Reliable Multicasting in the Xpress Transport Protocol

J. William Atwood
Concordia University
Montreal, Quebec
Canada

Octavian Catrina
Polytehnica University
Bucharest
Romania

John Fenton
Mentat, Inc.
Los Angeles, CA
U.S.A.

W. Timothy Strayer
Sandia National Laboratories
Livermore, CA
U.S.A.

Abstract

The Xpress Transport Protocol (XTP) is designed to meet the needs of distributed, real-time, and multimedia systems. This paper describes the genesis of recent improvements to XTP that provide mechanisms for reliable management of multicast groups, and gives details of the mechanisms used.

Introduction

The Xpress Transport Protocol (XTP)[1,2,3] is a high-performance transport protocol designed to meet the needs of distributed, real-time, and multimedia systems in both unicast and multicast environments. The XTP protocol definition is maintained by the XTP Forum; the current definition is Revision 4.0 published in March, 1995. There is an Addendum to the XTP 4.0 Specification that refines and replaces some of the XTP 4.0 multicast mechanisms. This paper describes the genesis of these changes and how they work.

XTP is a very flexible protocol, in that it consists of a number of independent features: flow control, error control, delivery priority, unicasting/multicasting, various data delivery semantics, parametric addressing, and traffic specification. The emphasis in the design of XTP has always been on the provision of basic mechanisms, from which the user can construct a service well-suited to the application at hand.

Multicasting has always been a part of the XTP specification. Revision 3.6[1] of the specification required the receivers to multicast their control responses to the entire group. An appendix to the specification described the *bucket algorithm* for managing responses from the receivers, and introduced the ideas of *damping* and *slotting*, with a goal of minimizing the control traffic from the receivers. Certain aspects of the bucket algorithm were subsequently shown to have undesirable features[4]. Multicasting of the responses is problematic in widely-distributed networks[5], and the suppression of responses from the receivers prevents the achievement of fully-reliable data transfer.

In revision 4.0[3], the procedures were substantially revised. The recommendation to suppress response packets was removed, and control packets are unicast (sent only to the transmitter) by the receivers. Mechanisms were introduced to uniquely identify the participants in an association. In addition, a comprehensive mechanism for the negotiation and re-negotiation of traffic specifications was introduced. However, the mechanisms for uniquely identifying receivers involve potentially inefficient mappings, which could substantially reduce the performance of a system. Further, it is not possible to learn the transport-level addresses of the receivers in certain cases. Finally, there is no convenient mechanism for determining the initial size of the window at the receivers.

In this paper, we outline a solution to the above deficiencies.

Multicast Concepts

Multicast communication involves one transmitter (the initiator), and one or more receivers (1:N multicasting). (There are several ways to construct M:N multicasting using the 1:N multicast facilities.) The existence of the association is governed by rules established by the transmitter. These rules can vary from "I do not care" to "I must know the exact membership". In the first case, all receivers will be *passive* receivers, whose control information (if ever issued) will be ignored by the transmitter. In the second case, all receivers will be *active* receivers, whose control information will be used by the transmitter in managing the group and assessing the progress of the data transmission. Clearly, a variety of policies can be envisaged that will allocate the receivers between the active and passive groups in different ways. The *active group integrity* is concerned with the maintenance of the membership of the group of active receivers: the set of rules that decide when the initial group has formed, govern admission and withdrawal of active receivers, and decide when the group must be dissolved.

DISCLAIMER

Reliable Multicast Protocols

Many protocols have been developed to support data exchange among various communications participants. Some of these protocols, such as MTP (Multicast Transport Protocol)[6,7], and RMP (Reliable Multicast Protocol[8], implement a centralized control and error correction scheme providing a totally-ordered multicast delivery. The central instance controlling data transfer may become a bottleneck when dealing with numerous receivers. An approach based on the establishment of multicast servers is given by Carle[9]. The approach used in XTP Revision 3.6, of damping "redundant" responses, furthers the scalability of the protocol, but prevents achievement of full reliability. SRM (Scaleable Reliable Multicast)[10] integrates a mechanism to implement receiver-based error control, and uses timers carefully set to avoid a flood of retransmission requests. However, the correct setting of the timers will be very difficult for highly dynamic networks with quickly changing load and frequently changing network structure. The Local Group Concept of Hofmann[11] presents a scaleable multicasting facility well adapted to a variety of environments.

All of the referenced protocols solve a particular problem in reliable multicasting. Some of them provide a total ordering of the delivered data from multiple senders. We do not attempt this; our goal is to present a protocol that permits constructing the service required by a wide range of applications, from totally unreliable, to fully reliable, without requiring the implementation of any particular policy for managing the active group.

Association Management in XTP

In this section, we will briefly introduce the features of XTP salient to the discussion of XTP's reliable multicast service. The current complete definition of the protocol is given in the XTP Specification[3]. XTP is a connection-oriented protocol, where connections are referred to as *associations* between *contexts* at the endpoints of the communication. A context is essentially a collection of information concerning the end-point and the progress of an association. Individual contexts on a host are identified using a *key* value, assigned by the host to be unique within the host. The key is used on all outgoing packets to differentiate flows from different contexts. Returning packets carry a return key value (the same key with its most significant bit set), which uniquely maps the packet onto the appropriate context, thus making it easy to reference the information about the context. A *key exchange* is a procedure by which the initiator of the association is told the key value assigned to the responder context, so that return keys may be used in both directions.

Mapping incoming packets onto the appropriate context requires either the returned key (in which case the mapping is direct, since the return key identifies the context), or the (non-return) key and sender's source address (since this source address is where the key was created). The first case is called an "abbreviated context lookup" because the mapping is fast. The second is called a "full context lookup" because the mapping requires a translation from the (sender address, key) pair to the proper context.

There are seven packet types in XTP 4.0: FIRST, DATA, CNTL, ECNTL, TCNTL, JOIN and DIAG. The FIRST packet carries addressing and traffic specification information; traffic specifications are further negotiated using a TCNTL packet. The FIRST packet may also carry data. Subsequent data are carried in DATA packets. Control information is exchanged using CNTL (normal) and ECNTL (errors detected) packets. The JOIN packet is used for XTP 4.0 multicast mechanisms. The DIAG packet indicates protocol error conditions.

Before a FIRST packet arrives to establish an association, a context at the receiving host must be in a *listen* state. It moves to the *active* state if the association is accepted, that is, a FIRST packet meeting the listening context's criteria is received.

In XTP, it is the transmitter that requests status reports from the receiver(s). The SREQ bit in the header is used for this purpose. The transmitter sets a timer, WTIMER, when a packet containing SREQ is sent, to provide protection against loss of the request or the response. The response is carried in a CNTL or ECNTL packet, depending on the information to be reported.

The procedures for association management have been described thoroughly in the XTP Specifications[1,3] and the book by Strayer, et al[2].

Multicast Management

A multicast association consists of one transmitter context, and 0 or more receiver contexts. Each participant has a (unicast) transport-level address, and a key identifying its context. Each host has a (unicast) Delivery Service address. The group has a (multicast) transport-level address, and may have a (multicast) Delivery Service address.

When a multicast association is to be established, the procedures are similar to those for unicast communication. The receiving contexts that will form the initial group must enter the *listen* state, and establish the criteria for acceptance of an incoming FIRST packet, which contains the same information as a unicast FIRST packet. When the FIRST packet arrives at a receiving host, bearing the Delivery Service address and the key assigned to the group, and if it meets the acceptance criteria defined by a receiving context, it is accepted.

If multiple contexts are in the *listen* state when the FIRST packet arrives at a particular host, the search through the contexts continues until all contexts have been examined, and a mapping is constructed between the (sender address, key) pair and the set of all acceptable contexts. If no suitable context is found, the packet is simply dropped.

Depending on the desired reliability of the association, a response packet may or may not be returned once the association is established. In Revision 3.6 this is a CNTL packet; in Revision 4.0 it is a TCNTL (traffic control) packet, containing a response to the offered Traffic Specification. In both cases, the transmitter can learn the key used at each receiver, but the transport-level address of the newly-added receiver is unknown. (There is no field in a CNTL or TCNTL packet to carry this address.)

At this point, the receiving end-point has learned the value of the key assigned at the transmitting context, and can use it as a return key. If the receiver uses this key, the lookup at the transmitter is efficient, but the responses from individual receivers cannot be differentiated. Differentiation of the receivers would be possible if the receivers used their own keys, but this would be inefficient, as it would force a full-context lookup at the transmitter for each receiver packet that arrived.

In a reliable system, the transmitter must maintain information on each participant in the association, during the lifetime of the association. This information must be accessed each time that a packet is received from a receiver. To ensure group integrity, and reliable data flow, status reports will be requested periodically from the active receivers. It is essential that the local information in the transmitter be accessible quickly.

In Revisions 3.6 and 4.0, new receiving contexts may join an existing group by sending a JOIN packet, which contains the transport-level addresses of the group and of the joining receiver, the desired traffic specification, and key=0. In this case the joining receiver is identified at the transport level, but its key is unknown unless it chooses to perform a key exchange. An efficient mapping mechanism is not provided.

The association is established once the active group integrity criteria are met at the transmitter. (This is likely to require knowledge of the transport-level address of each receiver.) The population of the active receiver group may vary over time, and each change will trigger assessment of the integrity criteria for the group.

To properly manage an on-going group with full reliability requires mechanisms that permit both the full and unique identification of the receivers, and efficient access to the information at the transmitter.

Summary of Changes

The XTP 4.0 Addendum solves several issues that were outstanding in the XTP 4.0 Specification. These include:

- Uniquely identifying every active receiver at the transmitter
- Effectively controlling the membership of the active receiver group through the lifetime of the association
- Determining the transport-level address of all receivers in the association
- Providing the window size of each receiver to the transmitter as soon as possible

The solution given in the XTP 4.0 Addendum relies on assigning unique identifiers to each receiver, and recognizes that these identifiers can be obtained from the global key space at the transmitting host. These keys are then used as return keys by the receiver, achieving efficient lookup. Since neither the JOIN packet nor the TCNTL packet have sufficient fields to permit communicating the unique identifier (key) to each receiver and communicating the transport-level address of the receiver to the transmitter, a new packet type, JCNTL, has been defined to do this and subsume the functionality of the JOIN packet.

The rest of this paper discusses the XTP 4.0 Addendum multicast mechanisms and, when appropriate, how they differ from XTP 4.0.

JCNTL Packet Format

This section introduces the notation required to describe the reliable multicast procedures, and the packet format for the new packet type (JCNTL).

Necessary notation

Kg	Transmitter's local key, for the multicast group
Kg'	Transmitter's local key, as a return key
Kr	Receiver's local key
Kr'	Receiver's local key, as a return key
Ki	Key assigned by the transmitter, to uniquely identify receiver i
Ki'	The same, as a return key
TAg(m)	The multicast Transport Address of the group
TAt(u)	The unicast Transport Address of the transmitter
TAr(u)	The unicast Transport Address of the receiver
DA	The destination address field in an Address Segment
SA	The source address field in an Address Segment
Ag(m)	The multicast Delivery Service address of the group (probably a network address, but possibly a MAC address in restricted environments)
At(u)	The unicast Delivery Service address of the

	transmitter
Ar(u)	The unicast Delivery Service address of the receiver
dest	The destination address used by the Delivery Service
src	The source address used by the Delivery Service

Notes:

1. The Kx forms (Kg, Kr, Ki) are used in outgoing packets when there is not yet (or will never be) information about a return key, so a full context lookup must be forced at the destination. The Kx' forms (Kg', Kr', Ki') are used whenever possible, to permit abbreviated lookups.
2. The Transport Address values [TAg(m), TAt(u), TAr(u)] will be carried in the destination address and source address fields of the address segment of a JCNTL packet. They will have values appropriate to the Address Format in use (see section 2.4.1 of the XTP Specification).
3. The Delivery Service Address values [Ag(m), At(u), Ar(u)] will be carried in the destination address and source address fields of the Delivery Service packets that encapsulate the XTP packets. They will have values appropriate to the Address Format used by the Delivery Service.

The JCNTL packet is assigned packet type 7. The packet layout is as follows:

Header	(key, cmd, dlen, check, sort, sync, seq)	
rseq	(8 bytes)	common control part
alloc	(8 bytes)	
echo	(4 bytes)	
rsvd	(4 bytes)	for alignment
xkey	(8 bytes)	
Address Segment		
Traffic Segment		

NOTE: The alloc field in a JCNTL from a receiver must be interpreted as a window size, because the receiver may not know what the joining seq value will be, and so cannot calculate an actual alloc value.

Multicast Packet Exchanges

XTP multicast distinguishes between transmitter-initiated multicast as an "invitation" to join the multicast association, and the receiver-initiated join as a form of "polling."

In transmitter-initiated multicast, the packet exchange begins with a FIRST packet sent to the group address soliciting receivers. A receiver sends a JCNTL packet back to the transmitter, requesting to join the multicast association. The transmitter decides on the receiver, and if it is accepted, replies with a JCNTL packet telling the receiver that (1) it is now part of the association, (2) what its multicast receiver identifier is.

In a receiver-initiated join, a potential receiver sends a JCNTL packet to the group address requesting admission from the transmitter. If accepted, the transmitter replies with a JCNTL packet telling the receiver both pieces of information as above.

Transmitter-initiated multicast

The transmitter-initiated group formation proceeds as follows:

FIRST packet from the transmitter to the group:

dest	= Ag(m)
src	= At(u)
key	= Kg
DA	= TAg(m)
SA	= TAt(u)
initial Tspec	

When this FIRST packet arrives at a receiving host, the mapping (At(u),Kg)→(Kr1,Kr2,...) must be added to the translation table, where (Kr1,Kr2,...) denotes the contexts listening at Transport Address TAg(m) that also satisfy the acceptance criteria as listed in Section 5.3.2 of the XTP Specification, Revision 4.0.

Unreliable groups:

If SREQ is not set in the FIRST packet, then the transmitter does not care about receiver membership, and the receiver should be silent. (This is exactly the same as the requirement in unicast that the receiver be silent if the FIRST packet does not have SREQ set.) However, this does not prevent the transmitter from using SREQ/DREQ in the future to gather responses from listening receivers, and using these responses to advance its outgoing sequence numbers. These responses will of necessity be returned with key=Kg'; the transmitter will need some algorithm for coalescing the responses. Some level of error detection and correction is therefore still possible, but reliable reception by a defined group of receivers cannot be guaranteed.

Reliable groups:

If SREQ is set in the FIRST packet, the FIRST packet is an "invitation" to join the group. The required response is a "JCNTL request packet", with the following fields:

dest	= At(u)
src	= Ar(u)
key	= Kg'
alloc	= the current window size for this receiver
xkey	= Kr'
DA	= TAg(m)
SA	= TAr(u)
response Tspec	

Note that the initial window size for this receiver is also the alloc value, because the starting seq value is always zero.

Although this packet is a "response" to the SREQ in the FIRST packet, which would normally mean that SREQ should not be set, it is also a "request" to join the group. The receiver must protect this request with WTIMER. (Recall that we are dealing with a reliable group.) Therefore, to ensure completion of the packet exchange for association establishment, it is required that this packet have SREQ set.

After the sequence (FIRST packet, JCNTL request packet) has completed, the transmitter has the key (Kr') to be used when sending to the new receiver, and the Transport level address for this receiver, so the transmitter knows uniquely who the new member of the group is.

Since a packet sent with key=Kr' may be returned as a DIAG packet with key=Kr, the transmitter should add the mapping (Ar(u),Kr)→Ki to its translation table.

The transmitter must now issue a "JCNTL response packet", to complete the exchange. It allocates a key Ki from its key space, and communicates this to the receiver:

dest = Ar(u)
src = At(u)
key = Kr'
xkey = Ki'
DA = TAr(u)
SA = TAt(u)
Tspec

The Tspec can be the Null Traffic Specifier (tformat 0) to indicate "no change" when it occurs in a control packet.

This JCNTL response packet must not have SREQ set.

When the JCNTL response packet arrives at the receiving host, with xkey=Ki', then the receiving host must record Ki' as the key to be used when sending packets to the transmitter, and should add the entry (At(u),Ki)→Kr to its translation table. This enables correct delivery of a future DIAG packet with key=Ki.

If the contents of the DA field in the JCNTL response packet do not match the unicast Transport Address value associated with context Kr, then a protocol error has occurred. This should be signaled with a DIAG packet containing key=Kr, code = 6 (Protocol Error), and val = 14 (Invalid Address).

If the transmitter does not wish to continue to uniquely identify each receiver (in spite of having asked the receivers initially to identify themselves), it sends the same JCNTL response packet as above, but places Kg' in the xkey field. In this case there is no Ki allocated, no record of Kr, and no need to record a mapping between (Ar(u),Kr) and Ki.

Subsequent packet exchanges

Any packet sent from the transmitter to the whole group:

dest = Ag(m)
src = At(u)
key = Kg

Because the target contexts will have different keys, it is always necessary for the receiving host to use a full context lookup with the pair (At(u),Kg) to find the appropriate contexts (Kr1,Kr2,...) on a target machine.

Any packet sent from the transmitter to a specific receiver:

dest = Ar(u)
src = At(u)
key = Kr'

This uses abbreviated context lookup at the receiving host, to map to a single receiver context.

Any packet sent from a receiver to the transmitter:

dest = At(u)
src = Ar(u)
key = Kg' or Ki'

Here the receiver uses Kg', the key that it learned from the initial FIRST packet, or Ki', the key that it learned from the JCNTL response packet, depending on which is available. The transmitting host uses abbreviated context lookup to map to the appropriate entity.

Receiver-initiated multicast

When a receiver wishes to join an existing multicast group, it sends a JCNTL request packet, as follows:

dest = Ag(m)
src = Ar(u)
key = 0
alloc = window size
xkey = Kr'
DA = TAg(m)
SA = TAr(u)
offered Tspec

This JCNTL request packet must have SREQ set (to correspond with the JCNTL request packet described above).

The transmitter responds with a JCNTL response packet, as follows:

dest = Ar(u)
src = At(u)
key = Kg
alloc = join point
xkey = Ki' or Kg'
DA = TAr(u)
SA = TAt(u)
response Tspec

As for transmitter-initiated multicast, the transmitter

allocates a key Ki from its key space, and sends this as Ki' in the xkey field. It should also add the mapping (Ar(u),Kr)→Ki to its translation table.

If the transmitter does not, in fact, wish to uniquely identify joining receivers, then it sends Kg'. In this case there is no Ki allocated, no record of Kr, and no need to record a mapping between (Ar(u),Kr) and Ki.

When the JCNTL response packet arrives at the host corresponding to Ar(u), if there are no other contexts on this host associated with the group, the full context lookup will fail [because the host did not previously know the (At(u),Kg) pair]. The host will then match the destination address [TAr(u)] against listening contexts, and will find Kr. It will make an entry translating (At(u),Kg) to Kr.

If there are previously-enrolled members of the group on the host, the full context lookup will not fail, but the transport address [TAr(u)] will not be in the set of contexts indexed by the translation table, so an additional entry will need to be made once the context Kr has been found. If TAr(u) is already in the set of contexts indexed by the translation table, the JCNTL is a duplicate.

If no match can be found for TAr(u), then the context that originally issued the JCNTL request packet has vanished. This should be signaled with a DIAG packet containing key=Kg', code=3 (Invalid context), and val=0 (Unspecified). This can be handled at the transmitter in various ways, depending on the reliability semantics of the group. If the transmitter does not care, the DIAG can be ignored. If the transmitter has a set of one or more Kr' values for the receiving host, it can use these values to single out the departed context, and then act in accordance with the improved information.

(Note that key=Kr' cannot be used in the JCNTL response packet, even though it is known to the transmitter, because Kg must be communicated, even if Ki' is to be used for reverse traffic. In addition, the use of Kg triggers the update to the translation table, which otherwise would not happen if the packet were handed directly to the context Kr'.)

As for transmitter-initiated multicast, if the arriving JCNTL response packet has xkey=Ki', then the receiving host should add the entry (At(u),Ki)→Kr to its translation table. This enables correct delivery of a future DIAG packet with key=Ki.

If the JCNTL response packet containing Kg is lost, it will normally be recovered when the WTIMER expiration at the receiver causes the JCNTL request packet to be re-issued. However, some packets with key=Kg may arrive at the receiver; these will be ignored, but will be recovered once the JCNTL response packet is resent. Finally, receipt by the receiver of a packet with key=Kr', before the receipt of the (sent or resent) JCNTL response packet will confuse the receiver, because it will know neither Kg' nor Ki'. In this case the JCNTL request packet must be re-issued, even if WTIMER has not expired. (It is clear that at least one round trip time has elapsed!)

Alternate Joining Procedure:

For a receiver-initiated join, the receiver normally specifies xkey=Kr' in the JCNTL request packet. However, some implementations may find it inconvenient to allocate Kr at the time that the JCNTL request packet is issued. The receiver indicates this by setting both key=0 and xkey=0 in the JCNTL request packet.

The transmitter responds with a normal JCNTL response packet, containing key=Kg and xkey=Ki' or Kg'. If Ki has been allocated, the transmitter records Ki as the "key to use when addressing this receiver", because it does not (yet) know Kr.

Use of Ki in this way will require a full context lookup at the destination. Also there is a window of vulnerability here if the JCNTL response packet is lost. If key=Ki is used to send a packet to this receiver, but the receiver does not know about Ki, then a DIAG will be returned with key=Ki', and the association will be in danger of being aborted. It is therefore recommended that the transmitter minimize the use of key=Ki to address an individual receiver.

When the JCNTL response packet arrives at the receiving host, the full context lookup proceeds as it would for the normal receiver-initiated case: the (At(u),Kg)→Kr mapping is added, and the (At(u),Ki)→Kr mapping is created if xkey=Ki'. In addition, if xkey=Ki', then the receiver may wish to initiate a key exchange with the transmitter, to send it Kr'. This should be done by sending a TCNTL (key=Ki', xkey=Kr', SREQ) containing the Null Traffic Specifier (tformat 0). The presence of SREQ will solicit a response confirming the key exchange.

When the TCNTL (key=Ki', xkey=Kr', SREQ) packet arrives at the transmitter, the transmitter records Kr' as the "key to use when addressing this receiver", and adds the mapping (Ar(u),Kr)→Ki to its translation table. The window of vulnerability mentioned above has now closed, because the transmitter is now sure that Ki is known at the receiver. However, the transmitter will never use Ki for subsequent packets, because it has also learned Kr'. The transmitter then responds to the SREQ in the TCNTL in the normal way.

Simultaneous FIRST and JCNTL (key=0) packets

It is possible for the transmitter-initiated and the receiver-initiated sequences to begin simultaneously. In

principle, it would be possible for an implementation to drop *one* of the extraneous packets. However, two different implementations might end up dropping *both* of the packets, so the association would never get established. Implementations are therefore required to respond to both, with the expectation that the extra JCNTL packets that result will be dropped by the duplicate detection mechanisms, once it is assured that the association has been established. The details are as follows:

During a transmitter-initiated sequence, once the FIRST (key=Kg) packet has been sent, the transmitter is expecting a JCNTL (key=Kg') as a reply. If a JCNTL (key=0) is received instead, the action to be taken is a combination of the action for JCNTL (key=0) and the action for JCNTL (key=Kg'): the response of the joining receiver must be added to the set of responses that will be communicated to the transmitting user once the criteria for the new group have been met (see the section below on Multicast Group Management), and the transmitter must respond to the receiver using the normal response to a JCNTL (key=0) packet.

During a receiver-initiated sequence, once the JCNTL (key=0) packet has been sent, the receiver is expecting a JCNTL (key=Kg). If a FIRST packet arrives, it should be responded to by issuing a JCNTL request packet with key=Kg'.

Key Management for Reliable Groups

This section provides further insight for the key management during multicast associations with fully-reliable groups. It presents the progress of the knowledge gained during the packets exchanges, about the keys to be used while sending information and the keys to be expected in arriving packets.

Transmitter state diagram

Figure 1 is a state diagram for the transmitter, showing the key exchanges. A '+' signifies an input and a '-' signifies an output. Transitions that definitely take place are shown with a continuous line and transitions that may take place at some time, but need not ever occur, are shown with a dashed line. Labeled states in the diagram represent the progress in the knowledge of the various keys. The column headings have the following meanings:

TxI	Transmitter-initiated
RxI	Receiver-initiated
RxA	Receiver-initiated; alternate procedure

In the beginning, for a transmitter-initiated exchange (column TxI), the transmitter knows Kg, and sends it in a FIRST packet. This progresses the exchange to state TA. When the transmitter receives a JCNTL request packet from a joining receiver, it learns Kr', assigns Ki,

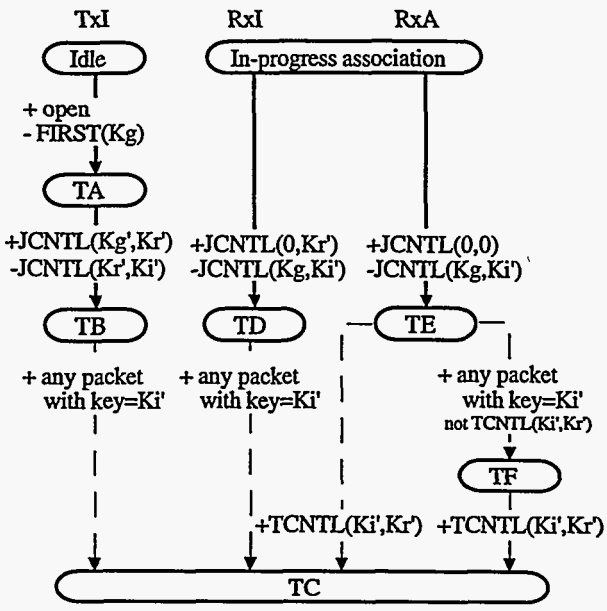

Figure 1. State diagram for the transmitter's key exchanges.

and sends a JCNTL response packet containing xkey=Ki'. The transmitter records the translation (Ar(u),Kr)→Ki to permit correct processing of returned DIAG packets. This progresses the exchange to state TB.

In state TB, the transmitter will use key=Kr' to single out the receiver, and key=Kg to send to the whole group. Since the JCNTL response packet may be lost, the only packet type that can safely be sent to the receiver in this state is a (repeated) JCNTL(Kr',Ki'). The transmitter expects packets to be returned to it containing key=Ki'. However, it is vulnerable to packets with key=Kg', until it receives any packet with key=Ki', at which time it knows that the receiver has recorded the value of Ki'. This progresses the exchange to state TC. (It is likely that this transition will take place naturally, when the receiver responds to a packet with key=Kg and SREQ set.)

For a normal receiver-initiated exchange (column RxI), the transmitter learns Kr' when the (unsolicited) JCNTL request packet arrives. It sends a JCNTL response packet containing key=Kg and xkey=Ki'. The transmitter records the translation (Ar(u),Kr)→Ki to permit correct processing of returned DIAG packets. This progresses the exchange to state TD. The rest of this exchange is identical to the steps for a transmitter-initiated exchange, except that the only packet that can safely be sent in state TD is a repeated JCNTL(Kg,Ki').

For a receiver-initiated exchange using the alternate procedure (column RxA), the transmitter learns only the transport-level address of the receiver when the JCNTL request packet arrives. It sends a JCNTL response packet containing key=Kg and xkey=Ki'. This progresses the

exchange to state TE.

In state TE, the transmitter will use Ki to single out the receiver, and Kg to send to the whole group. Since the JCNTL response packet containing Ki' may be lost, the only packet that can safely be sent to the receiver in state TE is a (repeated) JCNTL(Kg,Ki'). The transmitter expects packets to be returned to it with key=Ki'.

At some point in the future, the receiver may send the value of Kr' to the transmitter in the xkey field of a TCNTL packet. Once it learns Kr', the transmitter will use Kr' to single out the receiver, instead of using Ki. The transmitter records the translation (Ar(u),Kr)→Ki to permit correct processing of returned DIAG packets. This progresses the exchange to state TC. Alternatively, the transmitter may first receive some packet (other than a TCNTL) with key=Ki'. This moves it to state TF, where it knows that any packet with key=Ki can safely be sent, but it still does not know the value of Kr'.

Finally, for any receiver-initiated case, if the JCNTL response packet is lost, the receiver becomes a member of the multicast group without knowing it. It has not learned Kg, so it will not receive any packet sent to the group, and will not reply to any group synchronization. If it does not achieve membership in the association quickly enough [e.g., by repeating the JCNTL(key=0) packet], the transmitter may remove it from the group. This may result in the receiver actually "rejoining" the group rather than "joining" the group.

The knowledge about keys at the transmitter can be summarized as shown in Table 1. The symbols -, E, K, P, S, V have the following meanings:

- \- the transmitter has no knowledge of this key
- E packets are expected to arrive with this key
- K this key is known to the transmitter, but will never be used
- P it is possible that a packet will arrive with this key, but only under error situations
- S this key will be used to send packets
- V packets are not expected to arrive with this key, but the transmitter is nevertheless vulnerable to them (especially if the packet telling the receiver to use this key is lost)

Table 1. Transmitter key knowledge

State	Kg	Kg'	Ki	Ki'	Kr	Kr'
TA	S	E	-	-	-	-
TB	S	V	K	E	P	S
TC	S	P	K	E	P	S
TD	S	V	K	E	P	S
TE	S	P	S	E	-	-
TF	S	P	S	E	-	-

Receiver state diagram

Figure 2 is a state diagram for the receiver, showing

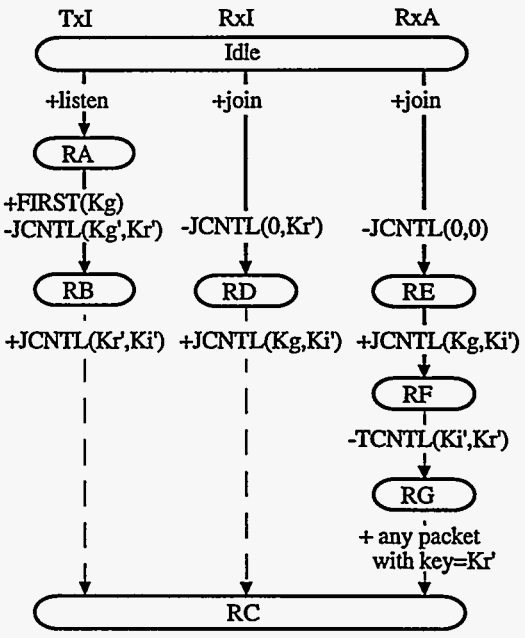

Figure 2. State diagram for the receiver's key exchanges.

the key exchanges. The symbols and column headings have the same meanings as those in Figure 1.

For a transmitter-initiated exchange (column TxI), the receiver moves to state RA as a result of a "listen" primitive. When the FIRST packet arrives, the receiver issues a JCNTL request packet with xkey=Kr'. The receiver records the translation (At(u),Kg)→(Kr1,Kr2,...) to permit correct processing of incoming packets addressed to the group. This progresses the exchange to state RB.

At this point, the receiver expects packets to arrive with key=Kg (if intended for the whole group), or with key=Kr' (if intended for the individual receiver). The receiver will use Kg' to send packets to the transmitter. When the JCNTL response packet arrives, the receiver learns from the xkey field its new key assignment (Ki'), and will use this in future to send packets to the transmitter. The receiver records the translation (At(u),Ki)→Kr to permit correct processing of returned DIAG packets. This progresses the exchange to state RC.

For a normal receiver-initiated exchange, the receiver moves to state RD as the result of a "join" primitive. It issues a JCNTL request packet with xkey=Kr'. When the JCNTL response packet arrives, the receiver learns the group key (Kg), and its individually assigned key (Ki'). This progresses the exchange to state RC. The receiver will expect packets to arrive with key=Kg (if intended for the whole group), or with key=Kr' (if intended for the individual receiver). The receiver will use Ki' to send packets to the transmitter. The receiver records the

translation (At(u),Ki)→Kr to permit correct processing of returned DIAG packets.

For a receiver-initiated exchange using the alternate procedure, the receiver moves to state RE as the result of a "join" primitive. It issues a JCNTL request packet with xkey=0. When the JCNTL response packet arrives, the receiver learns the group key (Kg), and its individually assigned key (Ki'). This progresses the exchange to state RF. At this point, the receiver expects packets to arrive with key=Kg (if intended for the whole group), or with key=Ki (if intended for the individual receiver). The receiver records the translation (At(u),Ki)→Kr to permit correct processing of these individually-addressed packets. The receiver will use Ki' to send packets to the transmitter.

The transition to state RG, if it occurs, will inform the transmitter that the receiver wants the transmitter to use Kr' in lieu of Ki when sending packets to this specific receiver. However, the receiver is still vulnerable to packets with key=Ki, until it receives any packet with key=Kr', which tells the receiver that the request has been received, and moves the receiver to state RC. The translation (At(u),Ki)→Kr is now only necessary to permit correct processing of returned DIAG packets.

The knowledge about keys at the receiver can be summarized as shown in Table 2. The symbols -, E, K, P, S, V have the same meanings as for the transmitter, with the roles of the receiver and the transmitter reversed.

Table 2. Receiver key knowledge

State	Kg	Kg'	Ki	Ki'	Kr	Kr'	
RA	-	-	-	-	K	-	Note1
RB	E	S	-	-	K	E	
RC	E	K	P	S	K	E	
RD	-	-	-	-	K	E	
RE	-	-	-	-	-	-	
RF	E	K	E	S	K	-	
RG	E	K	V	S	K	E	

Note 1: Kr may remain unassigned until the FIRST packet arrives (transition to state RB)

Multicast Group Management

The XTP Specification concentrates on the mechanisms to be used for gathering information about potential participants in a multicast group, and leaves unspecified the policies for deciding on the membership of the group. These policies may be determined directly by the transmitting user, or by a module that is closely associated with the XTP implementation.

When the group is forming, certain actions are reasonable and permitted, but these may become unreasonable or undesirable after this point. Without implying any particular implementation, the phrases "before the group has formed" and "after the group has formed" will be used to differentiate these two time periods. It is expected that the event "the group has formed" will be visible to the transmitter in some way.

Issuing duplicate FIRST and JCNTL packets

Various implementations may have different ideas of how hard they will work to ensure that the criterion for group formation is met. (Example criteria would be "at least K receivers" or "as many receivers as can be found within M seconds".) One mechanism for ensuring that as many potential receivers as possible are attracted to the group is to deliberately send duplicate FIRST packets upon expiration of the WTIMER that was set when the SREQ was included in the FIRST packet. While this is permitted, it is important that the transmitter be "well-behaved". Therefore, it is required that the transmitter use a variation of the Synchronizing Handshake to achieve this—specifically, it is required that the spacing between duplicated FIRST packets be backed off exponentially, and that the repetitions be limited by "retry count" or CTIMEOUT (or smaller values). Once this period of solicitation has completed, the group must be considered to be formed (or the association must be aborted), and no further FIRST packets may be sent.

Similarly, in a receiver-initiated join, the joining receiver is permitted to deliberately send duplicate JCNTL (key=0) packets after its WTIMER expires, but must also be "well-behaved"; the receiver is required to use a variation of the Synchronizing Handshake, as outlined in the previous paragraph.

Responding to duplicate FIRST packets

In the unicast case, a receiver is required to respond (using CNTL or TCNTL) to a duplicate FIRST packet containing SREQ or DREQ. While this is still true, in general, for the multicast case, because the JCNTL request packet from a joining receiver may have been lost, it is not necessary to force the issuing of a JCNTL response packet if the receiver is already in possession of Ki'. In this case, the JCNTL request packet should be reissued, but without the SREQ bit being set. It is not acceptable to ignore the duplicated FIRST packet, because the formation of the group may depend on getting responses from all members of the group, with all responses having the same ECHO value.

Late JCNTL (key=Kg') packets

After a group has been formed, a (late) JCNTL (key=Kg') packet may arrive. This arrival should be