INTERNATIONAL TELECOMMUNICATION UNION

ITU-T X.606.1

(02/2003)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI networking and system aspects – Networking

Information technology – Enhanced Communications Transport Protocol: Specification of QoS management for simplex multicast transport

ITU-T Recommendation X.606.1

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	**X.600–X.629**
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999

For further details, please refer to the list of ITU-T Recommendations.

INTERNATIONAL STANDARD ISO/IEC 14476-2
ITU-T RECOMMENDATION X.606.1

Information technology – Enhanced Communications Transport Protocol: Specification of QoS management for simplex multicast transport

Summary

This Recommendation | International Standard (the second part of ECTP) specifies an end-to-end multicast transport protocol to support QoS negotiation, monitoring and maintenance functions in a simplex (1-to-n) multicast connection. This protocol can be used by multicast applications such as real-time multimedia streaming services for supporting their QoS requirements.

If QoS management is enabled, QoS parameter negotiation can optionally be performed during the connection creation phase. The controlling sender then arbitrates parameter values from those proposed by the receivers. If negotiation is not enabled, the sender imposes predefined values.

The level of QoS achieved is monitored during the data transfer phase. Each receiver measures the QoS values actually experienced, and reports the status of each parameter to its parent controller using modified ACK packets. The sender aggregates the parameter status values reported from the receivers to obtain a view of connection QoS status. The sender takes QoS maintenance actions, such as data transmission rate adjustment, based upon analysis of the connection QoS status.

Source

ITU-T Recommendation X.606.1 was prepared by ITU-T Study Group 17 (2001-2004) and approved on 13 February 2003. An identical text is also published as ISO/IEC 14476-2.

Keywords

ECTP, multicast transport protocol, QoS, simplex multicast .

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 not 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.

CONTENTS

Introduction

This Recommendation | International Standard specifies the Enhanced Communications Transport Protocol (ECTP), which is a transport protocol designed to support Internet multicast applications running over multicast-capable networks. ECTP operates over IPv4/IPv6 networks that have the IP multicast forwarding capability with the help of IGMP and IP multicast routing protocols, as shown in Figure 1. ECTP could possibly be provisioned over UDP.

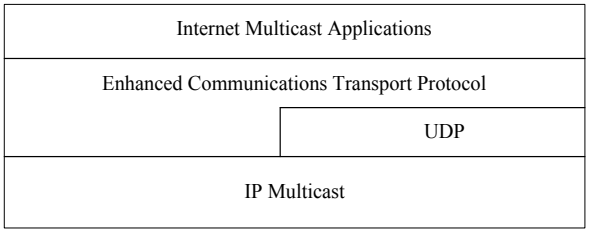


Figure 1 – ECTP model

ECTP is targeted to support tightly controlled multicast connections in simplex, duplex and N-plex applications. This part of ECTP (part 2) specifies the QoS management functions for stable management of the QoS of connection users in a simplex multicast connection. QoS management functionality consists of QoS negotiation, QoS monitoring, and QoS maintenance operations. The protocol procedures for reliability control in simplex multicast transport are defined in ECTP part 1 (ITU-T Rec. X.606 | ISO/IEC 14476-1), which forms an integral part of this Recommendation | International Standard. Further parts of the standard will define control procedures and associated QoS management functions respectively for the duplex case (X.ectp-3 | ISO/IEC 14476-3 and X.ectp-4 | ISO/IEC 14476-4) and for the N-plex case (X.ectp-5 | ISO/IEC 14476-5 and X.ectp-6 | ISO/IEC 14476-6).

In ECTP, all prospective members are enrolled into a multicast group, before a connection or session is created. Those members define an enrolled group. Each receiver in the enrolled group is referred to as an enrolled receiver. In the enrolment process, each member will be authenticated. The group information, including group key and IP multicast addresses and port numbers, will be distributed to the enrolled members during the enrolment process. An ECTP connection is created for these enrolled group members.

The sender is at the heart of multicast group communications. A single sender in the simplex multicast connection is assigned the role of the connection owner, designated top owner (TO) in this Specification. The connection owner is responsible for overall connection management by governing the connection creation and termination, the connection pause and resumption, and the late join and leave operations.

The sender triggers the connection creation process. Some or all of the enrolled receivers will participate in the connection, becoming designated "active receivers". Receivers active at this stage are able to participate in negotiating the desired quality of service for the session. Any enrolled receiver that is not active at this stage may participate in the connection as a late-joiner, but will have to accept the established QoS. An active receiver can leave the connection.

After the connection is created, the sender begins to transmit multicast data. While the connection is active, the sender monitors the status of the session via feedback control packets from the active receivers.

The sender may take a range of actions if network problems (such as severe congestion) are indicated by the feedback received from active receivers. These actions include adjusting the data transmission rate, suspending multicast data transmission temporarily, or in the last resort, terminating the connection.

This QoS management specification can be used in the multicast applications that want to support various QoS requirements and the corresponding billing/charging models.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Enhanced Communications Transport Protocol: Specification of QoS management for simplex multicast transport

1 Scope

This Recommendation | International Standard is an integral part of ITU-T Recs X.606.x | ISO/IEC 14476 "ECTP: Enhanced Communications Transport Protocol", which is a family of Protocol Specifications designed to support multicast transport services.

ITU-T Rec. X.606 | ISO/IEC 14476-1 provides a specification of various protocol operations for simplex multicast transport. Those protocol operations include connection management such as connection creation/termination and connection pause/resume, membership management such as late join, user leave and membership tracking, and error control for multicast data transport such as error detection and recovery.

This part of the Recommendation | International Standard provides a specification of QoS management for accomplishing desirable quality of service in simplex multicast transport connection.

This Specification describes the following QoS management operations:

a) *QoS negotiation*

   For QoS negotiation, this Specification assumes that a desired QoS level for multicast application service can be expressed in terms of a set of QoS parameters. QoS negotiation is performed via exchange of control packets between sender and receivers. Sender proposes the target values of QoS parameters obtained from the application's requirements, and then each receiver can propose modified values based on its system and/or network capacity. Sender arbitrates the modified values proposed by receivers. Target values for QoS parameters can be used as input parameters for reservation of network resources.

b) *QoS monitoring*

   QoS control in ECTP is based on feedback of control packets from receivers. The feedback messages from receivers enable the sender to keep track of the number of active receivers and also to monitor the connection status for multicast data transport. QoS monitoring is designed to allow the sender to diagnose the connection status in terms of QoS parameter values, and thus to take the necessary actions for maintaining the connection status at a desired QoS level. The monitored connection status will be reported to the application at the sender side. The information conveyed could provide statistics useful for billing purposes, for example.

c) *QoS maintenance*

   Based on feedback information from receivers, the sender takes one or more actions so as to maintain the connection status at a desired QoS level. These QoS maintenance actions include adjustment of the data transmission rate, connection pause and resume, troublemaker ejection and connection termination operations. These QoS monitoring and maintenance functions, based on monitored parameter status, provide rate-based congestion control.

This Recommendation | International Standard is an integral part of ITU-T Recs X.606.x | ISO/IEC 14476, which has 6 parts. All of the protocol components, including packet formats and protocol procedures specified in ITU-T Rec. X.606 | ISO/IEC 14476-1, are also valid in this Recommendation | International Standard.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- ITU-T Recommendation X.601 (2000), Multi-peer communications framework.
- ITU-T Recommendation X.605 (1998) | ISO/IEC 13252:1999, *Information technology – Enhanced Communications Transport Service definition*.

– ITU-T Recommendation X.606 (2001) | ISO/IEC 14476-1:2002, *Information technology – Enhanced Communications Transport Protocol: Specification of Simplex Multicast Transport*.

3 Definitions

3.1 Terms defined in ITU-T Rec. X.605 | ISO/IEC 13252

This Recommendation | International Standard is based on the concepts developed in Enhanced Communications Transport Service (ITU-T Rec. X.605 | ISO/IEC 13252).

a) QoS parameters;

b) QoS negotiation; and

c) QoS arbitration.

3.2 Terms defined in ITU-T Rec. X.606 | ISO/IEC 14476-1

This Recommendation | International Standard is described based on the concepts and terms developed in the specification of simplex multicast transport on ECTP (ITU-T Rec. X.606 | ISO/IEC 14476-1).

a) application;

b) packet;

c) sender;

d) receiver;

e) tree;

f) parent; and

g) child.

3.3 Terms defined in this Recommendation | International Standard

For the purposes of this Recommendation | International Standard, the following definitions apply:

a) **QoS monitoring**: Is the protocol operation that is used to diagnose the current connection status. For QoS monitoring, each receiver is required to measure the experienced parameter values and to report them to sender. Sender aggregates the status information reported from receivers.

b) **QoS maintenance**: Is the protocol operation that is used to maintain the connection status at a desired QoS level. Sender takes QoS maintenance actions based on the monitored status information.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply.

4.1 Packet types

ACK	Acknowledgment
CC	Connection Creation Confirm
CR	Connection Creation Request
CT	Connection Termination
DT	Data
HB	Heartbeat
JC	Late Join Confirm
JR	Late Join Request
LR	Leave Request
ND	Null Data
RD	Retransmission Data

4.2 Miscellaneous

API	Application Programming Interfaces
CHQ	Controlled Highest Quality
Diffserv	Differentiated Services
ECTP	Enhanced Communications Transport Protocol
ECTS	Enhanced Communications Transport Services
IP	Internet Protocol
LQA	Lowest Quality Allowed
MSS	Maximum Segment Size
OT	Operating Target
QoS	Quality of Service
RSVP	Resource Reservation Protocol

5 Conventions

In this Recommendation | International Standard, the key words "MUST", "REQUIRED", "SHALL", "MUST NOT", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" indicate requirement levels for compliant ECTP implementations.

6 Overview

This Recommendation | International Standard provides a specification of QoS management for one-to-many (simplex) multicast transport connections. This Specification describes the following QoS management operations:

1) QoS negotiation, including reservation of network resources;
2) QoS monitoring; and
3) QoS maintenance.

In the connection creation phase, sender informs the receivers whether QoS management is enabled. When QoS management is enabled, sender must also specify whether or not QoS negotiation will be performed in the connection. QoS monitoring and maintenance operations are performed, only if QoS management is enabled.

Figure 2 illustrates these QoS management operations for the simplex multicast connection. In the figure, the protocol operations marked as dotted lines are specified in ITU-T Rec. X.600 | ISO/IEC 14476-1.

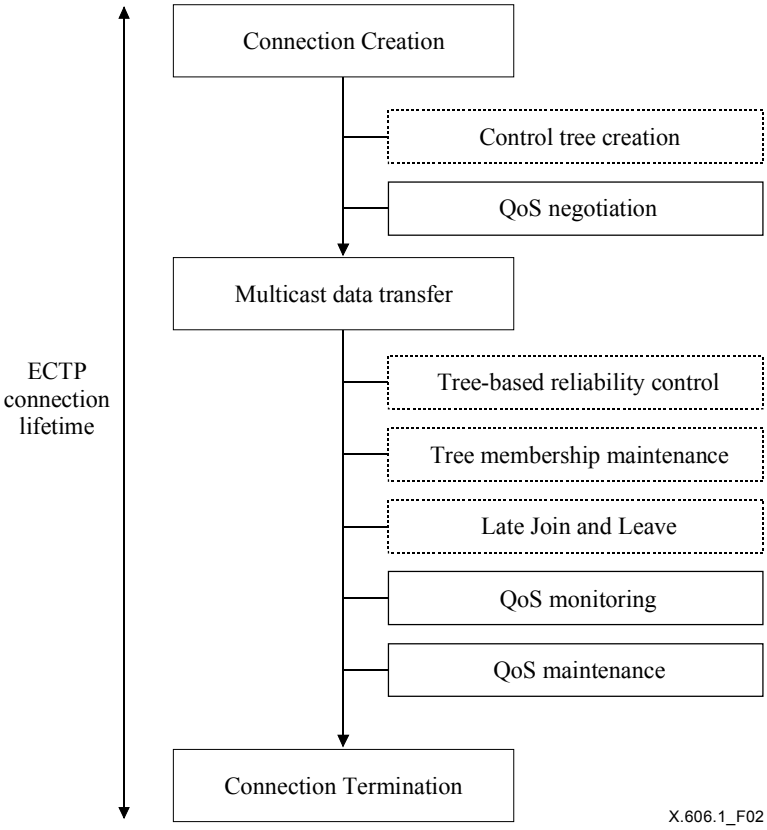


Figure 2 – QoS management in ECTP

In general, QoS represents the quality of service required for satisfactory reception of application data at a receiver, to achieve desirable audio/video display quality for example. In this Specification, it is assumed that the QoS requirements of an application are expressed in terms of one or more QoS parameters such as throughput, transit delay, transit delay jitter, and data loss rate. Depending on the application's requirements, some of these QoS parameters may not be used in the connection. For example, a non-real time service might not impose the transit delay requirement.

From the requirements of applications, sender will determine the target values for each QoS parameter. How to map from the application's requirements to those target parameter values is outside the scope of this Specification. Application programs could be used to carry out such mappings.

QoS negotiation is performed in the connection creation phase. Sender proposes the desired target values for each QoS parameter to all receivers by multicast. For throughput, three target values are specified: CHQ (controlled highest quality), OT (operating target) and LQA (lowest quality allowed). For the other parameters such as transit delay, transit delay jitter, and data loss rate, only two target values are specified: OT and LQA.

If QoS negotiation is enabled, each receiver can propose modifications to the sender's proposed parameter values. These modified values will be determined by considering the system capacity at the receiver side and network environments. The following restrictions are imposed for modification of parameter values by receivers:

1) OT values must not be modified by receivers;
2) the values modified by receivers must be within LQA and CHQ values proposed by sender.

The parameter values modified by receivers are delivered to sender via ACK messages. The sender arbitrates different parameter values for various receivers by taking a commonly agreed range of values.

Figure 3 shows an abstract sketch of QoS negotiation that can occur in ECTP. From the application's requirements, a set of target QoS parameter values will be configured at the sender. Sender informs the receivers about the target values (step 1). Based on those target values, each receiver begins to make resource reservations with the help of RSVP or Diffserv (step 2). If QoS negotiation is enabled in the connection, each receiver may propose modified values for QoS parameters (step 3). From the modified parameter values, the sender determines the arbitrated values (step 4). These arbitrated values are delivered to the receiver via subsequent HB or JC packets, and will be used for QoS monitoring and maintenance.

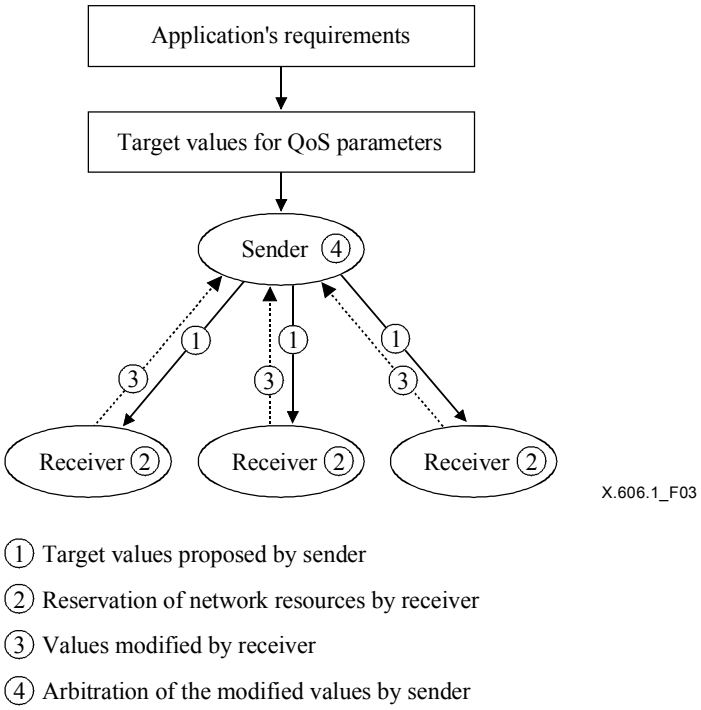


Figure 3 – QoS negotiation

After an ECTP connection is created, and if QoS management is enabled, the QoS monitoring and maintenance operations are performed for the multicast data transmission. For QoS monitoring, each receiver is required to measure the parameter values experienced. Based on the measured values and the negotiated values, a receiver determines a parameter status value for each parameter as an integer: normal (0), reasonable (1), possibly abnormal (2), or abnormal (3). These status values will be delivered to the sender via ACK packets.

Sender aggregates the parameter status values reported from the receivers. If a control tree is employed, each parent LO node aggregates the measured values reported from its children, and forwards the aggregated value(s) to its own parent using ACK packets.

Figure 4 illustrates the QoS monitoring and maintenance operations described in this Specification.

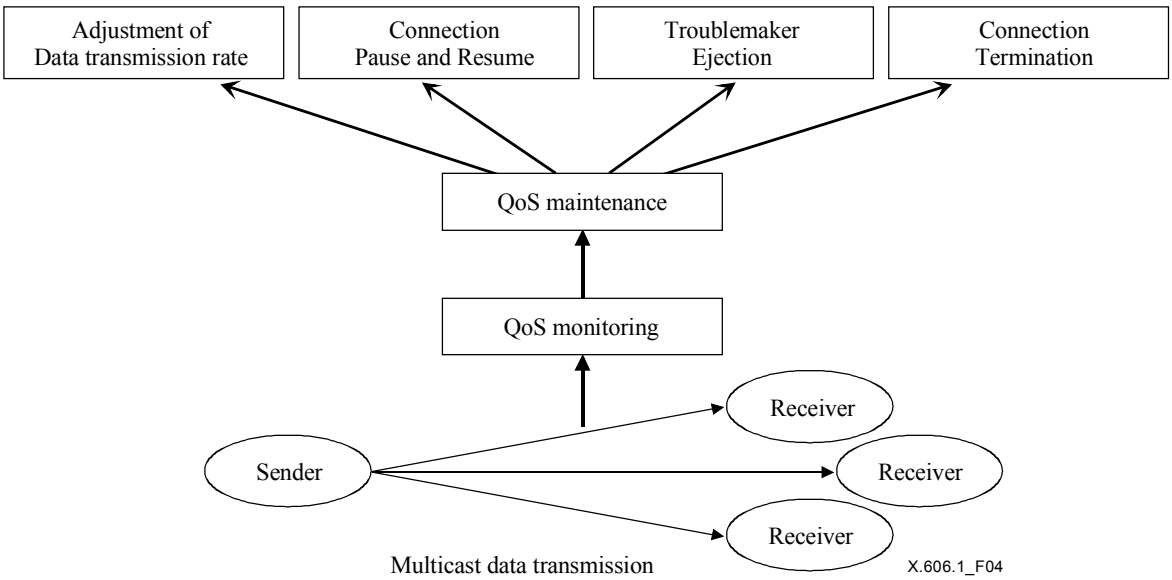


Figure 4 – Conceptual sketch of QoS monitoring and maintenance

Sender takes QoS maintenance actions necessary to maintain the connection status at a desired QoS level, based on the monitored status values. Specific rules are pre-configured to trigger QoS maintenance actions such as data transmission rate adjustment, connection pause and resume, troublemaker ejection and connection termination. Those actions will be taken by observing how many receivers are in the abnormal or possibly abnormal state.

7 Components for QoS management

This clause describes the ECTP protocol components required for QoS management operations. All the components are extended from those already defined in ITU-T Rec. X.606 | ISO/IEC 14476-1.

7.1 Connection information element

Figure 5 shows the connection information element specified in ITU-T Rec. X.606 | ISO/IEC 14476-1.

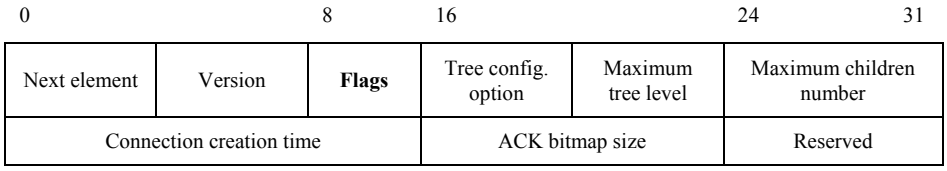
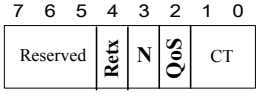


Figure 5 – Connection information element

For QoS management, the ECTP sender specifies the following three fields in the '**Flags**' byte:



1) *QoS* – is a flag bit to indicate whether QoS management is enabled (1) or not (0) in the connection. If this bit is set to '1', all the procedures for QoS management are invoked. The default value is '0'.
2) *N* (*negotiation*) – is a flag bit to indicate whether QoS negotiation is enabled (1) or not (0) in the connection. If this bit is set to '1', each receiver is allowed to propose its own parameter values. The default value is '0'.
3) *Retx* (*retransmission*) – is a flag bit to indicate whether retransmissions by parent are performed (0) or not (1). If this bit is set to '1', the sender or parents need not retransmit RD packets, even if the retransmission requests by ACK packets are received. The default value is '0'.

The QoS bit must be set to "1" (QoS enabled) before the N bit is valid. There are three possible cases.

a) QoS bit set to "1" and N bit set to "0" indicates that QoS is to be used in the connection, and QoS values will be imposed by the sender. The receivers cannot negotiate it.

b) Both bits set to "1" indicate that QoS is to be used in the connection, and QoS parameter values may be negotiated between receivers and the sender.

c) QoS bit set to "0" indicates that QoS is not to be used in the connection. The N bit is not used in this case.

The setting of the Retx (retransmission bit) can be done independently of the setting of the QoS bit. It is expected that real-time live streaming media applications will not need error recovery based on retransmissions, but they need the QoS management functions. Even in this case, ACK packets are still used to convey connection status information.

7.2 QoS parameters

In this Specification, the following four QoS parameters are defined:

1) throughput (bytes per second);
2) transit delay (millisecond);
3) transit delay jitter (millisecond);
4) data loss rate (percent).

Throughput represents an amount of application data output over a specific time period. Target throughput means a throughput value required for desirable display of application data. Applications generate multicast data and ECTP sender will transmit them, based on the target throughput value(s). Actual data reception rate at receiver's side will depend on data transmission rate, network conditions and end system capacity, etc.

For throughput, the sender shall configure the following target values:

1) CHQ throughput;
2) OT throughput;
3) LQA throughput.

Among them, the following inequalities must be enforced: LQA throughput ≤ OT throughput ≤ CHQ throughput.

Transit delay represents end-to-end transmission time from a sender to a receiver. For desirable display of multicast data, the sender may configure the following target values:

1) OT transit delay;
2) LQA transit delay.

Between them, the following inequalities must be enforced: OT transit delay ≤ LQA transit delay.

Transit delay jitter represents variations of transit delay values. For desirable display of data, the sender may configure the following target values:

1) OT transit delay jitter;
2) LQA transit delay jitter.

Between them, the following inequalities must be enforced: OT transit delay jitter ≤ LQA transit delay jitter.

Data loss rate is defined as a ratio of the amount of lost data over the amount of totally transmitted data. For desirable display of data, the sender may configure the following target values:

1) OT loss rate;
2) LQA loss rate.

Between them, the following inequalities must be enforced: OT loss rate ≤ LQA loss rate.

7.3 QoS extension element

For QoS management, the QoS extension element is newly defined in this Specification. All the extension elements used in ECTP are listed below.

Table 1 – Encoding table of ECTP extension elements

Element	Encoding
Connection Information	0001
Acknowledgment	0010
Tree Membership	0011
Timestamp	0100
QoS	0101
No element	0000

The QoS extension element specifies the Maximum Segment Size (*MSS*) and the target values for ECTP QoS parameters described in 7.2. As shown in Figure 6, the QoS element has a length of '28' bytes:

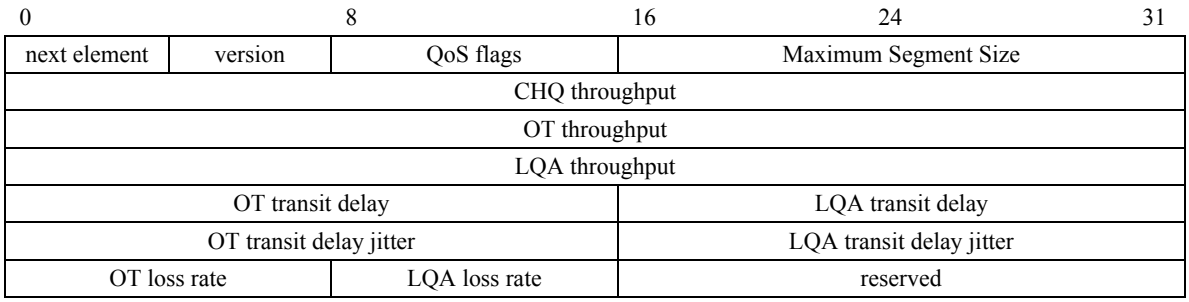


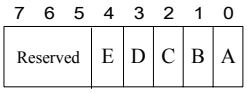
Figure 6 – QoS extension element

The following parameters are specified:

a) *Next Element* – Indicates the type of the next element immediately following this QoS element;

b) *Version* – Defines the current version of this element, starting at '1';

c) *QoS flags* – Is a flag byte to specify if each of QoS parameters and *MSS* are used in the connection. Encoding of this byte is depicted in the following figure. If a bit is set to '1', then the corresponding QoS parameter or MSS will be used. The default value is '0' for each bit.



1) A – throughput;
2) B – transit delay;
3) C – transit delay jitter;
4) D – data loss rate;
5) E – maximum segment size (*MSS*);
6) *Reserved* – is reserved for future use.

d) *Maximum Segment Size* (*MSS*) – Represents the maximum size of an ECTP segment or packet in unit of bytes. If the 'E' bit of the QoS flags is set to '1', *MSS* is subject to negotiation (see 8.1.3). Otherwise, the default *MSS* value of 1024 will be used.

e) *Throughput values* – Each value is a 32-bit unsigned integer in byte per second. The following target values are valid only if 'A' bit of QoS flags is set to '1':

1) CHQ throughput – upper limit for throughput;
2) OT throughput – target throughput for desired display of multicast data;
3) LQA throughput – lower limit for throughput.

f) *Transit delay values* – Each value is a 16-bit unsigned integer in millisecond. The following target values are valid only if 'B' bit of QoS flags is set to '1':

1) OT transit delay – target transit delay for desired display of multicast data;
2) LQA transit delay – maximally allowed transit delay.

g) *Transit delay jitter values* – Each value is a 16-bit unsigned integer in millisecond. The following target values are valid only if 'C' bit of QoS flags is set to '1':
1) OT transit delay jitter – target transit delay jitter for desired display of multicast data;
2) LQA transit delay jitter – maximally allowed transit delay jitter.

h) *Data loss rate values* – Each value is an 8-bit unsigned integer ranged from 0 to 100 in percent. The following target values are valid only if 'D' bit of QoS flags is set to '1':
1) OT loss rate – target loss rate for desired display of multicast data;
2) LQA loss rate – maximally allowed loss rate.

i) *Reserved* – Is reserved for future use.

The QoS element is used for the sender to inform the receivers about the target values for QoS parameters by a CR packet in the connection creation phase. In QoS negotiation, the QoS element is also used when a receiver proposes its own modified values to the sender. The negotiated QoS values will be announced to late-joiners via the JC packet and to existing receivers via HB packets.

These QoS values are also referred to in the QoS monitoring and maintenance operations.

7.4 Acknowledgement element

For QoS monitoring, each receiver is required to measure the parameter values that have been experienced. A measured parameter value is mapped to a parameter status value. A status value is an integer such as 0, 1, 2, or 3. A larger status value indicates a worse status for the connection.

The status values are delivered to sender via ACK packets. The acknowledgement element of the ACK packet contains the status values for QoS parameters used in the connection.

The acknowledgement element specified in ITU-T Rec. X.606 | ISO/IEC 14476-1 is shown below. In the figure, the '*parameter status*' byte is newly defined in this Specification.

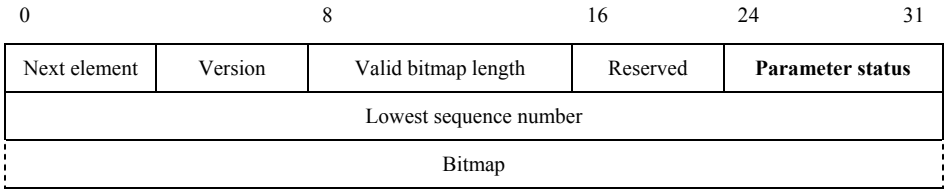
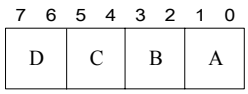


Figure 7 – Acknowledgement element

The '*parameter status*' byte has the following structure:



a) *A* – Represents two bits to indicate the status value for the measured throughput;
b) *B* – Represents two bits to indicate the status value for the measured transit delay;
c) *C* – Represents two bits to indicate the status value for the measured transit delay jitter;
d) *D* – Represents two bits to indicate the status value for the measured packet loss rate.

A status value consisting of two bits has one of the following values:

a) 00 – Indicates '0' as a status value;
b) 01 – Indicates '1' as a status value;
c) 10 – Indicates '2' as a status value;
d) 11 – Indicates '3' as a status value.

The detailed mapping schemes from a measured parameter value to a status value are described in 8.2.3.

7.5 Packets used for QoS management

Table 2 lists the ECTP packets used for QoS management.

Table 2 – ECTP packets used for QoS management

Packet type	Extension element				
	Connection information	Tree membership	Acknowledgement	Timestamp	QoS
CR	O				O
CC		O			O
HB		O		O	O
ACK		O	O	O	
JC	O				O

The CR packet contains a QoS element. This is used by the sender to propose (or impose) the target values of the QoS parameters that are used in the connection. These values can be referenced by resource reservation mechanisms and protocols, such as RSVP, if they are enabled in the network. If QoS negotiation is enabled, each receiver responds to the sender with its own proposed values for QoS parameters, via a CC packet. The sender will arbitrate the returned proposals, and the arbitrated values for the QoS parameters will be delivered to receivers via HB packets. For a late-joiner, the target QoS parameter values currently being used in the connection (whether imposed or negotiated originally) are notified via a JC packet (see 8.1).

The target or negotiated values will be referred to in QoS monitoring and maintenance operations. ACK packets are used to convey the status values for QoS parameters experienced at the receiver side (see 8.2).

8 Procedures for QoS management

In ECTP, the QoS management includes the following operations:

1) QoS negotiation, possibly with reservation of network resources;
2) QoS monitoring;
3) QoS maintenance.

QoS negotiation is performed in the connection creation phase, while the QoS monitoring and maintenance operations will be done in the data transmission phase.

If QoS management is enabled in the connection, the QoS monitoring and maintenance operations will be performed by default. On the other hand, QoS negotiation is enabled only when the *N* bit in the 'Flags' byte of the connection information element is set to '1'.

8.1 QoS negotiation

The ECTP sender transmits a CR packet to all receivers to start the connection creation phase. The CR packet contains the proposed (or imposed) target values for each QoS parameter such as CHQ, OT, and LQA. Each receiver can refer to these target values for resource reservation (see 8.1.4). If QoS negotiation is enabled in the connection, the negotiation procedures are activated (see 8.1.1). The imposed or negotiated target values are subsequently used in QoS monitoring and maintenance (see 8.2 and 8.3).

If QoS negotiation is enabled in the connection, each receiver can propose a new modified value in response to a target parameter value proposed by the sender. To propose a new value, a receiver is required to be able to identify the system or network resources to be used. For example, a modified throughput value may be assessed from line rates of transmission links accessible at the receiver site (e.g., DSL, cable modem, and wireless networks, etc). The modified value may also be determined by considering the end user's requirement at a receiver site. It is possible for an end host to use a software program to determine a modified parameter value for negotiation, based on network and system resources as well as end user's requirements. However, in real world scenarios, it is not easy to precisely identify the resource capacity of the networks involved with a receiver. Accordingly, at least in the near future, QoS negotiation will be done based on the end user's requirements at the application level or on the system capacity of the end host.

In this Specification, the sender is required to specify via the QoS extension element whether each QoS parameter is subject to negotiation (see 7.3). For the parameters that are negotiable, a receiver can propose modified values. If a receiver does not wish to modify a QoS parameter, it will just return the same QoS element received from the sender.

8.1.1 Negotiation procedures

If QoS negotiation is enabled in the connection, each receiver responds to the sender with a CC packet containing the modified target values for the respective QoS parameters.

This subclause describes the QoS negotiation procedures for the throughput parameter, which has three target values: LQA, OT, and CHQ. The negotiation procedures for the other parameters such as delay, jitter and loss rate are all the same except that these parameters have no CHQ values.

During QoS negotiation, receivers must not modify the OT value for each parameter.

The detailed procedures for QoS negotiation are described below and illustrated in Figure 8.

1) Sender proposes target parameter values:

   From application requirements, sender determines the target parameter values: LQAo, OTo, CHQo, where LQAo < OTo < CHQo, and then transmits a CR packet with QoS extension element to all receivers.

2) Receivers modify the parameter values:

   In response to the target values proposed by sender, each receiver *i* can propose the modified values: LQA_i and CHQ_i. OTo value must not be changed. Thus, the following inequalities are enforced: LQAo < LQA_i < OTo < CHQ_i < CHQo for each receiver *i*. Each receiver delivers the modified values to sender via a CC packet.

3) Sender arbitrates the modified parameter values:

   The sender arbitrates the modified parameter values proposed by receivers as follows:

   $$CHQmin = \min CHQ_i, \text{ for each receiver } i$$

   $$LQAmax = \max LQA_i, \text{ for each receiver } i$$

   LQAmax and CHQmin are the negotiated parameter values that have resulted from QoS negotiation.

4) Sender announces the negotiated parameter values:

   Sender announces LQAmax, CHQmin, and OT values to receivers via HB and JC packets.

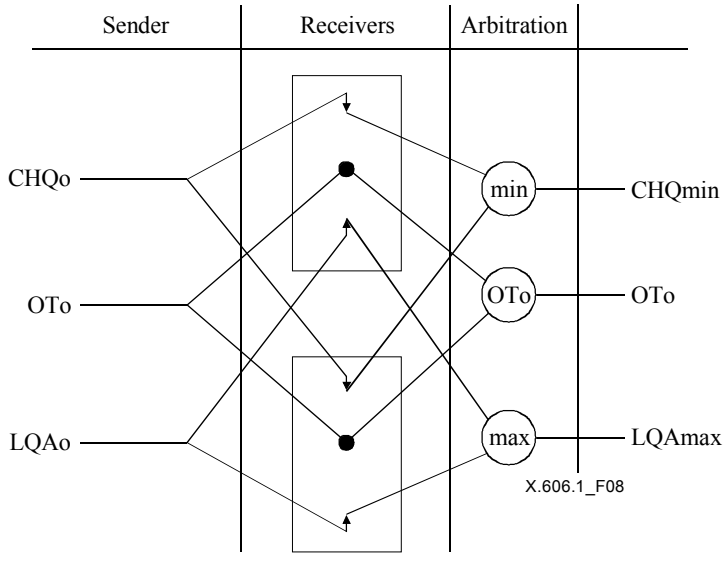


Figure 8 – QoS negotiation procedures

For delay, jitter, and loss rate parameters, an LQAmin value will be obtained instead of LQAmax, since CHQ values are not used and OT value < LQA value for those parameters. That is,

$$LQAmin = \min LQA_i, \text{ for each receiver } i$$

QoS negotiation is not performed for late-joining receivers. Sender just notifies the negotiated parameter values to the late-joining receiver via JC packet.

8.1.2 QoS negotiation in the tree hierarchy

If a control tree with more than two levels is employed, each parent LO must perform the QoS arbitration procedures for the modified values proposed by its children (step 3 in 8.1.1). In this case, the arbitration rule is as follows;

$$CHQmin' = \min CHQ_i, \text{ for each child } i$$

$$LQAmax' = \max LQA_i, \text{ for each child } i$$

The parent delivers the arbitrated values, CHQmin' and LQAmax', to its parent via CC packet. In this way, the sender receives CC packets from all of its children, and then obtains the arbitrated parameter values for all the receivers.

8.1.3 MSS negotiation

MSS represents the maximum packet size, which depends on *Maximum Transmission Unit* (*MTU*) for link layer transmissions. Typical *MTU* values are 1500 bytes for Ethernet, 1492 bytes for IEEE 802, 4352 bytes for FDDI, 576 bytes for X.25, and so on. The *MTU* determines the frame size for the link layer transmissions, and hence the *MSS* value.

The default *MSS* value of the ECTP is 1024 bytes, which supports most link types except X.25. If sender and/or receiver cannot identify its *MSS* value, it takes the default *MSS* value of 1024 bytes.

In ECTP, if *MSS* negotiation is indicated (see 7.3), the following procedures are performed:

1) Sender writes its *MSS* value into the CR packet, and then transmits it to all the receivers.
2) Each receiver proposes its own *MSS* value via the CC packet. The *MSS* will be affected by the *MTU* of the local network that the receiver belongs to. If the receiver's *MSS* is larger than the sender's *MSS*, the receiver takes the sender's *MSS*.
3) Sender arbitrates the connection's *MSS* by taking the minimum *MSS* value for all the receivers.

8.1.4 Resource reservation

ECTP itself cannot guarantee the QoS levels required by applications. However, the target values of QoS parameters can be used in the reservation of network resources such as Integrated Services with RSVP and Differentiated Services (Diffserv).

The RSVP model is a good match with the ECTP protocol, since the resource reservation is done in an end-to-end manner. In networks where RSVP is enabled, periodic RSVP PATH messages will be delivered to receivers. In response to the PATH message, each receiver sends periodic RSVP RESV messages to the sender via the multicast data path.

If RSVP is used with ECTP, the target values of ECTP QoS parameters will be referenced in the configuration of RSVP traffic descriptors for the RSVP TSPEC (or FLOWSPEC) contained in the RSVP PATH message. These PATH messages will be issued in accordance with CR packet and HB packets transmitted by the sender. It is noted that the RSVP modules are managed separately from ECTP in end systems. This means that the resource reservation and control functions in RSVP will only be performed only by the RSVP daemons concerned.

Annex A describes an example of interworking between ECTP and RSVP, along with a possible mapping scheme between the associated QoS and traffic parameters.

The differentiated service model provides differentiated classes of service for IP traffic, to support various types of applications, and specific business requirements. A small bit-pattern in each packet, in the IPv4 TOS (or DSCP) octet or the IPv6 Traffic Class octet, is used to mark a packet to receive a particular forwarding treatment, or per-hop behaviour, at each network node. A common understanding of the use and interpretation of this bit-pattern is required for inter-domain use, multi-vendor interoperability, and consistent reasoning about expected service behaviours in a network. Up to the present, a specific interworking scheme between ECTP and Diffserv has not been identified.

ECTP will need to utilize underlying network QoS mechanisms such as RSVP and Diffserv to establish network connections delivering the required QoS levels. However, the potential use of ECTP is not limited to current network QoS methods. ECTP is designed to be used for QoS improvement under a variety of different QoS models in the future.

8.2 QoS monitoring

The QoS monitoring function provides the sender with information about how well the connection is operating. To do this, each receiver is required to measure the parameter values experienced and report these values back to the sender.

For the QoS parameters used in the connection, each receiver measures the parameter values that have been experienced. The measured value is mapped to a parameter status value for each parameter. A parameter status is an integer having a value of 0, 1, 2, or 3. This status value is recorded in the acknowledgement element and conveyed to the sender via a subsequent ACK packet (see 7.4). Sender aggregates the parameter status values from all the receivers.

The purpose of QoS monitoring is to provide the sender with information on QoS status for the connection. Based on the monitored status information, the sender can also take any QoS maintenance actions necessary.

8.2.1 Generation of ACK

Each receiver reports the *parameter status values* to its parent by generating ACK packets. In ECTP-2, the ACK generation rule is somewhat different from that specified in ECTP-1. In ECTP-2, *AGT* is set to equal *AGN*. More specifically, *AGT* is set to be '*AGN* × 1 second, i.e., *AGN* seconds.' Accordingly, each receiver will generate periodic ACK packets for every *AGT* second. Setting of *AGN* is an implementation issue, but *AGN* is set to '8' by default in ECTP-2.

To generate ACK packet, each receiver must keep a timer, QoS monitoring time (*QMT*), in seconds. The QMT timer starts as soon as the receiver completes the connection setup, i.e. after reception of CC or JC packet from the sender. QMT timer increases monotonically as the connection progresses. That is, it will not be refreshed during the connection.

Each receiver transmits an ACK packet to its parent if:

$$QMT \% AGN = Child\ ID \% AGN$$

This scheme is employed to minimize ACK implosions at the parent side as much as possible. By this mechanism, each receiver will continue to generate ACK packets every *AGN* second, after the *QMT* timer begins. For example, if *AGN* is set to 8, then a receiver with *Child ID* of either 3 or 11 will generate ACK packets at the *QMT* times of 3, 11, 19, 27 seconds, etc. It is noted in the example that the first ACK will be generated for data packets transmitted only during 3 seconds not 8 seconds. However, the other succeeding ACK packets are generated for the time interval of *AGN* seconds. In this manner, each receiver will generate ACK packets every *AGN* second, except for the first ACK packet.

Each ACK packet conveys the measured parameter status value as described in 8.2.2.

8.2.2 Measurement of QoS parameter values

For multicast data transmission, each receiver measures the experienced values for each QoS parameter. All the parameter values are measured, recorded and calculated, until a new ACK packet is generated according to the ACK generation rule described above. When it is time to send an ACK, a receiver calculates the parameter status value for the data packets received and collected until then. After transmitting an ACK, the collected data is cleared and then new data will be gathered and recorded for generating a new ACK.

8.2.2.1 Throughput

Throughput is measured as the data reception rate in units of bytes per second. The data reception rate is calculated by:

Amount of the received data packets in bytes over AGN seconds.

Again, the first ACK packet may be generated before *AGN* seconds have elapsed. To measure the throughput value, a receiver needs to keep information about how much data packets (bytes) have been received for the specific time.

Each time a new throughput value is obtained, the value is mapped to the *parameter status value*, an integer value of 0, 1, 2, or 3, according to the mapping rule that will be described in 8.2.3.

8.2.2.2 Data loss rate

Data loss rate represents the packet loss rate, and it is expressed in percent. The packet loss rate is calculated by:

Number of lost packets over Number of the data packets received during AGN seconds.

Again, the first ACK packet may be generated before *AGN* seconds have elapsed. To measure the throughput value, a receiver needs to keep information about how many data packets have been lost. When a receiver generates an ACK packet, the currently measured loss rate value is mapped to the *parameter status value*.

8.2.2.3 Transit delay and jitter

To measure an end-to-end transit delay, the sender is required to transmit data packets with a timestamp element. Another requirement is the synchronization of time clocks between sender and receivers. Without satisfactory resolution of these requirements, it is hard to get exact information about transit delay and jitter. In this subclause, it is assumed that each receiver can measure an end-to-end transit delay of a data packet from the sender.

Transit delay is measured for each data packet received. These transit delay values are averaged over the data packets received during *AGN* seconds. Transit delay jitter is measured as the difference between the maximum and minimum transit delay values of those received data packets.

During the *AGN* seconds, every time a new data packet arrives, the transit delay is calculated, and the averaged delay and jitter values are updated. Just before a receiver generates an ACK packet, the currently measured value will be mapped to the *parameter status value*.

8.2.3 Mapping to a parameter status value

The measured parameter value is mapped to a parameter status value for each QoS parameter. The subsequent ACK packet will contain the status value. Note again that measurement of a QoS parameter is activated only if the use of the parameter in the connection is indicated (see 7.3).

Figure 9 illustrates the mapping from the measured value to a status value.

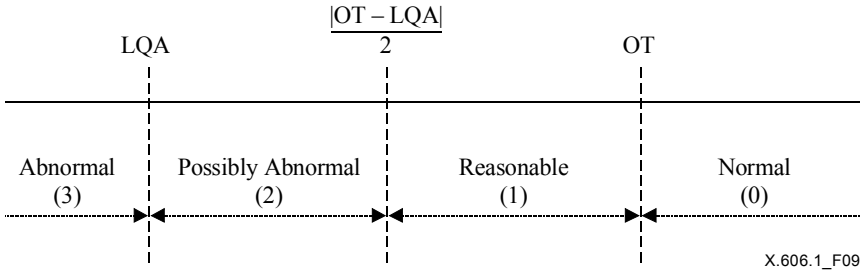


Figure 9 – Mapping of the measured value to a status value

As shown in the figure, the mapping from the measured parameter value to a status value is done based on OT and LQA parameter values. ECTP also uses a *threshold* value to classify the status into normal (0), reasonable (1), possibly abnormal (2), and abnormal (3). The *threshold* value is set to a median value between OT and LQA parameter values (i.e., |LQA – OT| / 2), as shown in the figure.

An initial *parameter status value* is set to '0'. Once the measured value is obtained, the mapping to a *parameter status value* is done as follows:

IF "measured parameter value > OT, " then status = 0*;*

ELSE IF "threshold < measured parameter value ≤ OT," then status = 1*;*

ELSE IF "LQA < measured parameter value ≤ threshold, "then status = 2*;*

ELSE IF "measured value ≤ LQA," then status = 3.

In the above mapping rules, the inequalities hold true only for throughput parameter. For the other parameters such as delay, jitter, and loss rate, those inequalities must be reversed, because OT values ≤ LQA values.

8.2.4 Reporting toward the sender

Each receiver reports the obtained *parameter status values* to its parent via ACK packets. The ACK packets are generated based on *child ID* and *AGN* seconds. In this fashion, all the information about *parameter status values* will be delivered toward the sender along the tree hierarchy.

8.2.4.1 Aggregation by LO in the tree hierarchy

In the tree hierarchy, each parent LO aggregates the ACK packets from its children. This aggregation will proceed just before the parent generates its own ACK. The LO also generates its own ACK packet every *AGN* second, as done by a

receiver. The parent simply takes an average value over the parameter status values reported from its children together with its own measured value, just before it generates ACK packets.

The averaged value is calculated for each QoS parameter by:

Summation of parameter status values from the responding children over CCN.

In the multi-level tree structure, the number of descendants is represented by *Active Receiver Number* (*ARN*) recorded in the tree membership element. In this case, the parameter status values reported by children are weighted by their *ARN* values as follows:

$$\text{Weighted average value} = \Sigma_{i=1, \ldots, CCN} \{ARN(i) \times \text{parameter status value}(i)\} \text{ over } \Sigma_{i=1, \ldots, CCN} ARN(i)$$

A parent LO rounds this average value to 0, 1, 2, or 3 for each QoS parameter, and composes its ACK packet.

8.2.4.2 Aggregation by sender

Aggregation of parameter status values by sender is the same as that by LO. The sender also performs aggregation of ACK packets reported from its children every *AGN* seconds.

After aggregation of ACK packets, the sender simply takes an average value weighted by *ARN* of each child. That is,

Aggregated status value = the weighted sum of reported status values over ARN of the connection.

The sender obtains an aggregated status value for each QoS parameter. More specifically, the sender will have the following aggregated status values (if each of the parameters is used in the connection):

a) aggregated status for throughput, denoted by *Tvalue*;

b) aggregated status for transit delay, denoted by *Dvalue*;

c) aggregated status for transit delay jitter, denoted by *Jvalue*;

d) aggregated status for data loss rate, denoted by *Lvalue*.

Each of the aggregate values is also ranged between 0 and 3.

Sender may forward the monitored status information to the application. The monitored information is helpful for the sending application to diagnose how well the connection is being operated in terms of QoS, which may further be useful for designing a billing/charging model.

The monitored information is also used in the QoS maintenance. Among the monitored values, *Lvalue* is used for adjustment of transmission data rate (see 8.3.1). A weighted sum value for all the status values, *Tvalue, Dvalue, Jvalue,* and *Lvalue*, can be used to trigger connection pause, troublemaker ejection and termination.

8.3 QoS maintenance

QoS maintenance is performed to maintain the quality of a connection at a desired level and to prevent the connection quality from being degraded below the negotiated QoS level.

Based on the monitored parameter status values, the sender will take the following QoS maintenance actions:

1) adjustment of data transmission rate;

2) connection pause and resume;

3) troublemaker ejection;

4) connection termination.

Data rate adjustment is related to the rate-based flow and congestion control. Connection pause/resume and termination are the actions which can be taken to manage the connection. These events will be announced to all the receivers via ND and CT packets transmitted by the sender.

To trigger these QoS maintenance actions, the sender needs to configure the following threshold values:

1) *threshold_rate_increase* and *threshold_rate_decrease* for adjustment of data transmission rate;

2) *threshold_connection_pause*.

All the threshold values are real numbers ranged between 0 and 3.

8.3.1 Adjustment of data transmission rate

ECTP uses a fixed-sized window-based flow control. The default *window size* is the same as *ACK Bitmap Size* (*ABS*), 32. Sender can maximally transmit the *window size* data packets at the rate of *Data Transmission Rate* (*DTR*). ECTP performs congestion control by dynamically adjusting *DTR*, based on the loss rate status values *Lvalue* (see 8.2.4.2).

Adjustment of data transmission rate is based on *threshold_rate_increase* and *threshold_rate_decrease*, which are pre-configured by sender from application requirements. These values are ranged as follows:

$$0 \leq threshold\_rate\_increase \leq threshold\_rate\_decrease \leq 3$$

The default values are *threshold_rate_increase* = 1.0 and *threshold_rate_decrease* = 2.0.

In the data transmission phase, sender starts with *DTR* = *LQA* throughput, and *DTR* can be adjusted as follows:

$$LQA \text{ throughput} \leq DTR \leq CHQ \text{ throughput.}$$

Every AGN second, the sender adjusts *DTR*, based on *threshold_rate_increase*, *threshold_rate_decrease*, and the monitored *Lvalue* as follows:

IF Lvalue < threshold_rate_increase

THEN DTR = Min {CHQ, DTR + Transmission Rate Increase (TRI)}

ELSF IF threshold_rate_increase ≤ Lvalue ≤ threshold_rate_decrease

THEN DTR is not changed

ELSE IF Lvalue > threshold_rate_decrease

THEN DTR = Max {LQA, DTR – Transmission Rate Decrease (TRD)}

Rate adjustment variables such as *TRI* and *TRD* may be set based on *CHQ* and *LQA* throughput. For examples,

$TRI = (CHQ - LQA) \times 1/20$

$TRD = (CHQ - LQA) \times 1/5$

8.3.2 Connection pause and resume

Connection pause can be performed by the sender to suspend the multicast data transmissions temporarily so as to prevent the connection quality from being more severely degraded.

Connection pause and resume may be performed, according to the request of application. In this case, the sending application will trigger the connection pause, based on monitored parameter status values such as *Tvalue, Dvalue, Jvalue* and *Lvalue*. If the connection pause is triggered, the sender transmits periodic ND packets with the *F* bit set to '1' in the fixed header (see ITU-T Rec. X.606 | ISO/IEC 14476-1). The sender must not transmit any new DT packet, while the control packets including HB packets can be sent. Each receiver may also send control packets such as ACK.

Connection pause may also be triggered, based on the pre-configured *threshold_connection_pause*. In this case, only if the monitored connection status value is larger than *threshold_connection_pause*, the ECTP sender will trigger the connection pause. The suggested *threshold_connection_pause* value is 2.5.

For this purpose, *Connection Status* value is calculated for all the monitored parameter status values as follows.

$$Connection\ Status = Tweight \times Tvalue + Dweight \times Dvalue + Jweight \times Jvalue + Lweight \times Lvalue.$$

Each of the weight values must also be configured, along with *threshold_connection_pause*, where the following constraints are imposed:

$$0 \leq Tweight, Dweight, Jweight, Lweight \leq 1$$

$$Tweight + Dweight + Jweight + Lweight = 1$$

A weight value is set to '0' if the corresponding QoS parameter is not enabled in the connection.

The connection pause is triggered if:

$$Connection\ Status \geq threshold\_connection\_pause$$

After connection pause was indicated, if the *Connection Pause Time* (*CPT)* interval has elapsed, then connection resume is triggered and the sender begins to transmit multicast data at the transmission rate of *LQA*. When connection resume is indicated, ND packets will set the *F* bit of the header to '0'.

8.3.3 Troublemaker ejection

The sender or LO may invoke a troublemaker ejection to maintain the QoS status at a desired level and also to prevent the connection status from being more severely degraded. A detailed scheme of the troublemaker ejection can be made differently by implementations, based on the parameter status values provided in this Specification.

For example, a receiver may be ejected by its parent, if it has reported a *parameter status value* larger than *threshold_connection_pause* several times more than a pre-configured threshold. The design and implementation of the troublemaker ejection scheme must be done carefully, since the ejection operation may have significant impact on the overall ECTP protocol behaviour.

8.3.4 Connection termination

The natural option for the connection termination is to terminate a connection when all the multicast data have been transmitted. In QoS management operations, connection termination is also triggered if the connection status is perceived as 'unrecoverable'.

Connection termination may be performed according to the request of the application. If the connection termination is triggered, the sender transmits a CT packet to all the receivers, and closes the connection.

Connection termination may also be triggered, based on the pre-configured *Connection Termination Time* (*CTT*). In this case, connection termination is triggered if:

Subsequent connection pause occurs again within *CTT* from connection resume.

The *CTT* timer is activated when connection resume is indicated. Connection termination may not be supported by some applications.

9 Timers and variables

The following are timers and variables used for QoS management.

9.1 Timers

a) ACK Generation Time (*AGT*) in seconds: Each receiver generates periodic ACK packet every *AGT* second, except for the first ACK packet. In ECTP-2, *AGT* is set to *AGN* seconds (see 8.2).

b) Connection Pause Time (*CPT*) in seconds: Once connection pause is indicated, the connection will pause during *CPT* interval (see 8.3.2).

c) Connection Termination Time (*CTT*) in seconds: After connection resume is indicated, if the connection pause occurs again within *CTT* interval, connection termination is triggered (see 8.3.4).

d) QoS Monitoring Time (*QMT*) in seconds: Each receiver measures the experienced values for QoS parameters every *QMT* interval (see 8.2.2).

9.2 Operation variables

a) Aggregated parameter status values: Sender aggregates the parameter status values reported from receivers, which results in *Tvalue, Dvalue, Jvalue,* and *Lvalue* (see 8.2).

b) ACK Generation Number (*AGN*): Each receiver generates periodic ACK packet every *AGT* second. In ECTP-2, *AGT* is set to *AGN* seconds (see 8.2). By default, *AGN* is set to 8.

c) Connection Status: For triggering the connection pause/resume, all the aggregated parameter status values can be weight-averaged with the pre-configured parameter weights such as *Tweight, Dweight, Jweight,* and *Lweight*. This results in *Connection Status* that represents overall status of the connection (see 8.3.2).

d) Data Transmission Rate (*DTR*): Sender transmits multicast data at the rate of *DTR* (see 8.3.1).

e) Measured parameter value: Each receiver is required to measure the experienced value for QoS parameters used in the connection. This results in *measured parameter value* (see 8.2).

f) Parameter status value: The *measured parameter value* is mapped onto a *parameter status value*, which is an integer such as 0, 1, 2, or 3 (see 8.2).

g) Transmission Rate Decrease (*TRD*): *DTR* is decreased by *TRD* (see 8.3.1).

h) Transmission Rate Increase (*TRI*): *DTR* is decreased by *TRI* (see 8.3.1).

i) *Threshold_rate_increase*: Threshold to increase data transmission rate (see 8.3.1).

j) *Threshold_rate_decrease*: Threshold to decrease data transmission rate (see 8.3.1).

k) *Threshold_connection_pause*: Threshold to trigger the connection pause (see 8.3.2).

Annex A

Interworking between ECTP and RSVP for resource reservation

(This annex does not form an integral part of this Recommendation | International Standard)

ECTP QoS can be guaranteed by interworking with RSVP/Intserv for network resource reservation protocols. Based on the QoS parameter values, RSVP can configure its traffic descriptors if it is enabled in the network.

This annex describes how to use the RSVP signalling along with ECTP for resource reservation and how to map ECTP QoS parameters to RSVP traffic descriptors. All the schemes illustrated below presume the non-negotiation mode (see 7.1 and 8.1), in which the QoS parameter values proposed by sender are enforced to all receivers without negotiation. How to use RSVP in the negotiation mode is for further study. However, even in the negotiation mode, the parameter mapping relationship between RSVP and ECTP is still valid. In this case, each of the 'negotiated' ECTP QoS parameter values will be mapped onto the RSVP traffic descriptors.

A.1 ECTP QoS parameters

ECTP QoS parameters and their target values are summarized as follows:

a) Throughput: CHQ, OT, and LQA;

b) Transit delay: OT and LQA;

c) Transit delay jitter: OT and LQA;

d) Data loss rate: OT and LQA.

In conclusion, at least from a point of view of the current RSVP technology, the transit delay, delay jitter and loss rate cannot be explicitly supported by RSVP. The RSVP signalling just provides a strict guarantee of network-level delay requirement by using the reservation of bandwidth and handling traffic scheduler at intermediate routers on the path. The term 'delay' defined in RSVP (network-level queuing delay) is quite different from that in ECTP (end-to-end transit delay). The following subclause gives a brief summary of RSVP operations.

A.2 Overview of RSVP

The RSVP (IETF RFC 2205, 2210, 2212) includes the following RSVP objects:

a) RSVP SENDER_TSPEC;

b) RSVP ADSPEC;

c) RSVP RECEIVER_FLOWSPEC.

A.2.1 RSVP SENDER_TSPEC

The traffic specification (TSPEC) of RSVP is an object that carries the traffic descriptors generated by a sender within an RSVP session. This object includes the following traffic descriptors:

a) Token bucket specification *r* and *b*, where *r* is a token generation or leaky rate, and *b* is a bucket depth;

b) Peak rate *p*;

c) Minimum policed unit *m*;

d) Maximum packet size *M*.

The *r* and *p* are measured in bytes per second and *b*, *m* and *M* are measured in bytes. The rates *r* and *p* are the mean and maximum rates of the information flow, respectively, and *b* is a parameter that bounds the variability of the traffic emission.

The other parameters provide bounds on the packet length distribution within the flow. *M* is the maximum size of a conforming packet and *m* is such that any packet whose size is less than *m* is treated by the network policer as if it were of size equal to *m*. The *m* and *M* are not directly concerned with the ECTP.

A.2.2 RSVP ADSPEC

The advertisement specification (ADSPEC) of RSVP is an object that carries information that is generated at either data senders or intermediate network elements. The RSVP ADSEPC is flowing downstream towards receivers, and may be used and updated inside the network before being delivered to receiving applications. This information includes both parameters describing the properties of the data path, including the availability of specific QoS control services, and parameters required by specific QoS control services to operate correctly. This object includes the following traffic descriptors:

a) Bandwidth of the link with the minimum available bandwidth on the end-to-end path.

b) End-to-end transmission delays except queuing delay; these delays can be measured by two error terms, *C* and *D*. Error term *C* is the rate-dependent error term. It represents the delay a packet in the flow might experience due to the rate parameters of the flow. Error term *D* is the rate-independent, per-element error term. It represents the worst-case non-rate-based transit time variation through the service element. It is generally determined or set to boot or configuration time.

A.2.3 RSVP FLOWSPEC

This object carries reservation request information generated by receivers, and consists of RECEIVER_TSPEC and RECEIVER_RSPEC. The information in the FLOWSPEC flows upstream from data receivers toward data sources. It may be used or updated at intermediate network elements before arriving at the sending application.

RECEIVER_TSPEC is identical to SENDER_TSPEC except the MTU of the end-to-end path. On the other hand, RECEIVER_RSPEC includes the following parameters:

a) *R*: a reserved bandwidth;

b) *S*: a slack term.

R must be greater than or equal to *r* of TSPEC. The slack term *S* is in microseconds. The slack term signifies the difference between the desired delay and the delay obtained by using a reservation level *R*. This slack term can be utilized by the network element to reduce its resource reservation for the associated flow.

A.2.4 RSVP API

Figure A.1 shows the implementation structure of RSVP in a system.

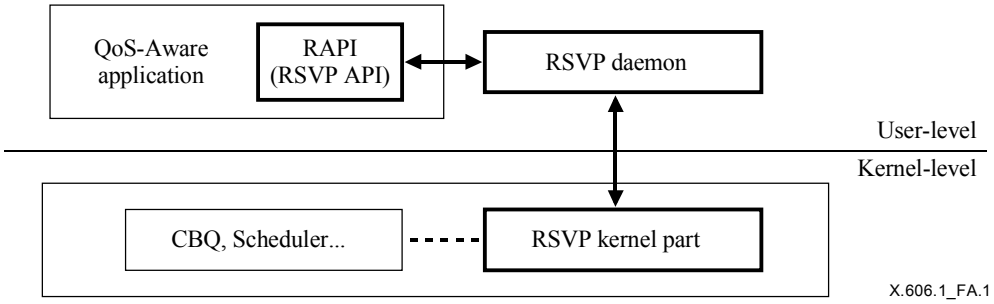


Figure A.1 – RSVP implementation structure

In a system, RSVP operates with three modules: RSVP Kernel, RSVP daemon, and RSVP (RAPI). The RSVP Kernel manages the data queuing and scheduling according to the requirements from the higher layer RSVP. The RSVP daemon handles a signalling between RAPI and RSVP Kernel. The RAPI provides an interface with QoS-aware applications, which ensures the RSVP operations to support the QoS requirements from the applications. ECTP QoS characteristics will be integrated with and enforced to the RSVP-enabled networks via RAPI.

The RAPI functions can be classified into two groups: the functions called by an application and the asynchronous (upcall) routines delivered to an application by the RASP daemon.

These basic RSVP functions are as follows:

– rapi_session(), which is used to establish an RSVP session;

– rapi_sender(), which is used by a sender to send RSVP PATH messages;

– rapi_reserve(), which is used to reserve by a receiver network resources;

– rapi_release(), which is used to terminate an RSVP session;
– rapi_getfd() and rapi_dispatch(), which are used by application to get status information about RSVP via the asynchronous upcall routines.

A.3 An example of the parameter mapping between RSVP and ECTP

Table A.1 summarizes an example of the parameter mapping between RSVP and ECTP, which can be used for QoS support along with the RSVP.

NOTE – A different mapping may also be used depending on implementation.

Table A.1 – Parameter mapping from ECTP to RSVP

RSVP parameters	ECTP QoS parameter	Description
p (TSPEC)	CHQ throughput	Directly mapped onto *p*
r (TSPEC)	OT throughput	Directly mapped onto *r*
b (TSPEC)	(CHQ throughout – OT throughput) × an arbitrary time (1 ~ 3 seconds)	Token bucket size is set arbitrarily based on ECTP throughput parameters
m (TSPEC)	IP+UDP+ECTP fixed header (20 + 8 + 16 = 44 bytes)	Directly mapped
M (TSPEC)	ECTP MSS (1024 bytes by default)	Directly mapped
R (RSPEC)	(CHQ throughput + OT throughput) / 2	The Reservation is set to the mean value of the CHQ and OT throughputs
S (RSPEC)	0	Slack term is not set

When ECTP daemon calls RSVP daemon, the ECTP QoS parameters are mapped onto the RSVP TSPEC traffic descriptors such as p, r, b, m, M, as shown in the table. These TSPEC parameter values are delivered from sender to receivers via RSVP PATH message.

Upon arrival of the RSVP PATH message at a receiver, the information in the SENDER_TSPEC and ADSPEC objects will be passed across the RSVP API to the ECTP application. The application interprets the arriving information, and uses it to select the resource reservation parameters. These parameters are composed into an RSVP FLOWSPEC object and will be transmitted to the sender via the RSVP RESV messages. Bandwidth reservation *R* and the slack term *S*, which are contained in the RSVP RSPEC object, will be mapped by using ECTP parameters, as shown in Table A.1.

A.4 A scenario of interworking between ECTP and RSVP

The target values for each ECTP QoS parameter can be used to construct an RSVP SENDER_TSPEC object. To do this, an interface between ECTP and RSVP is required so that the ECTP QoS parameter values are informed to the RSVP processor. The SENDER_TSPEC object will be delivered to receivers via RSVP PATH messages.

The time when ECTP calls RSVP depends on implementation, but it is recommended that the ECTP sender invoke RSVP after sending CR packet and that a receiver do after reception CR packet. Figure A.2 illustrates a model of interworking between ECTP and RSVP.

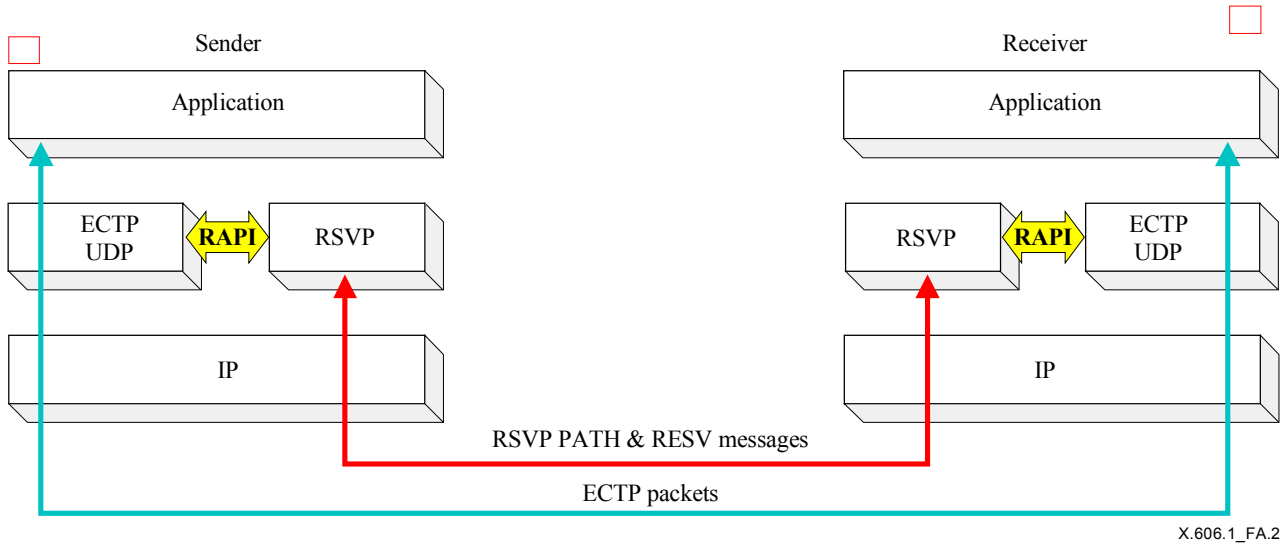


Figure A.2 – Interworking model between ECTP and RSVP

As shown in the figure, communications between ECTP and RSVP are done via RSVP API (RAPI). The ECTP sender transfers information on QoS parameters to RSVP via RAPI, when it invokes RSVP daemon. Once RSVP is triggered by ECTP, the RSVP communications will be done between sending RSVP daemon and receiving RSVP daemons. That is, RSVP peering relationship will be established between the sender and receivers.

During the ECTP connection, each RSVP module reports a status on the network resource reservation to its associated ECTP daemon via RAPI. Such a status will inform about whether the reservation has been done successfully or in an error condition. An example set of RSVP status codes is as follows:

- Status code 0: PATH events are indicated (used by the receiving RSVP).
- Status code 1: PATH error events are indicated (used by the sending RSVP).
- Status code 2: RESV events are indicated (used by the sending RSVP).
- Status code 3: RESV error events are indicated (used by the receiving RSVP).
- Status code 4: RESV_CONFIRM events are indicated (used by the receiving RSVP).

The ECTP protocol core can obtain these status codes by using the RSVP asynchronous upcall functions from the RSVP daemon.

The procedures performed by ECTP and RSVP modules for interworking between them can be summarized as follows:

(1) The ECTP sender transmits CR packet to the receivers. The CR packet contains QoS element indicating QoS parameter values required by application.

(2) The ECTP sender invokes its associated RSVP daemon.

(3) RSVP sender constructs the PATH message that contains TSPEC parameters based on ECTP QoS parameters.

(4) RSVP sender transmits the PATH messages to the receivers periodically.

(5) If an RSVP PATH error is indicated, the RSVP sender reports the corresponding status code to ECTP sender.

(6) As soon as an ECTP receiver receives a CR packet, it invokes its associated RSVP receiver daemon.

(7) Based on the PATH message received from the sender, a RSVP receiver construct receives the corresponding RESV messages including RSPEC parameters. A part of information on RSPEC such as *R* (bandwidth) and *S* (slack term) may be configured by interaction of RSVP receiver and ECTP receiver.

(8) An RSVP receiver transmits the corresponding RESV messages to the sender.

(9) When a RSVP RESV error is indicated, the RSVP receiver reports the corresponding status code to ECTP receiver.

When ECTP receives status code from its associated RSVP, it may indicate the information to its application. The detailed use of the status codes depends on implementations.

Figure A.3 illustrates the ECTP and RSVP message flows between Sender and receivers. After transmitting a HB packet, the ECTP sender can invoke an initial RSVP PATH message. The corresponding RSVP RESV messages will arrive from the ECTP receivers. The subsequent RSVP PATH messages will be triggered and repeated along with the periodic ECTP HB packets.

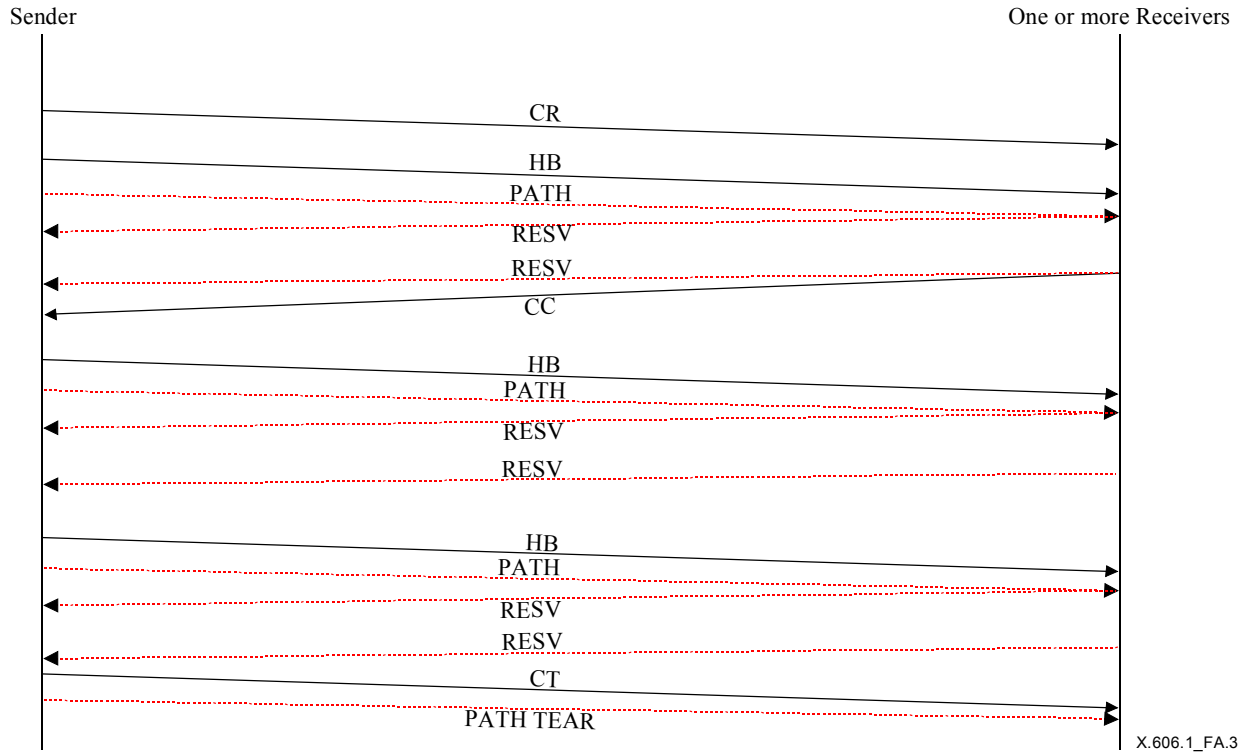


Figure A.3 – Interworking model between ECTP and RSVP

Annex B

Application Programming Interfaces

(This annex does not form an integral part of this Recommendation | International Standard)

This annex specifies the application programming interfaces (API) for ECTP. The ECTP API described in this Recommendation | International Standard can be used by applications that utilize the transport capabilities of ECTP part 1 (ITU-T Rec. X.606 | ISO/IEC 14476-1) and ECTP part 2 (ITU-T Rec. X.606.1 | ISO/IEC 14476-2).

This API has been designed based on the Berkeley socket API functions. However, to differentiate the ECTP API from the existing Berkeley socket functions, ECTP API functions are named with a prefix '*m*' (e.g., `msocket`).

B.1 Overview

B.1.1 API functions

Table B.1 summarizes API functions used for ECTP.

Table B.1 – ECTP API functions

Function name	Description
`msocket()`	Creates a new multicast socket in the ECTP communication domain.
`Mbind()`	Associates a set of local and group addresses/ports with the socket.
`Mconnect()`	Sender initiates a connection creation to a specified foreign address. Late-joining receiver initiates a join process.
`maccept()`	Prospective receivers join the ECTP connection by accepting the connection creation signal from the sender.
`Msend()`	Sends application data to a destination group.
`Mrecv()`	Delivers received data to application. Delivers some indication messages for control to application during the data transfer phase.
`mclose()`	Terminates connection and releases socket.
`mgetsockopt()`	Gets socket and protocol options from kernel.
`msetsockopt()`	Sets socket and protocol options to kernel.

B.1.2 Use of ECTP API functions

Figure B.1 illustrates an example use of ECTP API functions. The sequences of API functions invoked are shown in terms of sender, receiver, and late-joining receiver.

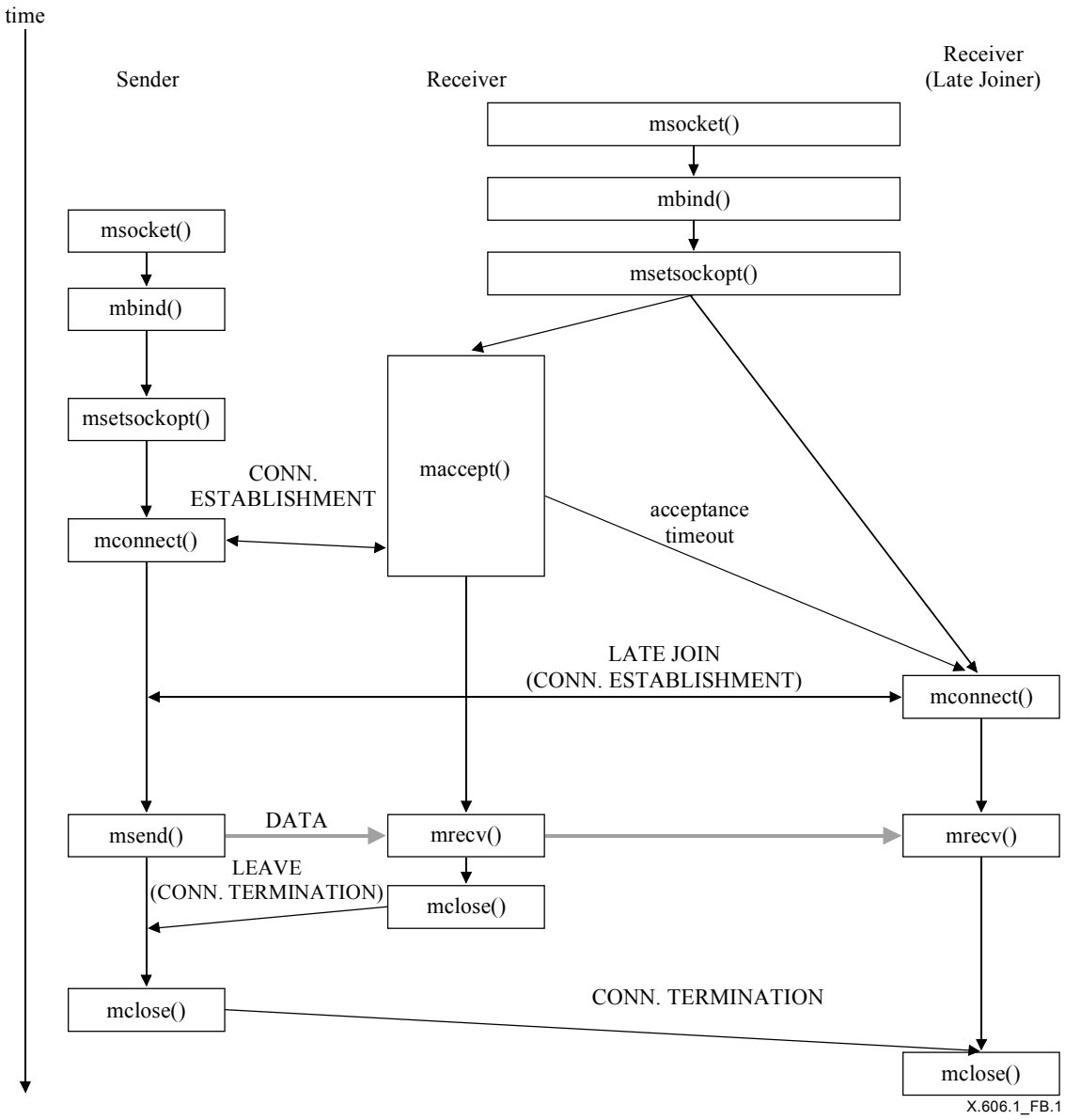


Figure B.1 – Use of ECTP API

As shown in the figure, a receiver takes different actions according to its node type: an early joiner or a late-joiner. An early joiner goes into the `maccept` mode after `msetsockopt`, while a late-joiner invokes `mconnect` function directly. An early joiner may go into `mconnect` mode after a specified acceptance timeout duration, if it cannot receive any connection creation indication signal.

B.2 ECTP API functions

B.2.1 msocket()

To use ECTP, an application MUST invoke the `msocket` function firstly, which specifies the type of communication protocol desired such as ECTP using IPv4, ECTP using IPv6, ECTP using OSI protocols, etc.

```
int msocket(int family, int type, int protocol);
```

Parameter Description:

- *family*: specifies the protocol family and is one of the constants shown in Table B.2;
- *type*: specifies the type of socket and one of the constants shown in Table B.3;
- *protocol*: is set to `0`.

Table B.2 – Protocol *family* constants used for `msocket` function

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_ISO	OSI domain protocols

Table B.3 – *type* of socket used for `msocket` function

type	Description
SOCK_ECTP1	socket for ECTP part 1
SOCK_ECTP2	socket for ECTP part 2

Table B.4 shows the valid combinations, along with the actual protocol that is selected by the pair.

Table B.4 – Combinations of *family* and *type* for `msocket` function

type	AF_INET	AF_INET6	AF_ISO
SOCK_ECTP1	ECTP1	ECTP1	ECTP1
SOCK_ECTP2	ECTP2	ECTP2	ECTP2

The `msocket` call returns non-negative descriptor if success, or –1 on following cases as listed in Table B.5.

Table B.5 – Error codes for `msocket` call

Error code	Description
EPROTONOSUPPORT	The protocol type or the specified protocol is not supported within this domain.
EMFILE	The per-process descriptor table is full.
ENFILE	The system file table is full.
EACCES	Permission to create a socket of the specified type and/or protocol is denied.
ENOBUFS	Insufficient buffer space is available.

B.2.2 mbind()

The `mbind` function assigns a set of local, group, control addresses, and the role of the node in the session to a socket. With the Internet protocols the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit port number.

`int mbind(int` *msockfd*`, const struct sockaddr` **laddr*`, socklen_t` *laddrlen*`, const struct sockaddr` **gaddr*`, socklen_t` *gaddrlen,* `struct sockaddr` **caddr*`, socklen_t` *caddrlen,* `int` *role* `);`

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *laddr*: is a pointer to a protocol-specific address to bind a local address to the above socket;
- *laddrlen*: is the size of the above address structure;
- *gaddr*: is a pointer to a protocol-specific address to bind a target group address to the socket;
- *gaddrlen*: is the size of the group address structure;
- *caddr*: is a pointer to a protocol-specific address to bind a control address to the socket;
- *caddrlen*: is the size of the control address structure; and
- *role*: specifies the role of this calling initiator such as TO, LO or LE.

Table B.6 – *role* of the socket user for `msocket` function

role	Description
TO	Connection creator and sender as the owner in the ECTP communications.
LO	Receiver taking a responsible for retransmissions in the tree-based hierarchy.
LE	Receiver that is not designated LO.

An application can `mbind` a specific IP address and a group network address to its socket. The source and group addresses must belong to an interface on the host.

The `mbind` call returns zero if success or –1 on the following reasons as listed in Table B.7.

Table B.7 – Error codes that `mbind` may cause

Error code	Description
EAGAIN	Kernel resources to complete the request are temporarily unavailable.
EBADF	*msockfd* is not a valid descriptor.
ENOTSOCK	*msockfd* is not a socket.
EADDRNOTAVAIL	The specified address is not available from the local machine.
EADDRINUSE	The specified address is already in use.
EACCES	The requested address is protected, and the current user has inadequate permission to access it.
EFAULT	The address parameter is not in a valid part of the user address space.
EROLE	The requested role is not valid.
NOTE – The shaded error code is newly defined for ECTP.	

B.2.3 maccept()

Only the passive session member like LE or LO can invoke this function. It can await sender's initiation for a specific timeout via `msetsockopt` and informs whether the multicast connection has established or not.

```
int maccept(int msockfd, struct sockaddr *raddr, socklen_t *raddrlen);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *raddr*: returns the protocol address of the remote connection initiator (the sender or TO); and
- *raddrlen*: is a pointer to the size of the socket address structure pointed by *raddr*.

If `maccept` is successful, it returns the same value as the first argument, *msockfd.* After that, we call this return value the *connected socket* descriptor.

The `maccept` call returns non-negative descriptor if success, or –1 on the following reasons as listed in Table B.8.

Table B.8 – Error codes used for `maccept`

Error code	Description
EBADF	The descriptor is invalid.
EINTR	The `maccept` operation was interrupted.
EMFILE	The per-process descriptor table is full.
ENFILE	The system file table is full.
ENOTSOCK	The descriptor references a file, not a socket.
EFAULT	The *addr* parameter is not in a writable part of the user address space.
EWOULDBLOCK	The socket is marked non-blocking and no connections are present to be accepted.
ECONNABORTED	A connection arrived, but it was closed while waiting on the listen queue.
ECRTIMEOUT	Indicates that the CR waiting time has expired.
NOTE – The shaded error code is newly defined for ECTP.	

B.2.4 mconnect()

The `mconnect` function is used by TO or late joining LE to establish a connection.

```
int mconnect(int msockfd, const struct sockaddr *daddr, socklen_t  daddrlen);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *daddr*: is a pointer to a protocol-specific destination address. This address may be a group address (in case that the calling initiator is TO), or a sender address (in case that the calling initiator is LO or LE);
- *daddrlen*: is the size of *daddr*.

The `mconnect` returns zero if success or –1 in the abnormal cases as listed in Table B.9.

Table B.9 – Error numbers of `mconnect` function

Error code	Description
EBADF	*msockfd* is not a valid descriptor.
ENOTSOCK	*msockfd* is a descriptor for a file, not a socket.
EADDRNOTAVAIL	The specified address is not available on this machine.
EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
EISCONN	The socket is already connected.
ECONNREFUSED	The attempt to connect was forcefully rejected.
ENETUNREACH	The network is not reachable from this host.
EADDRINUSE	The address is already in use.
EFAULT	The name parameter specifies an area outside the process address space.
EALREADY	The socket is non-blocking and a previous connection attempt has not yet been completed.
EDENIED	Indicates that the TO has declined LE's or LO's join request.
ETIMEDOUT	Connection establishment timed out without establishing a connection. Indicates that there is no response from TO.
NOTE – The shaded error codes are newly defined for ECTP.	

B.2.5 msend()

This `msend` function writes data from a buffer into the connected socket.

```
ssize_t msend (int msockfd, const void *buf, size_t buflen, int *flags);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *buf*: is a pointer to buffer to write from;
- *buflen*: is the size of *buf*; and
- *flags*: is not defined yet.

The `msend` returns the number of bytes written if success or –1 in the abnormal cases as listed in Table B.10.

Table B.10 – Error codes of `msend`

Error code	Description
EBADF	An invalid descriptor was specified.
EACCES	The destination address is a broadcast address, and SO_BROADCAST has not been set on the socket.
ENOTSOCK	The argument *msockfd* is not a socket.
EFAULT	An invalid user space address was specified for a parameter.
EMSGSIZE	The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.
EAGAIN	The socket is marked non-blocking and the requested operation would block.
ENOBUFS	The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.
ENOBUFS	The output queue for a network interface was full.This generally indicates that the interface has stopped sending, but may be caused by transient congestion.
EPARTITIONED	Indicates that the session has been partitioned
NOTE – The shaded error code is newly defined for ECTP.	

B.2.6 mrecv()

The `mrecv` function is used to receive the multicast data and indication signals for control purposes.

`ssize_t mrecv (int` *msockfd*`, void` **buf*`, size_t` *buflen*`, int` **flags*`, struct sockaddr` **fromaddr*`, socklen_t` **fromaddrlen*`);`

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *buf*: is a pointer to buffer to read into;
- *buflen*: is size of *buf*;
- *flags*: is not defined yet;
- *fromaddr*: is a pointer to a protocol-specific address to specify the sender;
- *fromaddrlen*: is the size of *fromaddr*.

When an application receives data from the buffer, it can identify the corresponding sender by using *fromaddr*.

The `mrecv` returns the number of bytes received if success. Otherwise it returns –1, if an error occurs or there is a control message that will be delivered to application. Error codes are listed in Table B.11.

Table B.11 – Error codes of `mrecv` function

Error code	Description
EBADF	The argument *msockfd* is an invalid descriptor.
ENOTCONN	The socket is associated with a connection-oriented protocol and has not been connected (see mconnect and maccept).
ENOTSOCK	The argument *msockfd* does not refer to a socket.
EAGAIN	The socket is marked non-blocking, and the receive operation would block, or a receive timeout had been set, and the timeout expired before data were received.
EINTR	The receive was interrupted by delivery of a signal before any data were available.
EFAULT	The receive buffer pointer(s) point outside the process's address space.
ETOTERM	TO has terminated the session.
ETOEXPEL	TO has expelled the LE or LO.
EPARTITIONED	Indicates that the session has been partitioned.
NOTE – The shaded error codes are newly defined for ECTP.	

B.2.7 mclose()

The `mclose` function is used to leave or terminate an ECTP connection by closing the socket. The default action of `mclose` with an ECTP socket is marking the socket as a closed and returning to the process immediately. The socket descriptor is no longer usable by the process.

```
int mclose (int msockfd);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function.

The `mclose` returns zero if success or –1 in the case of error as listed in Table B.12.

Table B.12 – Error codes of `mclose`

Error code	Description
EBADF	*msockfd* is not an active descriptor.
EINTR	An interrupt was received.

B.2.8 mgetsockopt() and msetsockopt()

The `mgetsockopt` is used to get the options and current connection characteristics that affect a socket.

```
int mgetsockopt(int msockfd, int level, int optname, void *optval, socklen_t *optlen);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *level*: specifies the code in the system to interpret the option: the general socket code, or some protocol-specific code (e.g., IPv4, IPv6, or ECTP);
- *optname*: specifies the type of option. Each level may define several option names;
- *optval*: a pointer to a variable into which the current value of the option is stored by `mgetsockopt`; and
- *optlen*: specifies the size of *optval* as a value-result for `mgetsockopt`.

The `msetsockopt` function is used to set the options that affect a socket.

```
int msetsockopt (int msockfd, int level, int optname, const void *optval, socklen_t *optlen);
```

Parameter Description:

- *msockfd*: is a socket descriptor that was returned by the `msocket` function;
- *level*: specifies the code in the system to interpret the option: the general socket code, or some protocol-specific code (e.g., IPv4, IPv6, or ECTP);
- *optname*: specifies the type of option. Each level may define several option names;
- *optval*: a pointer to a variable from which the new value of the option is fetched by `msetsockopt`; and
- *optlen*: specifies the size of *optval* as a value for `msetsockopt`.

Table B.13 summarizes the options that can be queried by `mgetsockopt` and `msetsockopt`.

Table B.13 – Socket options for `mgetsockopt` and `msetsockopt` with ECTP

Level	optname	`mget`	`mset`	Description	Flag	Datatype
`IPPROTO_ECTP`	`ECTP_QOS`	•	•	Get and set the QoS parameters		`QoS`
	`ECTP_OPPAR`		•	Get and set the control parent address		`OP_par`
	`ECTP_OPVAR1`	•	•	Get and set the operation values for ECTP part I		`OP_var1`
	`ECTP_OPVAR2`	•	•	Get and set the operation values for ECTP part II		`OP_var2`
	`ECTP_OPLJ`	•	•	Enables the late joining	•	`int`
	`ECTP_OPWCR`	•	•	Set the CR waiting time		`u_long`

The `mgetsockopt` and `mgetsockopt` return zero if success or –1 in the case of error as listed in Table B.14.

Table B.14 – Error codes for `mgetsockopt` and `msetsockopt`

Error code	Description
`EBADF`	The argument *msockfd* is not a valid descriptor.
`ENOTSOCK`	The argument *msockfd* is a file, not a socket.
`ENOPROTOOPT`	The option is unknown at the level indicated.
`EFAULT`	The address pointed to by *optval* is not in a valid part of the process address space. For `mgetsockopt`, this error may also be returned if *optlen* is not in a valid part of the process address space.

The following data structures are illustrated as information for use of `mgetsockopt` or `msetsockopt`.

```
typedef struct _op_var1 {
        int agn;    /* ACK Generation Number */
        int mcn;    /* Maximum Children Number */
        int mrn;    /* Maximum Retransmission Number */
        int mtl;    /* Maximum Tree Level */
        int nft;    /* Node Failure Threshold */
        int arn;    /* Active Receiver Number */
        int ccn;    /* Current Children Number */
        int crn;    /* Current Receiver Number */
        int ctl;    /* Current Tree Level */
        int ctr;    /* Current Transmission Rate */
        int cct;    /* Connection Creation Time */
}OP_var1;
```

```
typedef struct _op_var2{
        int mintr;  /* Minimum Transmission Rate */
        int ctr;    /* Current Transmission Rate */
        int maxtr;  /* Maximum Transmission Rate */
}OP_var2;
```

```
typedef struct _op_par{
        u_long   ctladdr;  /* Local Group Control Multicast Address */
        u_short  ctlport;  /* Local Group Control Port */
}OP_par;
```

```
typedef struct _QoS {
         u_shortthreshold;          /* threshold rate  */
         u_long   th_CHQ_TQA:2,th_CHQ_N:1,th_CHQ:29;    /* CHQ throughput */
         u_long   th_OT_TQA:2,th_OT_N:1,th_OT:29;       /* OT throughput  */
         u_long   th_LQA_TQA:2,th_LQA_N:1,th_LQA:29;    /* LQA throughput */
         u_long   td_OT_TQA:2,td_OT_N:1,td_OT:29;       /* OT transit delay */
         u_long   td_LQA_TQA:2,td_LQA_N:1,td_LA:29;     /* LQA transit delay */
         u_long   tdj_OT_TQA:2,tdj_OT_N:1,tdj_OT:29;    /* OT transit delay jitter */
         u_long   tdj_LQA_TQA:2,tdj_LQA_N:1,tdj_LQA:29;  /* LQA transit delay jitter */
         u_long   cr_TSDU_TQA:2,cr_TSDU_N:1,cr_TSDU:29;  /* corrupted data rate */
         u_long   l_TSDU_TQA:2, l_TSDU_N:1, l_TSDU:29;   /* lost data rate */
}QoS;
```

B.3 An example of the msocket.h header file

The following describes the 'msocket.h' header file that is used in an implementation of ECTP for information.

```
        #ifndef    __MSOCKET_H_
        #define    __MSOCKET_H_
        #include   <sys/types.h>
        #include   <sys/socket.h>
        #include   <netinet/in.h>
        #include   <arpa/inet.h>
        #include   <sys/un.h>
        #include   "ectpcb.h"
        #include   "var.h"
        #define    SIZE_MSETSOCKOPT          524
        struct _common{
                u_char command;
                u_short  len;
                u_long   error;
        };
```

```
enum{ MCONNECT = 101,
    MBIND,
    MSEND,
    MRECV,
    MACCEPT,
    MLISTEN,
    MCLOSE,
    MSETSOCKOPT,
    MGETSOCKOPT,
    DETACH};

/* msocket return values  */
#define    ECONNECT_OK          0x00
#define    ECONNECT_ROLE   0x01
#define    ESEND_OK       0x0000
#define    ESEND_RESEND    0x1004
#define    ESEND_NOSEND    0x1001
#define    ESEND_NOONE          0x1002
#define    EBIND_OK       0x00
#define    EBIND_ADDR           0x01
#define    EBIND_ROLE    0x02


#define    ECLOSE_OK     0x00
#define    ECLOSE_TO     0x01    // When TO invoked
#define    ECTP_QOS     0x4001
#define    ECTP_OPVAR1 0x4002
#define    ECTP_OPVAR2 0x4003
#define    ECTP_OPLJ    0x4004
#define    ECTP_OPWCR          0x4005
#define    ECTP_OPTREE         0x4006
#define    ECTP_OPTIME         0x4007
#define    ECTP_OPFLOW         0x4008
#define    ECTP_OPPAR0x4009

/* msocket message types */
#define    FIXED_SIZE      24      // sizeof fixed_header
struct _mbind_req{
          struct _common c;
          int         role;
          struct sockaddr_in ctrl;
          struct sockaddr_in local;
          struct sockaddr_in grp;

};
struct _mconnect_req{
```

```
        struct _common c;
};
struct _maccept_req{
        struct _common c;
        struct sockaddr_in to;
};
struct _msend_rep{
        struct _common c;
};
struct _mrecv_req{
        struct _common c;
};
struct _msend_req{
        struct _common c;
        u_char  rsvd[FIXED_SIZE - sizeof(struct _common)];
        u_char    data[MAXDATA];
};
struct _mrecv_rep{
        struct _common c;
        u_char  rsvd[FIXED_SIZE - sizeof(struct _common)];
        u_char    data[MAXDATA];
};
struct _mclose_req{
        struct _common c;
};
struct _mopt_op{
        int agn;
        int arn;
        int ccn;
        int ctl;
        int mcn;
        int mrn;
        int mtl;
        int nft;
        int lrtt;
};
struct _mopt_time{
   long agt;    // ACK Generation time
   long cct;    // Connection Creation time
   long hgt;    // HB Generation time
   long iat;    // Inactivity time
   long ndt;    // ND Generation time
   long rbt;    // Retransmission Backoff time
   long rxt;    // Retransmission time
   long tct;    // Tree Creation time
};
```

```
struct _mopt_flow{
        int cit;
        int cmn;
        int crn;
        int mintr;
        int ctr;
        int maxtr;
        int iri;
        int cri;
        int crd;
};
struct _mopt_tree{
   int tree_ct;    // connection type
   int tree_conf; // Tree Level Configuration
   int tree_mtl;  // Max Tree Level (only valid if tree_conf ==2)
   int tree_mcn;  // Max Children Number
   int bitmapsize;  // ack bitmap size
};

struct _mgetsockopt_req{
        struct _common c;
        int         optname;
        int         optlen;
        u_char    optval[MAXDATA/2];
};
struct _msetsockopt_req{
        struct _common c;
        int         optname;
        int         optlen;
        u_char    optval[MAXDATA/2];
};
char cbreq[128];
char dbreq[4096];
#endif
```

Bibliography

The following IETF RFCs are useful to understand or implement this Specification:

- IETF RFC 768, *User Datagram Protocol*, *Internet Standard*, August 1980.
- IETF RFC 791, *Internet Protocol*, *DARPA Internet Program, Protocol specification*, *Internet Standard*, September 1981.
- IETF RFC 793, *Transmission Control Protocol*, *DARPA Internet Program, Protocol specification*, *Internet Standard*, September 1981.
- IETF RFC 1112, *Host Extensions for IP Multicasting*, *Internet Standard*, August 1989.
- IETF RFC 1119, *Network Time Protocol*, *Internet Standard,* May 1990.
- IETF RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, *Best Current Practice*, March 1997.
- IETF RFC 2205, *Resource ReSerVation Protocol (RSVP)* – Version 1 Functional Specification, *Proposed Standard,* September 1997.
- IETF RFC 2210, *The Use of RSVP with IETF Integrated Services*, *Proposed Standard,* September 1997.
- IETF RFC 2236, *Internet Group Management Protocol, Version 2*, *Proposed Standard,* November 1997.
- IETF RFC 2460, *Internet Protocol, Version 6 (IPv6) Specification*, *Draft Standard,* December 1998.
- IETF RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, *Proposed Standard,* December 1998.
- IETF RFC 2597, *Assured Forwarding PHB Group*, *Proposed Standard,* June 1999.
- IETF RFC 2598, *An Expedited Forwarding PHB*, *Proposed Standard,* June 1999.
- IETF RFC 2750, *RSVP Extensions for Policy Control*, *Proposed Standard,* January 2000.
- IETF RFC 2836, *Per Hop Behavior Identification Codes*, *Proposed Standard,* May 2000.

SERIES OF ITU-T RECOMMENDATIONS

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

Geneva, 2003