseau particulière destinée à être utilisée pour la multidiffusion sur un hôte fortement connecté.

Paramètres:

address – adresse Ethernet

Envoie: SocketException

lorsque l'adresse n'existe pas dans la machine locale.

getInterface

```
EtherAddress getInterface( );
```

Obtient l'adresse de l'interface en cours d'utilisation.

Renvoie:

l'adresse en cours d'utilisation.

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	**Commutation et signalisation**
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	Réseaux de données et communication entre systèmes ouverts
Série Y	Infrastructure mondiale de l'information, protocole Internet et réseaux de nouvelle génération
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Imprimé en Suisse
Genève, 2004