ITU-T Recommendation Q.2111 – Amendment 2

SERIES Q: SWITCHING AND SIGNALLING
Broadband ISDN – Signalling ATM adaptation layer (SAAL)

B-ISDN ATM adaptation layer – Service specific connection oriented protocol in a multilink and connectionless environment (SSCOPMCE)

Amendment 2: API for SSCOPMCE over Ethernet
ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING

| SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE | Q.1–Q.3 |
| INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING | Q.4–Q.59 |
| FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN | Q.60–Q.99 |
| CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS | Q.100–Q.119 |
| SPECIFICATIONS OF SIGNALLING SYSTEM No. 4 | Q.120–Q.139 |
| SPECIFICATIONS OF SIGNALLING SYSTEM No. 5 | Q.140–Q.199 |
| SPECIFICATIONS OF SIGNALLING SYSTEM No. 6 | Q.250–Q.309 |
| SPECIFICATIONS OF SIGNALLING SYSTEM R1 | Q.310–Q.399 |
| SPECIFICATIONS OF SIGNALLING SYSTEM R2 | Q.400–Q.499 |
| DIGITAL EXCHANGES | Q.500–Q.599 |
| INTERWORKING OF SIGNALLING SYSTEMS | Q.600–Q.699 |
| SPECIFICATIONS OF SIGNALLING SYSTEM No. 7 | Q.700–Q.799 |
| Q3 INTERFACE | Q.800–Q.849 |
| DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1 | Q.850–Q.999 |
| PUBLIC LAND MOBILE NETWORK | Q.1000–Q.1099 |
| INTERWORKING WITH SATELLITE MOBILE SYSTEMS | Q.1100–Q.1199 |
| INTELLIGENT NETWORK | Q.1200–Q.1699 |
| SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2000 | Q.1700–Q.1799 |
| SPECIFICATIONS OF SIGNALLING RELATED TO BEARER INDEPENDENT CALL CONTROL (BICC) | Q.1900–Q.1999 |
| BROADBAND ISDN | \[Q.2000–Q.2999\] |
| General aspects | Q.2000–Q.2099 |

**Signalling ATM adaptation layer (SAAL)**

General aspects Q.2100–Q.2199

For further details, please refer to the list of ITU-T Recommendations.
Summary
This amendment to ITU-T Rec. Q.2111 provides an Application Programming Interface for SSCOPMCE over Ethernet. It is being provided to ease the incorporation of SSCOPMCE into communication systems utilizing Ethernet.

Source
Amendment 2 to ITU-T Recommendation Q.2111 was prepared by ITU-T Study Group 11 (2001-2004) and approved under the WTSA Resolution 1 procedure on 13 April 2002.
FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2002

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Clause 2.2 – Bibliography</td>
<td>1</td>
</tr>
<tr>
<td>2) Clause 4 – Abreviations</td>
<td>1</td>
</tr>
<tr>
<td>3) Clause 5.3 – Modes of Operation</td>
<td>1</td>
</tr>
<tr>
<td>4) Annex F</td>
<td>1</td>
</tr>
</tbody>
</table>

Annex F – API for SSCOPMCE over Ethernet 1                               | 1    |

1 Introduction                                                          | 1    |
2 Objectives of the Ethernet Databus API                                 | 2    |
3 Overview of the Ethernet Databus API Implementation                     | 2    |
4 Summary of Ada Package Definition                                      | 3    |
5 Description of Ada Package Definition                                  | 6    |
  5.1 EtherAddress Subroutines                                           | 6    |
  5.2 EtherTag Subroutines                                               | 7    |
  5.3 EtherSocket Subroutines                                            | 8    |
  5.4 EtherServerSocket Subroutines                                      | 10   |
  5.5 Datagram Subroutines                                               | 12   |
  5.6 DatagramSocket Subroutines                                         | 14   |
  5.7 MulticastSocket Subroutines                                        | 16   |
ITU-T Recommendation Q.2111

B-ISDN ATM adaptation layer – Service specific connection oriented protocol in a multilink and connectionless environment (SSCOPMCE)

Amendment 2
API for SSCOPMCE over Ethernet

1) Clause 2.2 – Bibliography
Add the following reference:

2) Clause 4 – Abreviations
Add the following definition alphabetically:
API Application Programming Interface

3) Clause 5.3 – Modes of Operation
Add the following sentence at the end of the paragraph immediately following Figure 2:
In addition, Annex F provides an Application Programming Interface (API) for SSCOPMCE over Ethernet.

4) Annex F
Add new Annex F (API for SSCOPMCE over Ethernet) as follow:

Annex F
API for SSCOPMCE over Ethernet

1 Introduction
Annex E to this Recommendation specifies the deployment of SSCOPMCE on top of the connectionless service provided by IEEE 802.3 Ethernet networks. The primary driver for the configuration is to realize an open-systems databus for closed-loop systems.

Applications can utilize the following services of SSCOPMCE through the SAP offered by the SSCF at the UNI [12]:
• Unacknowledged transfer of data;
• Assured transfer of data;
• Transparency of transferred information;
• Establishment and release of connections for assured transfer of data.

Whereas the main body of the Recommendation and Annex E contain the specifications necessary to develop a product based on an Ethernet network interface card, this annex specifies an application programming interface (API) to the SAP. The reason for specifying an API is to drive development tool and/or real-time operating system vendors to offer a standard, open and familiar
interface for software developers to take advantage of the network capabilities offered by an Ethernet-based databus.

2 Objectives of the Ethernet Databus API

The Ethernet Databus API is relatively small and self-contained, allowing a programmer to access SSCOPMCE services when such services operate over an Ethernet datalink layer. Two objectives were used in designing an API:

- The API should be based on the notion of sockets, which has been widely used in the majority of existing network APIs for desktop and real-time operating systems. Sockets essentially treat each network connection as a stream into which bytes can be written-to or read-from, allowing them to be an extension of familiar file I/O concepts.
- The API should include provisions for exception-handling in order to manage run-time errors.

3 Overview of the Ethernet Databus API Implementation

The Ethernet Databus API is written in the Ada 95 programming language [3]. The choice of Ada is based on its widespread use in aerospace and defense systems, one of the application areas driving the specification of Annex E/Q.2111. Consequently, an Ada-based API will permit the migration of existing system architectures toward an Ethernet-based databus. In addition, new system architectures may be based on it. Such an API will also offer a standard programming interface for use with an Ethernet-based databus.

The Ada-based API defines the following types (objects):

- **EtherAddress**: Represents an Ethernet address.
- **EtherSocket**: Implements a client-side socket that utilizes the assured data transfer capabilities of SSCOPMCE. Data is transported in one or more sequenced-data (SD) PDUs within Ethernet frames.
- **EtherTag**: Contains the attributes associated with the 802.1 tag type [22].
- **EtherServerSocket**: Implements a server-side socket that utilizes the assured data transfer capabilities of SSCOPMCE. Data is transported in one or more sequenced-data (SD) PDUs within Ethernet frames.
- **Datagram**: Creates a datagram referring to an un-numbered user data (UD) PDU.
- **DatagramSocket**: Creates a socket to send or receive a datagram.
- **MulticastSocket**: Creates a multicast socket to send or receive a datagram. Data is transported in one or more un-numbered user data (UD) PDUs. Multicast operation is based on the GARP Multicast Registration Protocol (GMRP) [21].

The fact that only a few types are defined is based in large part to the streamlined mapping of protocol layers allowed in Annex E/Q.2111. From a definition viewpoint, these types, and the associated operations on these types, are contained in the package Ethernet Databus. A driver associated with a network interface card must be compliant with it. From an implementation viewpoint, these types are designated as private, and, like the specification of associated operations, are outside the scope of this Recommendation. This has been done to allow flexibility in the implementation and evolution of the API.
4 Summary of Ada Package Definition

The following is a summary of the Ethernet Databus package:

```ada
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 BYTE is mod 2**8;
  type BYTE_ARRAY is array (POSITIVE 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;
                           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;

end Ethernet Databus;
```

ITU-T Rec. Q.2111/Amd.2 (04/2002) 3
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;
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;
IllegalArgumentException: exception;
BindException: exception;
IOException: exception;
SocketException: exception;

private
-- Implementation dependent
end Ethernet Databus;

5 Description of Ada Package Definition
The following is a detailed description of each of the subroutines:

5.1 EtherAddress Subroutines
getAddress
function getAddress(addr: EtherAddress) return STRING;
    Returns a fully-qualified 48-bit Ethernet address
Parameters:
    addr – Ethernet address
Returns:
    A single Ethernet address, as a string describing the bytes in hex notation, e.g., "3407A4CE0000".
getOUI
function getOUI(addr: EtherAddress) return STRING;
    Returns the first three bytes of an Ethernet address: the Organizationally Universal Identifier.
Parameters:
    addr – Ethernet address
Returns:
    The OUI part of the address, as a string describing the bytes in hex notation, e.g., "3407A4".
getLocal
function getLocal(addr: EtherAddress) return STRING;
    Returns the last three bytes of an Ethernet address: the locally assigned part.
Parameters:
    addr – Ethernet address
Returns:
    The locally assigned part of the address, as a string describing the bytes in hex notation, e.g., "CE0000".
isGroupAddress
function isGroupAddress(addr: EtherAddress) return BOOLEAN;
    Determines whether the Ethernet address is a group address, if the first bit of the highest order byte is zero
Parameters:
    addr – Ethernet address
Returns:
    True if the address is a group address, false otherwise
getLocalAddress

    function getLocalAddress(addr: EtherAddress) return STRING;
    Returns the address associated with the local host.
    Parameters:
    addr – local Ethernet address
    Returns:
    An Ethernet address
    Throws: UnknownHostException
    if no Ethernet address for the host could be found

getAllHostAddresses

    function getAllHostAddresses return EtherAddresses;
    Returns an array of the addresses associated with a multi-homed host
    Returns:
    An array of Ethernet addresses
    Throws: UnknownHostException
    If not Ethernet address for the host could be found

5.2 EtherTag Subroutines

makeEtherTag

    function makeEtherTag(cos: in COS_TYPE) return EtherTag;
    Sets the CoS field of the 802.1 tag. The VLAN field is set to a default value of all zeroes. The CFI field
    is set to a default value of zero.
    Parameters:
    cos – class of service

makeEtherTag

    function makeEtherTag(vlan: in VLAN_TYPE) return EtherTag;
    Sets the VLAN field of the 802.1 tag. The CoS field is set to a default value of all zeroes. The CFI field
    is set to a default value of zero.
    Parameters:
    vlan – vlan identifier

makeEtherTag

    function makeEtherTag(cos: in COS_TYPE;
            cfi: in BOOLEAN;
            vlan: in VLAN_TYPE)
            return EtherTag;
    Sets all the fields of the 802.1 tag.
    Parameters:
    cos – class of service
    cfi – canonical format identifier
    vlan – virtual LAN identifier

get_cos

    function get_cos(tag: EtherTag) return COS_TYPE;
    Returns the value of the CoS field in the 802.1 tag.
    Parameters:
    tag – 802.1 tag
    Returns:
    the class of service

get_cfi

    function get_cfi(tag: EtherTag) return BOOLEAN;
    Returns the value of the CFI field in the 802.1 tag.
    Parameters:
    tag – 802.1 tag
    Returns:
    the canonical format identifier

get_vlan

    function get_vlan(tag: EtherTag) return VLAN_TYPE;
    Returns the value of the VLAN field in the 802.1 tag.
    Parameters:
    tag – 802.1 tag
Returns:
the vlan identifier

5.3 EtherSocket Subroutines

makeEtherSocket

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

Creates a socket to the specified port on the specified host and tries to connect.
Parameters:
host – destination host address
port – destination port

Throws: IOException
if an I/O error occurs while creating the socket

makeEtherSocket

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

Creates a socket to the specified port on the specified host and tries to connect.
Parameters:
host – destination host address
tag – 802.1 tag
port – destination port

Throws: IOException
if an I/O error occurs while creating the socket

makeEtherSocket

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

Creates a socket to the specified port on the specified host and tries to connect. It connects to the host and
port specified in the first two arguments, and from the local network interface and port specified in the
last two arguments.
Parameters:
host – destination host address
port – destination port
interface – local address
localPort – local port

Throws: IOException
if an I/O error occurs while creating the socket

makeEtherSocket

function makeEtherSocket(host: EtherAddress;
port: PORT_TYPE,
tag: EtherTag;
interface: EtherAddress;
localPort: PORT_TYPE)
return EtherSocket;

Creates a socket to the specified port on the specified host and tries to connect. It connects to the host and
port specified in the first two arguments, and from the local network interface and port specified in the
last two arguments.
Parameters:
host – destination host address
port – destination port
tag – 802.1 tag
interface – local address
localPort – local port

Throws: IOException
if an I/O error occurs while creating the socket
getEtherAddress
  function getEtherAddress(socket: EtherSocket) return EtherAddress;
  Returns the remote host the socket is connected to or, if the connection is now closed, which host the socket was connected was connected to when it was connected.

Parameters:
  socket – Ethernet socket

Returns:
  the remote Ethernet address to which the socket is connected

getPort
  function getPort(socket: EtherAddress) return PORT_TYPE;
  Returns the port the socket is, or was or will be, connected to on the remote host.

Parameters:
  socket – Ethernet socket

Returns:
  the port connected to on the remote host

getLocalPort
  function getLocalPort(socket: EtherAddress) return PORT_TYPE;
  Returns the port number for the local host.

Parameters:
  socket – Ethernet socket

Returns:
  the local port number

getLocalAddress
  function getLocalAddress(socket: EtherAddress) return EtherAddress;
  Gets the local address to which the socket is bound.

Parameters:
  socket – Ethernet socket

Returns:
  the local address

getInputStream
  function getInputStream(socket: EtherSocket) return STREAM_ACCESS;
  Returns an input stream for this socket.

Parameters:
  socket – Ethernet socket

Returns:
  A reference to an input stream for reading bytes from this socket.

Throws:
  IOException
  if an I/O error occurs while creating the output stream.

getOutputStream
  function getOutputStream(socket: EtherSocket) return STREAM_ACCESS;
  Returns an output stream for this socket.

Parameters:
  socket – Ethernet socket

Returns:
  A reference to an output stream for writing bytes to this socket.

Throws:
  IOException
  if an I/O error occurs while creating the output stream.

close
  procedure close(socket: in EtherSocket);
  Closes the socket.

Parameters:
  socket – Ethernet socket

Throws:
  IOException
  if an I/O error occurs while closing the socket.
5.4 EtherServerSocket Subroutines

makeEtherServerSocket

    function makeEtherServerSocket(port: PORT_TYPE)
        return EtherServerSocket;
    Creates a server socket on the port specified by the argument.

    Parameters:
    port – local port

    Throws: BindException
    if the socket cannot be created and bound to the requested port, or if another server socket is already
    using the requested port

makeEtherServerSocket

    function makeEtherServerSocket(port: PORT_TYPE;
        tag: EtherTag)
        return EtherServerSocket;
    Creates a server socket on the port, and based on the tag, specified by the arguments.

    Parameters:
    port – local port
    tag – 802.1 tag

    Throws: BindException
    if the socket cannot be created and bound to the requested port, or if another server socket is already
    using the requested port

makeEtherServerSocket

    function makeEtherServerSocket(port: PORT_TYPE;
        queueLength: POSITIVE)
        return EtherServerSocket;
    Creates a server socket on the specified port with the specified queue length (in bytes) for incoming
    connection requests.

    Parameters:
    port – local port
    queueLength – queue length

    Throws: BindException
    if the socket cannot be created and bound to the requested port, or if another server socket is already
    using the requested port

makeEtherServerSocket

    function makeEtherServerSocket(port: PORT_TYPE;
        queueLength: POSITIVE;
        tag: EtherTag)
        return EtherServerSocket;
    Creates a server socket on the specified port with the specified queue length (in bytes) for incoming
    connection requests.

    Parameters:
    port – local port
    queueLength – queue length
    tag – 802.1 tag

    Throws: BindException
    if the socket cannot be created and bound to the requested port, or if another server socket is already
    using the requested port

makeEtherServerSocket

    function makeEtherServerSocket(port: PORT_TYPE;
        queueLength: POSITIVE;
        bindAddress: EtherAddress)
        return EtherServerSocket;
    Creates a server socket on the specified port with the specified queue length to hold incoming connection
    requests; the socket binds only to the specified Ethernet address.

    Parameters:
    port – local port
    queueLength – queue length
    bindAddress – address to bind to
Throws: BindException
if the socket cannot be created and bound to the requested port, or if another server socket is already
using the requested port

makeEtherServerSocket
function makeEtherServerSocket(port: PORT_TYPE;
    queueLength: POSITIVE;
    tag: EtherTag;
    bindAddress: EtherAddress)
    return EtherServerSocket;
Creates a server socket on the specified port with the specified queue length and tag to hold incoming
connection requests; the socket binds only to the specified Ethernet address.
Parameters:
port – local port
queueLength – queue length
bindAddress – address to bind to
tag – 802.1 tag
Throws: BindException
if the socket cannot be created and bound to the requested port, or if another server socket is already
using the requested port

accept
function accept(socket: EtherServerSocket) return EtherSocket;
Listen for a connection to be made to this socket and accepts it. The method blocks until a connection is
made.
Parameters:
socket – Ethernet server socket
Throws: IOException
if an I/O error occurs while waiting for a connection.

close
procedure close(socket: in EtherServerSocket);
Closes this socket.
Parameters:
socket – Ethernet server socket
Throws: IOException
if an I/O error occurs while closing the socket.

getEtherAddress
function getEtherAddress(socket: EtherServerSocket) return EtherAddress;
Returns the local address of this server socket.
Parameters:
socket – Ethernet server socket
Returns:
the local address.

getLocalPort
function getLocalPort(socket: EtherServerSocket) return PORT_TYPE;
Determines the local port being listened on.
Parameters:
socket – Ethernet server socket
Returns:
the local port number.

tag
function getTag(socket: EtherServerSocket) return EtherTag;
Returns the tag of this server socket.
Parameters:
socket – Ethernet server socket
Returns:
The tag; otherwise, a null is returned if the no tag is associated with this socket.
5.5 Datagram Subroutines

makeDatagram

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

Creates a datagram object for receiving data. The received datagram’s data is stored in buffer until the appropriate UD PDU is filled or until length bytes have been written into the buffer.

Parameters:
buffer – array of bytes
length – number of bytes

Throws: IllegalArgumentException
if the specified length overflows the buffer

makeDatagram

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

Creates a datagram object for receiving data. The received datagram’s data is stored in buffer, beginning at buffer[offset], until the appropriate UD PDU is filled or until length bytes have been written into the buffer.

Parameters:
buffer – array of bytes
offset – offset, in bytes
length – number of bytes

Throws: IllegalArgumentException
if the specified length overflows the buffer

makeDatagram

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

Creates a datagram for sending data. The datagram is filled with length bytes of data. The destination points to the host the datagram is to be delivered to; the port is the destination port on that host.

Parameters:
data – array of bytes
length – number of bytes
destination – destination address
port – destination port

Throws: IllegalArgumentException
if the length is greater that size of the data array

makeDatagram

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

Creates a datagram for sending data. The datagram is filled with length bytes of data starting at offset. The destination points to the host the datagram is to be delivered to; the port is the destination port on that host.

Parameters:
data – array of bytes
offset – offset, in bytes
length – number of bytes
destination – destination address
port – destination port

Throws: IllegalArgumentException
if the length is greater that size of the data array
**getAddress**

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

Returns the address of the remote host from which the datagram was received.

**Parameters:**
- d – Datagram

**Returns:**
- the remote host address
```

**getPort**

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

Returns the remote port from which the datagram was received.

**Parameters:**
- d – Datagram

**Returns:**
- the remote port number
```

**getData**

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

Returns a byte array containing the data from the datagram.

**Parameters:**
- d – Datagram

**Returns:**
- array of bytes
```

**getLength**

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

Returns the number of bytes in the datagram.

**Parameters:**
- d – Datagram

**Returns:**
- Number of bytes
```

**getOffset**

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

Returns the point in the array returned by `getData` where the data from the datagram begins.

**Parameters:**
- d – Datagram

**Returns:**
- Point in array where data begins
```

**setData**

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

Changes the payload of the datagram.

**Parameters:**
- d – Datagram
- data – byte array
```

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

Sends data in length pieces beginning at offset.

**Parameters:**
- d – Datagram
- data – byte array
- offset – offset
- length: size of data chunk
```

**setAddress**

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

Changes the destination address of a datagram.
```
**Parameters:**

- \( d \) – datagram
- remote – Remote Ethernet address

**setPort**

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

Changes the port a datagram is addressed to.

**Parameters:**

- \( d \) – datagram
- port – destination port

**setLength**

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

Changes the number of bytes in the internal buffer so datagrams are not truncated between receptions.

**Parameters:**

- \( d \) – datagram
- length – Length in bytes

### 5.6 DatagramSocket Subroutines

**makeDatagramSocket**

```pascal
function makeDatagramSocket return DatagramSocket;
```

Creates a socket bound to an anonymous port. The same socket may be used to receive datagrams that a server sends back to it.

**Throws:** SocketException

if the socket cannot be created.

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

Creates a socket that listens for incoming datagrams on a specific port, specified by the port argument.

**Parameters:**

- port – listening port

**Throws:** SocketException

if the socket cannot be created.

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

Creates a socket that listens for incoming datagrams on a specific port, specified by the port argument, and specified tag, specified by the tag argument.

**Parameters:**

- port – listening port
- tag – 802.1 tag

**Throws:** SocketException

if the socket cannot be created.

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

Creates a socket that listens for incoming datagrams on a specific port and network interface. This constructor is especially useful for a multi-homed host.

**Parameters:**

- port – listening port
- address – Ethernet address of the host

**Throws:** SocketException

if the socket cannot be created.

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

Creates a socket that listens for incoming datagrams on a specific port, tag and network interface. This constructor is especially useful for a multi-homed host.

**Parameters:**
- port – listening port
- tag – 802.1 tag
- address – Ethernet address of the host

**Throws:** SocketException
if the socket cannot be created.

**send**

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

Sends a single datagram dp over the network using this datagram socket.

**Parameters:**
- socket – datagram socket
- d – datagram object

**Throws:** IOException
if datagram to be sent is larger than can be supported by the native software

**receive**

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

Receives a single datagram from the network and stores it in the datagram d.

**Parameters:**
- socket – datagram socket
- d – datagram object

**Throws:** IOException
If there’s a problem receiving the data

**close**

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

Frees the port occupied by the socket.

**Parameters:**
- socket – datagram socket

**getLocalPort**

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

Returns the local port on which the socket is listening.

**Parameters:**
- socket – datagram socket

**Returns:**
the local port

**connect**

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

Enables the capability to send datagrams to and receive datagrams from the specified remote host on the specified remote port.

**Parameters:**
- socket – datagram socket
- host – Ethernet address
- port – remote port

**disconnect**

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

Disables the capability of the socket so that it can send datagrams to, and receive datagrams from, any host and port.

**Parameters:**
- socket – datagram socket
getPort

function getPort(socket: DatagramSocket) return PORT_TYPE;
    Returns the remote port to which the socket is connected.

Parameters:
    socket – datagram socket

Returns:
    the remote port used by the connection; otherwise, a null is returned if the socket is not connected.

getEtherAddress

function getEtherAddress(socket: DatagramSocket) return EtherAddress;
    Returns the address of the remote host to which the socket is connected.

Parameters:
    socket – datagram socket

Returns:
    The address of the remote host; otherwise, a null is returned if the socket is not connected.

getTag

function getTag(socket: DatagramSocket) return EtherTag;
    Returns the tag associated with the socket.

Parameters:
    socket – datagram socket

Returns:
    The tag; otherwise, a null is returned if no tag is associated with this socket.

5.7 MulticastSocket Subroutines

makeMulticastSocket

function makeMulticastSocket return MulticastSocket;
    Creates a multicast socket bound to an anonymous port. A recipient replies to the same port.

Throws:
    SocketException if the socket cannot be created

makeMulticastSocket

function makeMulticastSocket(port: PORT_TYPE) return MulticastSocket;
    Creates a multicast socket on a specific port.

Parameters:
    port – source port

Throws:
    SocketException if the socket cannot be created, e.g., if the port is already in use

makeMulticastSocket

function makeMulticastSocket(port: PORT_TYPE; tag: EtherTag) return MulticastSocket;
    Creates a multicast socket on a specific port using a specified tag

Parameters:
    port – source port
    tag – 802.1 tag

Throws:
    SocketException if the socket cannot be created, e.g., if the port is already in use

joinGroup

procedure joinGroup(socket: MulticastSocket; address: in EtherAddress);
    Once a multicast socket is created, this method allows it to join a multicast group.

Parameters:
    socket – multicast socket
    address – Ethernet address

Throws:
    IOException if the address is not a group address

leaveGroup

procedure leaveGroup(socket: MulticastSocket; address: in EtherAddress);
Once a multicast socket has joined a group, it can leave it by calling this method.

**Parameters:**
socket – multicast socket
address – Ethernet address

**Throws:** IOException
if the address is not a group address

**setInterface**
procedure setInterface(socket: MultiastSocket;
address: in EtherAddress);
Associates a particular network interface for multicast use on a multi-homed host.

**Parameters:**
Socket – multicast socket
address – Ethernet address

**Throws:** SocketException
if the address does exist on the local machine

**getInterface**
function getInterface(socket: MultiastSocket)
return EtherAddress;
Gets the address of the interface in use.

**Returns:**
the address in use

**send**
procedure send(socket: in MultiastSocket;
d: in Datagram);
Sends a single datagram dp over the network using this datagram socket.

**Parameters:**
socket – multicast socket
d – datagram object

**Throws:** IOException
if datagram to be sent is larger than can be supported by the native software

**receive**
procedure receive(socket: in MultiastSocket;
d: in out Datagram);
Receives a single datagram from the network and stores it in the datagram d.

**Parameters:**
d – datagram object

**Throws:** IOException
If there’s a problem receiving the data

**close**
procedure close(socket: in MultiastSocket);
Frees the port occupied by the socket.

**Parameters:**
socket – multicast socket

**getLocalPort**
function getLocalPort(socket: MultiastSocket)
return PORT_TYPE;
Returns the local port on which the socket is listening.

**Parameters:**
socket – Multicast socket

**Returns:**
the local port

**connect**
procedure connect(socket: in MultiastSocket;
host: in EtherAddress;
port: in PORT_TYPE);
Enables the capability to send datagrams to and receive datagrams from the specified remote hosts on the specified remote port.
disconnect

procedure disconnect(socket: in MulticastSocket);
Disables the capability of the socket so that it can send datagrams to, and receive datagrams from, any
host and port.
Parameters:
socket – Multicast socket

getPort

function getPort(socket: MulticastSocket) return PORT_TYPE;
Returns the remote port to which the socket is connected.
Parameters:
socket – Multicast socket
Returns:
the remote port used by the connection; otherwise, a -1 is returned if the socket is not connected.

getEtherAddress

function getEtherAddress(socket: MulticastSocket) return EtherAddress;
Returns the address of the remote host to which the socket is connected.
Parameters:
socket – Multicast socket
Returns:
The address of the remote host; otherwise, a null is returned if the socket is not connected.

getTag

function getTag(socket: MulticastSocket) return EtherTag;
Returns the tag associated with the socket.
Parameters:
socket – Multicast socket
Returns:
The tag; otherwise, a null is returned if no tag is associated with this socket.
SERIES OF ITU-T RECOMMENDATIONS

<table>
<thead>
<tr>
<th>Series</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Organization of the work of ITU-T</td>
</tr>
<tr>
<td>B</td>
<td>Means of expression: definitions, symbols, classification</td>
</tr>
<tr>
<td>C</td>
<td>General telecommunication statistics</td>
</tr>
<tr>
<td>D</td>
<td>General tariff principles</td>
</tr>
<tr>
<td>E</td>
<td>Overall network operation, telephone service, service operation and human factors</td>
</tr>
<tr>
<td>F</td>
<td>Non-telephone telecommunication services</td>
</tr>
<tr>
<td>G</td>
<td>Transmission systems and media, digital systems and networks</td>
</tr>
<tr>
<td>H</td>
<td>Audiovisual and multimedia systems</td>
</tr>
<tr>
<td>I</td>
<td>Integrated services digital network</td>
</tr>
<tr>
<td>J</td>
<td>Cable networks and transmission of television, sound programme and other multimedia signals</td>
</tr>
<tr>
<td>K</td>
<td>Protection against interference</td>
</tr>
<tr>
<td>L</td>
<td>Construction, installation and protection of cables and other elements of outside plant</td>
</tr>
<tr>
<td>M</td>
<td>TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits</td>
</tr>
<tr>
<td>N</td>
<td>Maintenance: international sound programme and television transmission circuits</td>
</tr>
<tr>
<td>O</td>
<td>Specifications of measuring equipment</td>
</tr>
<tr>
<td>P</td>
<td>Telephone transmission quality, telephone installations, local line networks</td>
</tr>
<tr>
<td>Q</td>
<td><strong>Switching and signalling</strong></td>
</tr>
<tr>
<td>R</td>
<td>Telegraph transmission</td>
</tr>
<tr>
<td>S</td>
<td>Telegraph services terminal equipment</td>
</tr>
<tr>
<td>T</td>
<td>Terminals for telematic services</td>
</tr>
<tr>
<td>U</td>
<td>Telegraph switching</td>
</tr>
<tr>
<td>V</td>
<td>Data communication over the telephone network</td>
</tr>
<tr>
<td>X</td>
<td>Data networks and open system communications</td>
</tr>
<tr>
<td>Y</td>
<td>Global information infrastructure and Internet protocol aspects</td>
</tr>
<tr>
<td>Z</td>
<td>Languages and general software aspects for telecommunication systems</td>
</tr>
</tbody>
</table>