M-Connection Service: A Multicast Service for Distributed Multimedia Applications

José F. de Rezende[1], Andreas Mauthe[2], David Hutchison[2] and Serge Fdida[1]
{rezende,fdida}@masi.bp.fr, {andreas,dh}@comp.lancs.ac.uk

[1] Laboratoire MASI - CNRS
Université Pierre et Marie Curie
4, Place Jussieu - 75252 Paris Cedex 05
France
[2] Computing Department
Lancaster University
Lancaster LA1 4YR
UK

Abstract. New high speed networks and the requirements of users for the support of interpersonal communication in a computerised heterogeneous environment demand for new communication services. These services have to provide support for multimedia group communication. In this paper we introduce a multicast transport service for multimedia group applications called M-Connection Service. It is a protocol independent service which provides an interface to different transport protocols. Apart from data transfer this service offers functions to establish, join, leave, change properties of and terminate multicast connections. Multicast connections are characterised by QoS and integrity conditions. We propose distinct QoS negotiation and group management schemes suitable for multicast multimedia communication. We describe the service elements of the M-Connection Service in detail. To demonstrate the feasibility of the service its service elements are mapped onto XTP 4.0 protocol mechanisms.

1 Introduction

Emerging high-speed networks provide enough bandwidth to support the exchange of both discrete and continuous media. This is met on the user side with an ever-increasing demand for interpersonal communication in a computerised heterogeneous environment. In order to support this communication, new services have to be provided. Only a few protocols and services exist which utilise the features of high-speed networks optimally *and* provide the required service to the user. In particular there is a lack of services supporting multimedia *group* communication.

Multimedia group applications place a multitude of requirements on such services [1, 2]. The services offered have to be flexible, efficient and dynamic, i.e. it must be possible to change characteristics, such as topology and QoS.

These stringent requirements make the design of supporting services complex and difficult.

Some proposals to support group communication at different levels of the system architecture have been introduced in [3–6]. However, these proposals only address particular aspects or problems of group communication. Many suffer from a high degree of complexity, are too general in some issues, whereas in others they are very specific.

To deal with the high demands and complexity of group communication a *comprehensive* approach towards the problem is necessary. To provide efficient and flexible services we propose a group communication service architecture built from independent service modules. In this architecture all tasks are clearly defined and performed once by just one service module where this can be done most efficiently. Thus, redundant service definitions are avoided. Policies and mechanisms are separated, and any user dependent policy decisions are taken close to the user whenever possible.

In this paper we introduce the M-Connection Service which is a fundamental module of this architecture. It provides an end-to-end communication support between one sender and multiple receivers. It is a transport service that interfaces multiple transport protocols which support multicast (1:N) and a certain degree of QoS. In the overall group communication architecture it offers the *communications* support needed by multimedia group applications. The M-Connection Service is an independent module that can be either used by other service modules or directly by the application.

The service offered by the M-Connection Service is independent of any specific protocol. However, two protocols have so far been considered as potential candidates for our service, viz. XTP 4.0 and a proprietary multimedia transport protocol developed at Lancaster University [7]. To show the feasibility of the M-Connection Service and how it can be provided we map it onto XTP calls and mechanisms. XTP was chosen because it provides the required features and an implementation is available [8].

This paper is organised in five sections. An overview on XTP 4.0 is given in section two. In section three we introduce the M-Connection Service. Section four describes the service elements of our service in detail and maps them onto XTP mechanisms. Finally we discuss open issues and summarise the paper in section five.

2 XTP 4.0 "Xpress Transport Protocol": Overview

XTP 4.0 is a transport protocol designed to be used by all kinds of applications on top of emerging high speed networks [11]. It allows different modes of communication, viz. connection-less, transaction and connection-oriented. The fundamental design principle of XTP is to provide mechanisms rather than to implement any particular policy. In the view of the protocol designers only the application has enough information to optimally configure the control procedures for the data exchange, therefore XTP is a set of mechanisms whose functionalities are mutually orthogonal. The control procedures can be turned on or off to tailor the protocol according to specific requirements. Bitflags in the XTP header are used for this purpose. There is no service interface specified for XTP. Thus, it is up to the implementor of the protocol to define his own service interface adapted to the specific needs of the user.

Certain features of XTP, for instance the provision of multicast and QoS support, make it suitable for use in multimedia group communication. We decided to exploit these features and to use XTP as basic transport protocol for the M-Connection Service. A detailed discussion of XTP is beyond the scope of this paper. In the following we discuss those aspects which are relevant to the M-Connection Service.

2.1 XTP Multicast

XTP multicast is an integral part of the protocol and not an addendum. It provides a simplex data flow from one transmitter to an arbitrary number of receivers. Before a multicast association can be established, a set of users who wish to communicate must request that their contexts are placed into the *listening* state. A multicast association is established when one or more listening multicast contexts receive and accept a FIRST packet from the multicast transmitter. A FIRST packet is accepted if address, traffic specification and options match the ones specified during the listen. In the options, the kind of control procedures employed are specified. The MULTI bit in the option field indicates that this association is a multicast association. After accepting a FIRST packet the context moves into the *active* state.

The receivers reply with a TCNTL packet to the FIRST packet if the SREQ bit was set. The TCNTL packet contains all necessary information about the receiver including its traffic specification. Only then does the multicast transmitter know the number and identity of receivers in the receiver group. Different user reliability semantics are proposed in the specification. These include k-reliability, a specific sub-set of members or a hybrid reliability semantic.

The traffic specification, i.e. QoS values, are negotiated during the establishment phase (FIRST packet) or during the course of an on-going association (TCNTL packet). This negotiation may be initiated by the user and can involve all users, XTP contexts and the network provider. Also, the implementor can construct XTP in such a way that the contexts can renegotiate without user intervention. This is for instance useful when conditions arise where the traffic specification is no longer satisfied.

Receivers can join an on-going association by sending a JOIN packet to the multicast group. If the join is accepted the multicast transmitter answers with a JOIN packet, else a DIAG packet is sent to the joining context. To leave an association the receiver sends a control packet with the END bit set to the multicast transmitter. The multicast transmitter can force a receiver to leave by sending it a CNTL packet with the END bit set.

Once all data from the multicast transmitter has been sent, the multicast data stream can be closed. A multicast association closure may be achieved with two degrees of gracefulness, viz. graceful close and abortive close. In a graceful close, the transmitter initiates the termination by setting the WCLOSE bit in an outgoing packet. The transmitter then collects responses from all active multicast receivers carrying the RCLOSE bit. After that it sends a packet with all three bits WCLOSE, RCLOSE, and END set to end the association. In an abortive close, the multicast transmitter sends the END bit at any time which closes the association immediately, regardless of the state of its multicast data stream.

XTP multicast is a collection of mechanisms that support group communication; it does not impose how these mechanisms should be used. In general, it is up to the user to determine how multicast mechanisms are used and how changes in the membership are accomplished. Therefore, the XTP service interface must provide its users with the ability to determine the policies to be used in a multicast association.

2.2 Group Management

Group management in XTP 4.0 is list-based. The receiver group list is created from the replies of the receivers. The multicast transmitter gathers control information during the communication from the receiver group to up-date this list. At regular intervals, a multicast transmitter must send DATA or CNTL packets with the SREQ or DREQ bit set. These intervals can be determined by various, user specified, policies. The transmitter resolves the information in its receiver group list to determine the association's current state according to the replies of the receivers. This is done after a packet with the SREQ or DREQ bit set is sent. If for some reason a CNTL response from any of the receivers does not arrive for some time, the association enters into a synchronising handshake during which the transmission of data is stopped.

The active group is made up of active receivers which are receivers whose control information is used by the multicast transmitter when it runs its control algorithms. List-based multicast offers reliable multicast by tracking the explicit membership of the receiver set designed for reliable delivery.

2.3 Error, Flow and Rate Control in XTP Multicast

The multicast transmitter sends all data packet to the multicast group address. In case of an error in the transmission the lost or corrupted data might be retransmitted to the multicast group. Either positive or negative acknowledgment is used to indicate errors. Retransmission may be either go-back-N or selective. The NOCHECK bitflag indicates if the checksum is calculated over the header fields only or over the whole packet. NOERR is a mode bit that enables or disables error control. When set, it informs the receiver that error correction processing shall be disabled because the sender will not retransmit data. In the multicast case, the multicast transmitter retransmits all data reported lost or corrupted in response to a SREQ bit.

To control data transmission XTP offers two independent mechanisms: flow and rate control. For flow control an end-to-end windowing flow control mechanism is employed. The NOFLOW bitflag indicates to the receiver that the transmitter will not be constrained by flow control. For multicast transmission flow control is done according to the slowest receiver in the active group. Rate control values are determined during the traffic specification negotiation.

3 M-Connection Service

The M-Connection Service is a basic module which supports multimedia multicast communication at the transport level. It offers a protocol independent service interface to transport protocols that provide multicast (1:N) connections and support QoS. This service is provided by a restricted set of service elements which support all functionality needed for communication between one sender and multiple receivers. The service offered by the M-Connection Service allows service users to establish, join, leave, change properties of and terminate *m-connections*. M-connections are multicast transport connections established for the purpose of transmitting multimedia data. Unicast connections are considered as a special case of multicast connections. M-connections are characterised by certain properties, such as QoS and policies to determine the success of the communication, which can be dynamically changed by the service user. All service primitives that comprise the M-Connection Service are shown in table 1.

The aim of the M-Connection Service is twofold, i) to provide the user with a set of clearly specified service elements for multicast communication and ii) to hide the details and specific characteristics of different transport protocols.

3.1 Service Classes and QoS Schemes

The M-Connection Service is designed to support continuous as well as discrete media data transfer. Three distinct service classes to accommodate the requirements of different kinds of traffic are provided:

A - best-effort
B - group integrity and/or QoS
C - group integrity and data transfer reliability

Class A provides a best-effort service without QoS negotiation. However, the sender can specify the QoS of the data stream only to inform the receivers about the traffic type and its QoS. This service class will be used by dissemination services like radio or television broadcast.

The second class, Class B, offers group integrity and the possibility to negotiate QoS. Group integrity in this context refers to the possibility to specify integrity conditions for the m-connection. QoS can be negotiated among all participants; three different negotiation schemes are considered:

service element	*primitives*
M-Connection set-up	listen_mconn, open_mconn.req
	open_mconn.ind, open_mconn.rsp
	open_mconn.cnf, open_mconn.ack
Leave M-Connection	leave_mconn.req, leave_mconn.ind
	leave_mconn.cnf
Join M-Connection	join_mconn.req, join_mconn.ind
	join_mconn.rsp, join_mconn.cnf
Change M-Connection Properties	chg_mconn_qos.req, chg_mconn_qos.ind
	chg_mconn_qos.rsp, chg_mconn_qos.cnf
	chg_mconn_qos.ack
	chg_mconn_policy.req, chg_mconn_policy.cnf
Terminate M-Connection	close_mconn.req, close_mconn.ind

Table 1. M-Connection Service primitives.

- receiver-selected: The sender specifies the maximum QoS it can provide. Each receiver selects the most appropriate QoS which can be supported through mechanisms like filtering [9].
- imposed-QoS: The sender specifies a distinctive QoS which can not be negotiated. Individual receivers can only accept or reject it.
- common-denominator: This has to be agreed on a common QoS among all participants.

The common-denominator case is distinguished from the other cases in two ways. First, the sender might give a range of QoS rather than a clearly specified value. Receivers can propose their QoS within this range. Second, a third message is required to inform the receivers of the common QoS. The QoS parameters we consider are throughput, end-to-end delay, delay jitter, and error rate. A flow specification like that proposed in [10] can be used to fully describe the QoS requirements in such a case.

If QoS negotiation without group integrity is required the parameters determining group integrity are set to nil. If only group integrity is required, no QoS will be specified. Service class B is intended to be used for continuous data transfer and for discrete data transfer where no data transfer reliability is required.

Class C provides group integrity as well as data transfer reliability. Data transfer reliability is provided through retransmission of any lost or corrupted data. The specified QoS in this class gives only an indication of bandwidth and timing requirements for the m-connection. Any reservation of network or end-system resources are done according to this QoS specification[1]. No guarantees for the timely transmission of correct data units are given. Data reliability in this class has always priority over timing constraints.

3.2 Group Communication Policies

The M-Connection Service is characterised by a minimal set of policies which are related to the way a multicast communication is considered successful. Compared to the unicast case it is much more difficult in multicast communication to determine the success or failure of establishment and data transfer. Certain conditions have to be defined which specify the number and/or identity of participants necessary to consider the establishment and/or data transmission successful. We call these conditions establishment and communication integrity conditions (M-AGI, Active Group Integrity) respectively. These conditions might be different for each m-connection.

We distinguish between establishment and communication integrity conditions for two reasons. First, establishment conditions are specified by the initiator of a group communication and will never be negotiated. Communication can only commence if these conditions have been fulfilled. In contrast, communication integrity conditions can be changed and it is left to the service user when and under what conditions this is done. Second, different conditions might be specified for the establishment and the data transfer. A connection set-up, for example, is usually quite costly and therefore it might only be worth proceeding if a certain number of users agree to participate. During the communication, on the other hand, it might be sufficient if only one participant is left who bears the cost of transmission.

The following conditions are considered:

- *key-member* (identity) and/or
- k (number)
- *all*
- *quorum*[2].

We are currently considering an upper-bound on the maximum number of receivers permitted. This might be necessary for several reasons; for instance the available resources that can be allocated to a specific m-connection might be restricted.

[1] Note that error rate in this class will always be zero since the data transfer is reliable.

[2] For establishment condition *quorum* and *all* refers to a known group, i.e. the number and/or identity of group members is known by the service user or the M-Connection Service. For the communication integrity conditions *quorum* and *all* refer to the number of positive replies during connection establishment.

4 M-Connection Service Elements

The M-Connection Service provides five service elements to set-up, manage and terminate a multicast connection. For each service element the primitives and their parameters are defined in this section . Their function is explained and they are mapped onto corresponding XTP mechanisms whenever appropriate.

4.1 M-Connection Establishment

M-Connections are established between a service user who acts as sender and one or more receivers. The initiating user is called *master*.

The service primitives and parameters used during the establishment phase are shown in table 2 and 3, respectively.

primitives	*parameters*
listen_mconn	(@mgroup, service)
open_mconn.req	(@master, @mgroup, QoS, mconn_policy);
open_mconn.ind	(@master, @mgroup, QoS);
open_mconn.rsp	(@user, @master, mconn_id(, QoS));
open_mconn.cnf	(mconn_id, QoS, status);
open_mconn.ack	(mconn_id, QoS);

Table 2. M-Connection Establishment primitives.

parameters	*meaning*
@master	address of the master
@mgroup	address of the receiver group
@user	address of a receiver
QoS	quality of service specification
mconn_policy	policies applied to the m-connection
mconn_id	identifier of the m-connection
status	returns success or failure and a list of the receiver group
service	service class (A, B or C)

Table 3. M-Connection Establishment parameters.

Before an m-connection can be established, all users who wish to participate must issue a *listen_mconn* primitive to be placed in listening state. The parameters carried by this primitive inform the M-Connection Service to which address the user wishes to listen and the type of service expected. This primitive is directly mapped onto an XTP listen. A XTP listening context is established where the address and service value are specified. The XTP traffic specification is left empty.

To establish an m-connection a potential sender issues an *open_mconn.req* primitive. This is mapped onto a FIRST packet which is sent to the multicast group address. The FIRST packet is encoded according to the parameters specified in the *open_mconn.req* primitive. The option bits in the FIRST packet are set according to the profiles shown in the table 4. For each service class a specific profile exists. In order to gather the information from the receiver the SREQ bit must be set. In the classes B and C the SREQ bit is set when integrity conditions and/or QoS have been specified.

Class	*NOCHECK*	*NOERR*	*MULTI*	*NOFLOW*	*SREQ*	*RCLOSE*
A	ON	ON	ON	ON	OFF	ON
B	(i)	ON	ON	(ii)	ON	ON
C	OFF	OFF	ON	OFF	ON	ON

(i) - according to the QoS parameters.
(ii) - according to the kind of traffic.
Table 4. FIRST packet bitflags.

When receiving a FIRST packet an *open_mconn.ind* is issued to each user listening on the specified *mgroup* address. This primitive conveys the QoS specified by the master and the policy to be used for QoS negotiation. Receivers respond either positively or negatively by issuing an *open_mconn.rsp* primitive. In case the SREQ bit has been set in the FIRST packet, a positive response provokes the emission of a TCNTL packet. This is encoded according to the parameters specified in the *open_mconn.rsp* primitive. In case of a negative response, a CNTL packet with the END bit set is issued. If no SREQ bit was set no packet is sent back to the master.

At the master side all replies are gathered and a data structure is built that identifies the receivers and captures their current state. Establishment conditions are validated using these replies. If the establishment is considered successful an *open_mconn.cnf* is issued to the master including the list of the receiver group. If for QoS negotiation the common-denominator scheme is used a TCNTL packet specifying the final QoS is sent to the receivers. The reception of this packet causes the emission of a positive or negative *open_mconn.ack* primitive at the receiver side. If the agreed QoS is equal or “weaker” than the QoS selected by the user a positive *open_mconn.ack* primitive is issued. Otherwise, a negative *open_mconn.ack* primitive is used. In this case, the user will not take part on the m-connection.

A *close_mconn.ind* primitive is issued to the master user if the establishment conditions are not met. Further, a CNTL packet with the END bit set is sent to the receiver group. At the receiver side this packet generates a *close_mconn.ind* primitive.

The complete time sequence diagram for a successful m-connection establishment with the corresponding mapping onto the XTP protocol is shown in the figure 1. Figure 2 depicts a non-successful m-connection establishment time sequence diagram.

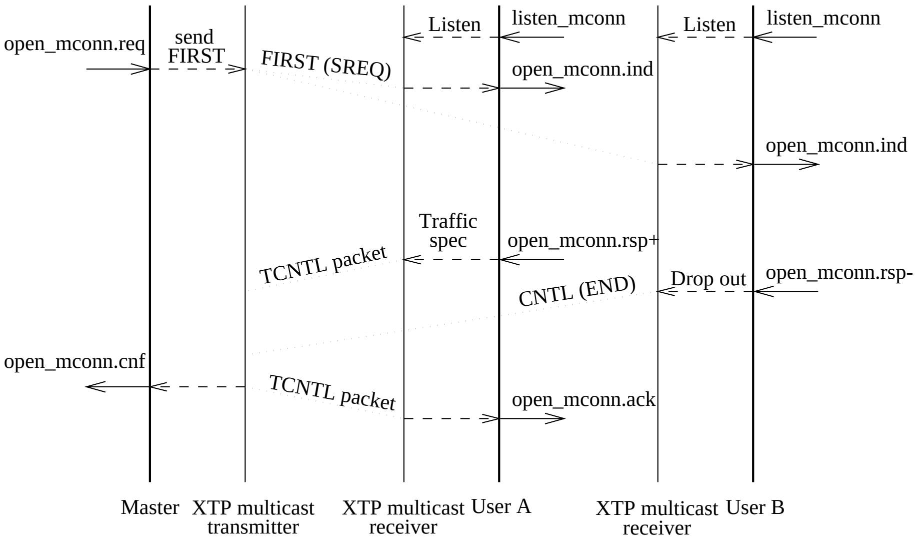

Fig. 1. Successful M-Connection Establishment time sequence diagram.

Group Management. During the establishment phase integrity conditions are specified according to which the establishment and data transfer are deemed successful. The M-Connection Service validates these conditions. In XTP 4.0 the already discussed group management scheme is proposed to accomplish this task. Semantics for what is called *group reliability* are also proposed, but we found the way these semantics are supported are too restrictive and not suitable for our purpose. In particular the proposed synchronising handshake after one or more receivers fail to respond to an SREQ bit does not suit our requirements. During a synchronising handshake all data transmission is stopped. In such a case there will be unacceptable gaps in the media play-out. Lost or corrupted continuous-media data can generally not be retransmitted in time. Therefore,

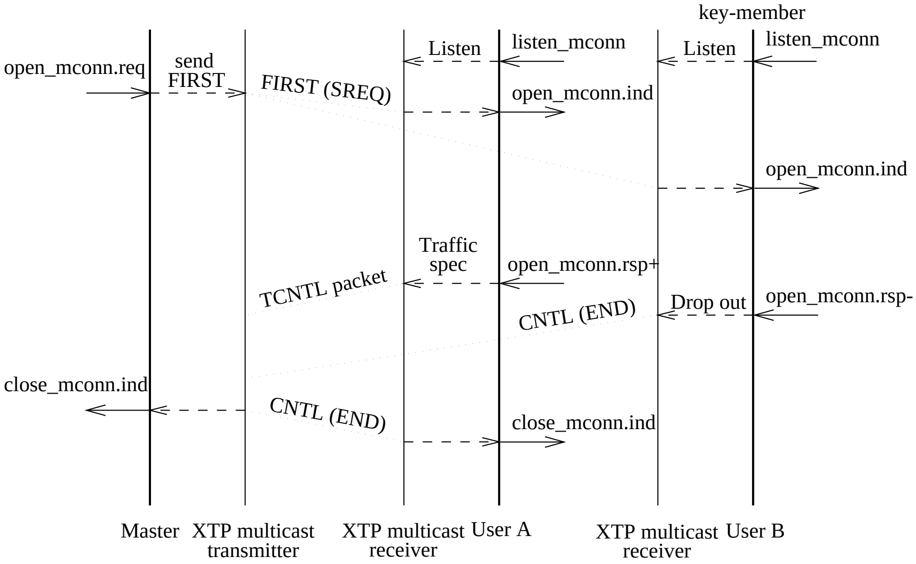

Fig. 2. Non-Successful M-Connection Establishment time sequence diagram.

data reliability is not required. Only group reliability or integrity conditions are validated *a posteriori*. The following group management scheme accommodates our requirements better.

Our proposed group management scheme is also list-based. A list of receivers is constructed from the replies to the FIRST packet. This list contains both static and dynamic receiver information. The former contains receiver address and return key. The latter, i.e. dynamic part of the list contains the following fields: the latest *echo* value (return to an SREQ), *rseq* (sequence number of the next in-sequence byte expected) and *alloc* (credit left at the receiver).

Packets (CNTL or DATA packets) with the SREQ bit set are sent periodically (SREQ interval). Every reply from a receiver up-dates the dynamic receiver information. After a clearly specified number of SREQ intervals the list is checked and the integrity conditions are evaluated. If a key member has not replied for some time and/or the necessary number of receivers are not present, the m-connection will be terminated.

Group management policies determine how multicast mechanisms are used and how changes in the membership are accomplished. The M-Connection Service provides its users with the ability to specify these policies. For each class different group management policies are used. The following policy areas specified in the mconn_policy parameter are relevant to our service:

- establishment condition - all, k, quorum or/and key-member.
- communication integrity (M-AGI) - all, k, quorum or/and key-member.

– ejection[3] - gives a policy according to which a receiver is forced to leave the m-connection because it breaches certain conditions (e.g. QoS).

4.2 Leave M-Connection

The leave operation is used to drop a corresponding user from an m-connection.

The service primitives and parameters used in this operation are shown in table 5 and 6, respectively.

primitives	*parameters*
leave_mconn.req	(@user, mconn_id, reason);
leave_mconn.ind	(@user, mconn_id, reason);
leave_mconn.cnf	(@user, mconn_id, status, reason);

Table 5. Leave M-Connection primitives.

parameters	*meaning*
@user	address of the receiver
mconn_id	identifier of the m-connection
reason	reason to leave
status	returns success or failure

Table 6. Leave M-Connection parameters.

This operation may be initiated by three different entities:

1. master user
2. receiver user
3. master provider

In the first case the master user initiates a leave operation by issuing a *leave_mconn.req* primitive in order to drop a specific receiver user from a particular m-connection. The service provider checks the integrity conditions before the operation is executed. It refuses to execute the leave if this would breach the M-AGI. Then, a negative confirmation, *leave_mconn.cnf*, is sent to the master user. This policy was adopted to prevent a master user from releasing an m-connection accidently. In highly dynamic groups this can happen when the master user is not aware of the current state of the receiver group. It should be

[3] This case might not apply to continuous media where receivers who can not keep up with the pace of the communication do not disturb other receivers.

ultimately a user decision if an m-connection is released or if instead integrity conditions are changed.

If the requested leave does not breach the integrity conditions, this primitive provokes the emission of a CNTL packet with the END bit set to the unicast address of the respective receiver. At the receiver side this packet causes a *leave_mconn.ind* to be issued to the receiver user. The master user receives a positive *leave_mconn.cnf* primitive indicating that the user has been dropped. Figure 3 shows the complete time sequence diagram for the leave initiated by the master user. The corresponding mapping onto the XTP mechanisms is also depicted in this figure.

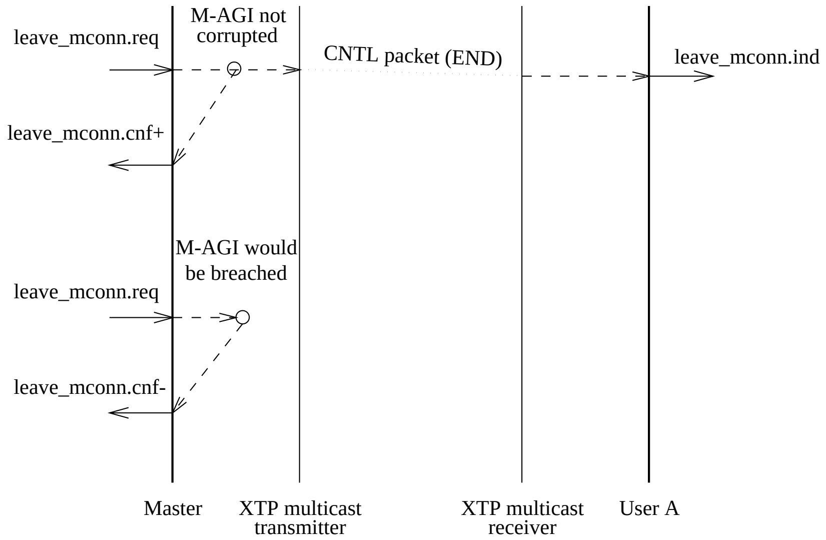

Fig. 3. Leave M-Connection initiated by the master user time sequence diagram.

In the receiver initiated case the receiver user issues a *leave_mconn.req* to quit the m-connection. This primitive is mapped onto a CNTL packet with the END bit set which is sent to the unicast address of the multicast transmitter. After receiving this packet the master verifies the M-AGI conditions. When the M-AGI is not corrupted the master user receives a *leave_mconn.ind* primitive that indicates that the corresponding receiver has left the m-connection. Otherwise the m-connection must be terminated. Figure 4 shows the time sequence diagram for a leave operation initiated by the receiver user.

During the establishment phase, the master user has to specify under which conditions a receiver must be dropped from the m-connection. For example, if the receiver falls too far behind in a reliable multicast data transfer this might jeopardise the success of the data transmission for other receivers. In this case this user should be ejected from the m-connection. Hence, the third case of the leave operation is related to the group management policy. The ejection policy

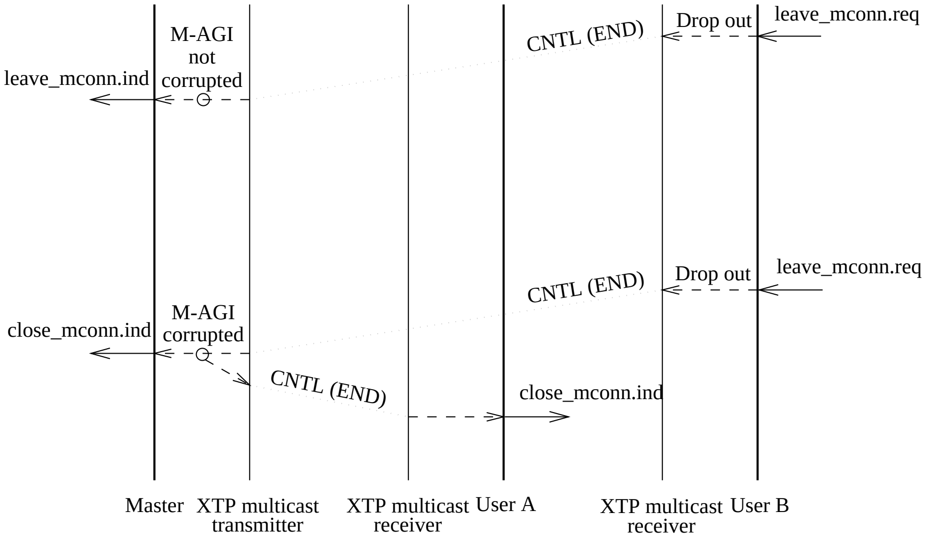

Fig. 4. Leave M-Connection initiated by the corresponding user time sequence diagram.

must determine if the service provider has the right to drop certain endpoints even if the M-AGI is breached. This would result in the termination of the m-connection. As above, two cases must be considered. In case the M-AGI is not corrupted, the master provider sends a *leave_mconn.ind* primitive to the master user. This primitive informs the master user that a receiver has been removed and gives the identity of this receiver including the reason. In parallel, it issues a CNTL packet with the END bit set to the receiver that has to be dropped. At the receiver side this packet causes the emission of a *leave_mconn.ind* primitive. If the M-AGI is corrupted the master provider terminates the m-connection. The first case is depicted in figure 5.

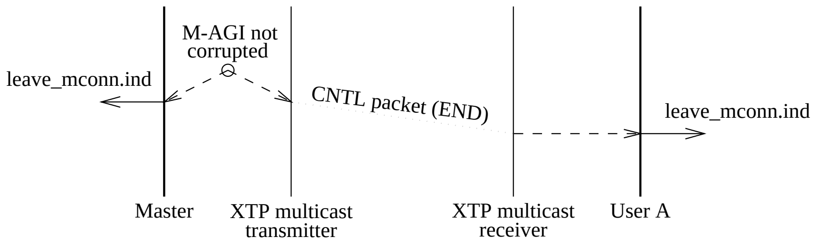

Fig. 5. Leave M-Connection initiated by the master provider time sequence diagram.

4.3 Join M-Connection

The join operation is used to add a new receiver to an m-connection. It is an unconditional service element of type confirmed. The primitives and parameters used to join an m-connection are described in table 7 and 8.

primitives	*parameters*
join_mconn.req	(@user, mconn_id, QoS);
join_mconn.ind	(@user, mconn_id, QoS);
join_mconn.rsp	(@master, @user, mconn_id, QoS, status);
join_mconn.cnf	(@master, @user, mconn_id, QoS, status);

Table 7. Join M-Connection primitives.

parameters	*meaning*
@user	address of the joining user
@master	address of the master
mconn_id	identifier of the m-connection
QoS	QoS required by the joining user
status	returns success or failure

Table 8. Join M-Connection parameters.

The join is performed in a two-way handshake. The joining user issues a *join_mconn.req* in which it gives its address, the id of the m-connection it wants to join and the required QoS. The *join_mconn.req* is mapped onto an XTP JOIN packet. An incoming JOIN packet is passed as a *join_mconn.ind* to the master user. In the *join_mconn.rsp* the master gives his address, the QoS (in case the policy requires a common QoS for all) and the status, i.e. if the join was successful or not. The response is mapped onto a JOIN packet in XTP. If the join was unsuccessful a DIAG packet is sent. This might be for instance the case when the receiver and/or network can not handle the required QoS. At the receiver side the JOIN packet is indicated as a positive *join_mconn.cnf*. A DIAG packet will be passed as a negative *join_mconn.cnf* to the service user. The time sequence diagram for the join operation is shown in figure 6.

4.4 Changing M-Connection Properties

Policies (i.e. integrity and ejection conditions) and QoS of an m-connection can be dynamically changed. The M-Connection Service Interface offers two generic

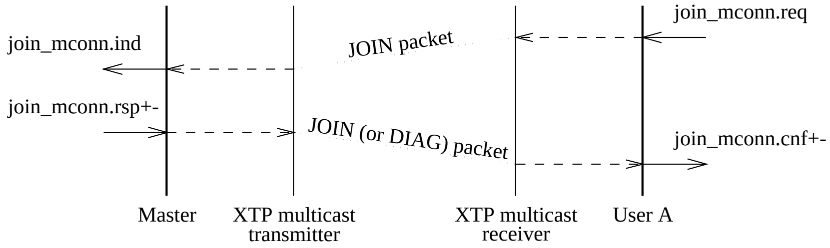

Fig. 6. Join M-Connection time sequence diagram.

sets of service primitives to change QoS and policies. Integrity conditions can only be changed by the master. The same is true for QoS which is valid for the whole m-connection. A receiver may request to change its QoS in the receiver-selected case. Further, the service provider, i.e. the network might indicate a change in QoS, which would result in an indication to the m-connection service user.

Table 9 shows the primitives provided to change m-connection properties and table 10 describes the parameters of this operation.

primitives	*parameters*
chg_mconn_qos.req	(@user, mconn_id, QoS);
chg_mconn_qos.ind	(@user, mconn_id, QoS);
chg_mconn_qos.rsp	(@user, mconn_id, QoS);
chg_mconn_qos.cnf	(@user, mconn_id, QoS);
chg_mconn_qos.ack	(mconn_id, QoS);
chg_mconn_policy.req	(mconn_id, mconn_policy);
chg_mconn_policy.cnf	(mconn_id, status);

Table 9. Changing M-Connection Properties primitives.

In the case of receiver-selected QoS scheme a receiver can change its QoS by issuing a *chg_mconn_qos.req*. This is mapped onto an XTP TCNTL packet with the SREQ bit set which is addressed to the multicast transmitter. New resource reservations will be made according to the QoS specification and filters will be re-instantiated if necessary. The multicast transmitter will answer this TCNTL packet with a TCNTL packet addressed to the requesting user stating the agreed QoS. At the receiver this TCNTL packet is indicated as a *chg_mconn_qos.cnf*. The change is only successful if the specified QoS matches the one stated in the *chg_mconn_qos.req*.

parameters	*meaning*
@user	requesting user address
mconn_id	identifier of the m-connection
QoS	new QoS
mconn_policy	new policy
status	returns success or failure

Table 10. Changing M-Connection Properties parameters.

In the two other cases (i.e. common-denominator and imposed-QoS) only the master can issue a *chg_mconn_qos.req* primitive. This will be mapped onto a TCNTL packet with the SREQ bit set sent to the receiver group. At the receiver side a *chg_mconn_qos.ind* is passed to the service user containing the QoS that can be provided by the network. The user responds with a *chg_mconn_qos.rsp*. Again, this is mapped onto a TCNTL packet sent to the multicast transmitter. A *chg_mconn_qos.cnf* is issued to the master user. In the common denominator case a last TCNTL packet is sent to the the multicast group stating the new common value. This is passed as a *chg_mconn_qos.ack* to the users.

The time sequence diagrams for changing the QoS initiated by the master and by the user are shown in the figures 7 and 8, respectively.

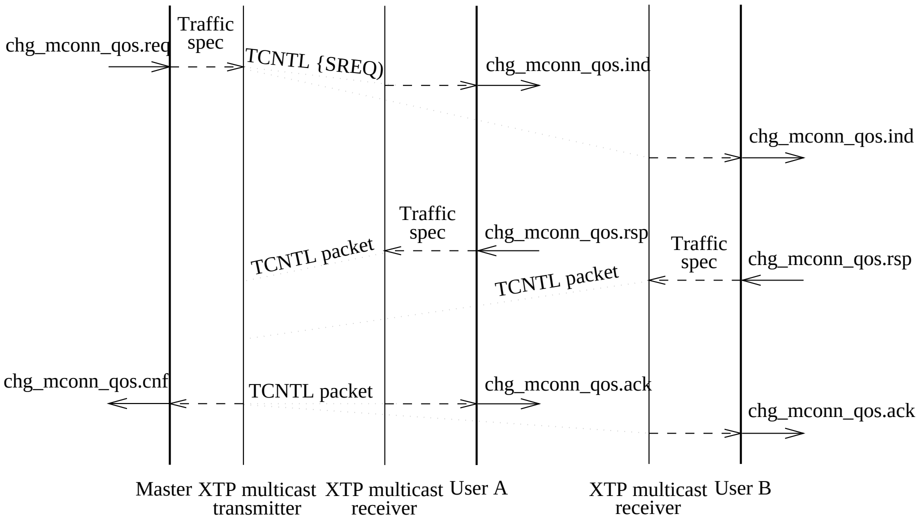

Fig. 7. Master initiated change of M-Connection QoS.

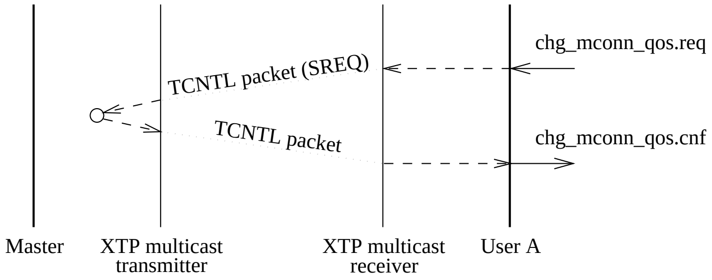

Fig. 8. Receiver initiated change of M-Connection QoS.

To change the policy associated with an m-connection, a *chg_mconn_policy.req* primitive is issued by the master user including the new policy to be applied. Following this, the master user receives a confirm from the service provider that indicates whether the operation was successful or not. This does not involve any interaction with the receivers and/or the XTP protocol. Therefore nothing has to be sent over XTP.

4.5 M-Connection Termination

Using this operation both the master user or master provider may terminate an m-connection. It is an unconfirmed service element. The service primitives and their parameters used during the termination are shown in table 11 and 12.

primitives	*parameters*
close_mconn.req	(mconn_id, mode, reason);
close_mconn.ind	(mconn_id, mode, reason);

Table 11. M-Connection Termination Primitives.

parameters	*meaning*
mconn_id	identifier of the m-connection
mode	mode of the closure (graceful or abrupt)
reason	reason to terminate the m-connection

Table 12. M-Connection Termination Parameters.

The M-Connection Service provides two ways to terminate an m-connection, *m-release* and *m-abort*. With the former it is ensured that the m-connection is only terminated when all data was correctly received according to the specified integrity conditions. With the latter an m-connection is aborted immediately. This can be easily mapped onto XTP abbreviated graceful close and abortive close respectively.

To terminate an m-connection an *close_mconn.req* primitive is issued by the master user indicating whether it is an m-release or an m-abort operation and the identifier of the m-connection. In case of an m-release, this primitive causes the XTP protocol to enter into a standard graceful close procedure for multicast as described in section 2. In the case of an m-abort a packet with the END bit set is sent to the multicast group, the multicast association is terminated immediately. After receiving a packet with the END bit set a *close_mconn.ind* is issued, the m-connection does not exist any longer. Figure 9 depicts the time sequence diagram for the m-release or m-abort operation.

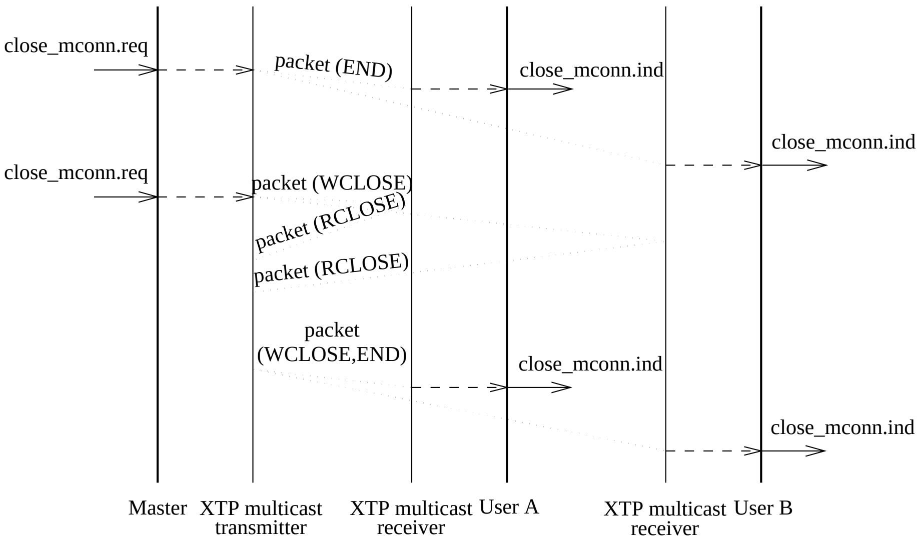

Fig. 9. M-Connection Termination time sequence diagram.

An m-connection can also be terminated through an m-abort initiated by the service provider. This is for instance the case when M-AGI conditions are breached. The master service provider issues a *close_mconn.ind* to the master user and sends a packet with the END bit set to the receiver group. At the receiver side, a *close_mconn.ind* primitive is sent indicating the reason why the m-connection has been aborted.

The m-release will be only used with service class C where data transfer reliability is provided. In any other case an m-connection will be terminated using m-abort.

5 Conclusion

The M-Connection Service is part of a service architecture to support multimedia group communication. It is a protocol independent service that provides an interface to different transport protocols. A set of service elements to support multicast communication is defined. Using this service interface, a user can establish, join, leave, change properties of and terminate multicast connections. These connections are characterised by QoS and integrity conditions. The service specification includes the definition of QoS negotiation and group management schemes. Also, different group management policies are defined which can be specified by the user through the service interface.

The aim of this service is twofold, to provide the user with a set of clearly specified service elements for multicast communication and to hide the details and specific characteristics of different transport protocols.

To show how the M-Connection Service can be provided with an existing transport protocol each service element is mapped onto XTP mechanisms. The corresponding XTP functions are discussed in detail for each service element.

XTP does not dictate in every detail how to use the offered protocol mechanisms. On the contrary, every mechanism and function can be adapted to specific user needs. It gives a high degree of freedom how to use its mechanisms. A drawback of this approach is that descriptions of mechanisms are very often unclear and ambiguous. For instance the way a response to a SREQ bit is generated is considered as an implementation aspect. Sometimes user intervention is required, for example during the establishment of an association. The protocol does not clearly specify if the reply to a SREQ bit is generated by the user or the receiver context. However, the text in the protocol specification indicates that the response is generated by the receiver context without user intervention.

The group management scheme proposed by XTP was apparently designed for reliable multicast. This can be seen in the way a failure to respond to an SREQ bit is treated. In this case the multicast transmitter initiates a synchronising handshake during which the entire data transmission is stopped. This is not suitable for multimedia applications where continuous, uninterrupted data streams are required. Moreover, this actually breaches the design philosophy of XTP not to impose any policy. To overcome this problem we specify our own group management scheme using similar mechanisms.

In the M-Connection Service specification an m-connection is always terminated when the integrity conditions are breached (hard AGI). It is still an open issue if it should be possible to suspend an m-connection in some cases (soft AGI). This is for instance proposed in [12]. However, in the case of soft AGI a problem arises with XTP since it does not provide a way to suspend a multicast association and resume it later. It is left for further research if the support of soft AGI conditions is required and how this could be accommodated by the lower layers.

Another open issue is if join invitation is needed at this level. It is not needed if the availability of the potential user and his willingness to participate is checked before the actual join. On other hand, it is required if a user is called to participate in a telephone like model. In this case, it has to be clear if the service provider has sufficient resources before calling the user. This problem will be also addressed in our future research.

References

1. A. Mauthe, D. Hutchison, G. Coulson and S. Namuye: "From Requirements to Services: Group Communication Support for Distributed Multimedia Systems", in *Proc. 2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures (IWACA94)*, Heidelberg, Germany, Sept. 1994.
2. C. Szyperski and G. Ventre: "Efficient Support for Multiparty Communication", in *Proc. of Multimedia Transport and Teleservices, International COST237 Workshop*, Vienna, Austria, Nov. 1994.
3. L. Henckel: "Multipeer Transport Services for Multimedia applications", in *Proc. of HPN'94, 5th IFIP Conference on High Performance Networking*, Grenoble, France, pp. 165–183, June 1994.
4. W. J. Clark and J. Boucher: "Multipoint Communications - The Key to Groupworking", *BT Technology Journal*, vol. 12, no. 3, pp. 72–80, July 1994.
5. G. J. Heijenk, X. Hou and I. Niemegeers: "Communication Systems Supporting Multimedia Multi-user Applications", *IEEE Network Magazine*, vol. 8, no. 1, pp. 34–44, Jan. 1994.
6. M. Altenhofen, et.al: "The Berkom Multimedia Collaboration Service", in *Proc. First ACM International Conference on Multimedia*, Anaheim, CA, May 1993.
7. F. Garcia: "Continuous Media Transport and Orchestration Services", PhD Thesis, Lancaster University - Computing Department - Lancaster, UK, May 1993.
8. W. T. Strayer, S. Grayer and J. Raymond E. Cline: "An Object-Oriented Implementation of the Xpress Transfer Protocol", in *Proc. 2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures (IWACA94)*, Heidelberg, Germany, Sept. 1994.
9. N. Yeadon, F. Garcia, A. Campbell and D. Hutchison: "QoS Adaptation and Flow Filtering in ATM Networks", in *Proc. 2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures (IWACA94)*, Heidelberg, Germany, Sept. 1994.
10. F. Garcia, A. Mauthe, N. Yeadon and D. Hutchison: "QoS Support for Video and Audio Multipeer Communications", in *Expert Contribution to SC6 WG4 and SC21 - ISO-Meeting*, Beppu, Japan, 1995.
11. XTP Revision 4.0: "Xpress Transport Protocol Specification", XTP Forum, Santa Barbara, CA, Mar. 1995.
12. P. Cocquet: "4th Draft on Multipeer Taxonomy", ISO/IEC-JTC1/SC6/N, Mar. 1995.