ANSI E1.17- 2015, Architecture for Control Networks– Session Data Transport Protocol

Part of ANSI E1.17 - 2015, which was approved by the ANSI Board of Standards Review on 21 May 2015.
This part has no substantive changes from the 2010 edition.

PLASA
630 Ninth Avenue, Suite 609
New York, NY 10036
USA

Abstract

The E1.17 protocol suite is intended to provide the next-generation standard for the distribution of data in lighting, entertainment technology and other control networks. Ideally, E1.17 will unify control networking in its field, allowing a single network to carry many different kinds of data and to connect equipment from many different manufacturers. This document describes a generalized transport layer protocol that carries more specific higher level data protocols. The SDT transport layer is intended to provide reliable, ordered, and flexible delivery of data from a leader to one or more members within a session.

Contents

Figures

Tables

Suggestions for improvement of this standard are welcome. They should be sent to ESTA.

1 Introduction

Control of devices in an E1.17 system using Device Management Protocol [DMP] requires that multiple short command messages for different components be packed together and sent to the entire group of components. Messages are packed together to reduce the network overhead per message. The packing of messages follows as a result of the fact that the common communication transport layers available (TCP/IP, Ethernet) are optimized for the transfer of packets larger than a typical DMP message. Session Data Transport (SDT) provides group messaging enabling this packing and delivery to occur.

During the development of DMP, it was recognized that certain messaging features fulfil a generic need that goes beyond DMP. Providing these features in SDT, a separate protocol, fits with the modular architecture and other design goals of the E1.17 project. For example the ability to add new protocols on top of SDT allows extensibility.

SDT provides a session based, ordered delivery mechanism in which each session forms a bi-directional transport connection for generic data flowing between a single leader and one or more members. It offers a great deal of flexibility over reliability, latency, and buffering requirements, that may be tuned to meet the needs of the specific data being sent and the type and size of network environment. A typical system will have many sessions operating at one time.

Within a session, ordering, reliability and on-line status are all achieved by a combination of using sequence numbers in packets with receivers (members) negatively acknowledging (*NAK*ing) missed messages and positively acknowledging the point in the sequence that they have reached (when asked by the leader).

Reductions in network traffic are possible when *reliable delivery* of messages is provided by the underlying communication transport layer. Reliable delivery can ensure that messages are delivered, allowing a controller to avoid sending identical values multiple times to achieve *reliable control*. This reduces both network traffic and host processing requirements. Reliable delivery also allows a controller to send only the values of those control variables that have changed since they were last sent, further reducing network traffic. Finally, the presence of reliable delivery allows for the robust use of control variables whose effect on systems depends on transitions in their values as well as on the immediate values of those variables. The use of SDT in this reliable delivery mode makes the most sense when control values are changing infrequently compared to the target rate of update of the system.

SDT also provides unreliable ordered delivery of messages, which may be freely mixed with reliable delivery of messages. The use of unreliable delivery is most appropriate in many situations, for example, when the application uses other means to achieve reliability, such as time-outs or repeated sending of messages (streaming).

Repeated sending of (potentially redundant) values on the network is a method for obtaining *reliable control* when the effect of a control variable on the system does not depend upon its former value and only depends upon its current value. This is the model of control used in DMX [DMX512] systems. Streaming of control values does not provide reliable control when all of the values a control variable goes through are important to its effect on the system.

SDT supports the repeated sending of values as a means to obtain reliability. The use of SDT in this steaming mode makes the most sense when control values are being generated in a continuous stream of updates, all of which are calculated at the source of control, and which are changing near the target update rate of the system.

2 Scope, Purpose, and Application

2.1 Scope

This standard defines a protocol to be used within E1.17 systems to transport sequenced data both reliably and unreliably between E1.17 components.

This protocol is intended for use within the E1.17 protocol suite. This protocol does not provide for security, authentication, or encryption. These functions may be provided by industry standard protocols operating at other layers of the protocol stack. This protocol may find use outside of the E1.17 protocol suite but this is not a primary design goal.

2.2 Purpose

The purpose of this standard is to specify the transport protocol to be used within E1.17 systems.

2.3 Application

This protocol is to be used within E1.17 entertainment control systems to transfer data from component to component.

3 Sessions

SDT provides a bi-directional *session* based transport mechanism for E1.17 component communications. A session transports a single client protocol. The advantages of session based communication (traffic partitioning, connection status, ordering, optional reliability, flexibility, and efficient addressing) are made available as a service to client protocols. SDT builds sessions on top of unidirectional *sequenced channels*, see Section 3.3.

A session consists of a single *leader* with a group of zero or more *members.* The bulk of traffic in a session is nearly always from the leader to the members, though members also send messages to the leader. Members shall not send messages to other members. Individual data items from the leader may be transported to one or all members of the session. An E1.17 system will typically have many sessions operating concurrently and any component may be leader or member (or both) of zero or more sessions at any one time. In a DMP system for example, a controller may lead one or more sessions to send commands to and receive responses from the devices it is controlling. Devices that are generating asynchronous *event* messages might each lead one or more sessions, with the components monitoring those events being members of the session or sessions.

Figure 1 shows a block diagram of traffic in a session.

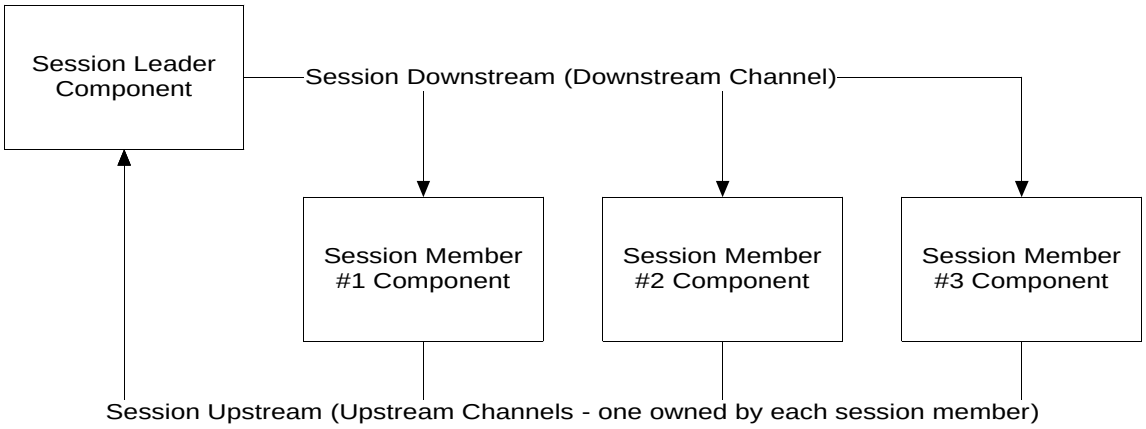


Figure 1: A Bi-directional SDT Session

3.1 Session Identity

A session is uniquely identified by the combination of:

- the component ID (CID) of the session leader,
- the session number the leader has assigned to the session, and
- the protocol ID of the protocol carried by the session.

The *session number* is defined as the *channel number* of the *sequenced channel* carrying the session's downstream traffic (see Section 3.3). In SDT messages, the leader CID is obtained from the Root Layer Protocol PDU address fields. For messages that do not originate from the leader, it is included in the message data.

3.2 Downstream and Upstream Traffic

Within a session, messages from the leader to the members are said to be sent *session downstream*. Messages from a member to the leader are said to be sent *session upstream*. Only client protocol messages, e.g., DMP messages, carried by SDT are spoken of as *session upstream* or *session downstream* messages.

3.3 Sequenced Channels

Sessions are built on top of *sequenced channels.* Sequenced channels are unidirectional communication channels (unicast or multicast) from an owner component to one or more member components. All messages sent on sequenced channels are sent to a channel's destination address from the owner to its members with the exception of *NAK*, *Join Accept, Join Refuse,* and *Leaving* messages, which are sent to a channel's source address from one member to the owner. Two sequenced channels (in opposite directions) operate together to provide ordered bi-directional message delivery between participating components. The combination of outgoing sequenced messages from one session leader component to one or more member components (on a single sequenced channel owned by the leader) with the response messages from those members back to the leader (on individual sequenced channels, each owned by one member) forms a *bi-directional session*. A sequenced channel may carry only one session for a given client protocol because the protocol id is the only portion of the session id that distinguishes it from other sessions on a channel. However, a channel may carry multiple sessions so long as they are for distinct client protocols. In this case, multiple sessions share sequencing on the same channel.

Sequenced channels transport three categories of traffic: *SDT internal traffic, Session downstream traffic, and Session upstream traffic*. SDT internal traffic is necessary for reliability, ordered message delivery, channel management, and session management. Session upstream and session downstream traffic transport other protocols, for example, DMP.

Communication between two components in SDT occurs over reciprocal pairs of channels. For each outgoing sequenced channel from one component to another, there exists a specific reciprocal channel that is paired to it for its entire lifetime. Regardless of the direction of message and response, all communication initiated on this pair is responded to on this pair. Each component may lead sessions on this pair.

Even though there is a tight pairing of reciprocal channels connecting any two components, a component may use a single channel as the reciprocal channel for many incoming channels so long as they are from different components.[1]

When a component is asked to join a sequenced channel, it shall join as requested, unless it does not have sufficient resources to do so. In order to successfully complete this join, the requested component must either already have established or must create a sequenced channel on which the requesting component is a member. It is an error if this reverse channel does not exist and cannot be established within the *Join* timeout period. If

[1] The ability to use a single channel as the reciprocal partner for channels from many components lowers the processing and resource demands of an SDT implementation.

this error occurs, the member shall send a *Join Refuse* message to the channel owner indicating this condition as the reason code.

A sequenced channel has a destination (outbound) address in the underlying transport, which may be unicast or multicast.[2] Where the channel has members residing at more than one network address, the destination address shall be multicast. Where there is just one member or multiple members at one network address, the destination address may be unicast. The destination address need not be unique to a single channel – for example if two components own channels sending traffic to the same group of members, they may both use the same destination address.

All outbound messages in sequenced channels (sent from the channel owner to channel members) are carried in sequenced wrappers and are therefore ordered. These wrappers may be reliable or unreliable. Reliable wrappers are resent if lost or dropped, unreliable wrappers are not. Use of reliable sequenced wrappers in combination with *ACK*s and *NAK*s add reliability to the channel.

A sequenced channel also has a source (inbound) address that is a unicast address obtained from the transport layer. The only messages sent to the source address of a channel (inbound) are *NAK*, *Join Refuse*, *Join Accept* and *Leaving*. These messages are not sent in wrappers and are not sequenced.

The component assigns a channel number to a sequenced channel when it is first created. *Channel numbers* are integers in the range 1..65535. Channel number 0 is reserved. No component shall have multiple active channels with the same number at the same time. Additionally, it is important for the component to avoid reusing channel numbers in situations such as a restart or temporary loss of network connectivity (see Section 3.5).

Figure 2 shows a block diagram of traffic in a channel.

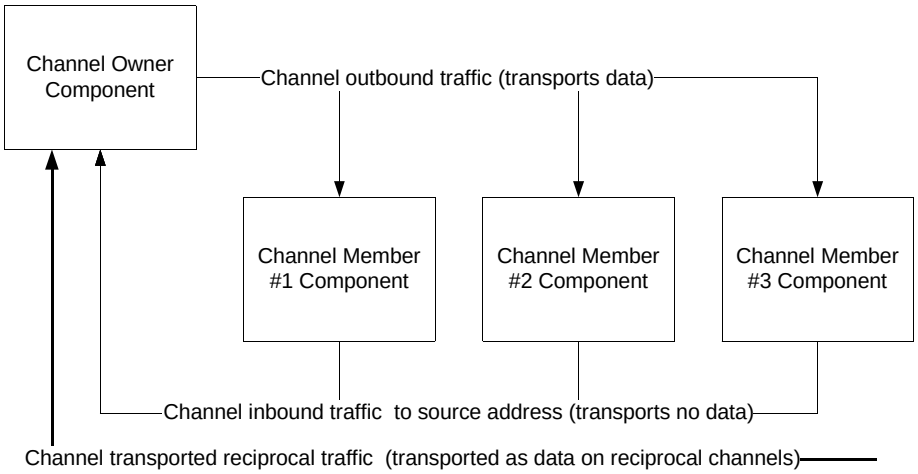


Figure 2: An SDT Sequenced Channel (Unidirectional data transport)

Note that Figure 2 shows that some of a channel's messages are received on a reciprocal channel. A single channel may not exist alone, as at least one reciprocal channel from each member is required. The wrapped messages *ACK*, *Connect Accept, Connect Refuse, and Disconnecting,* are called reciprocal inbound traffic. They are sent outbound on reciprocal channels because:

- this ensures that open sessions remain bi-directional at all times;

[2] In this context, unicast means delivery of a single message to a single receiver and multicast implies delivery of a single message to multiple receivers. The details are transport specific.

- it is more network bandwidth efficient to send *ACK* messages along with other traffic;
- messages other than *ACK* benefit from the reliability of the reciprocal channel rather than member components having to use timeouts and send directly to the source address of the channel.

Figure 3 shows a block diagram of the simplest channel configuration for a session: two reciprocal channels with only one member in each. Note that this pair of channels could also carry a session in the other direction. To actually establish this session, the *Connect*, and *Connect Accept* messages would need to be sent over these channels.

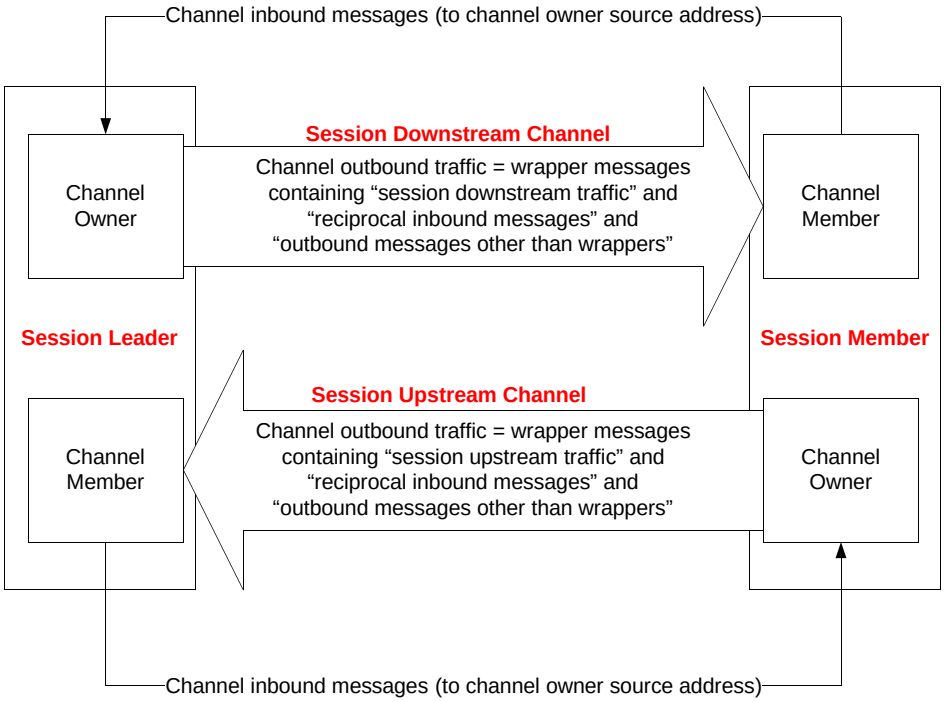


Figure 3: Simplest Session - Reciprocal Channels each with single member.

3.4 Ad-hoc and Channel Messages

SDT messages may be classified as either ad-hoc or channel messages. Additionally channel messages may be subdivided into those messages that are sent outbound, inbound, and reciprocal inbound.

3.4.1 Ad-hoc Messages

Join, Get Sessions, and *Sessions* messages are referred to as ad-hoc messages because they are not sent within a channel. *Join, and Get Sessions* are sent to ad-hoc addresses. An ad-hoc address is an address in the underlying transport (e.g., IPv4 address and port) at which a component is prepared to received ad-hoc messages. The *Sessions* message is a response message sent to the address that the Get Sessions message originated from (i.e., the source address of the packet containing the *Get Sessions* message).

Ad-hoc addresses are provided by the discovery features of E1.17 (see [Discovery-IP]) and as the required source address of the *Join* message. Source addresses are provided by the underlying transport, such as IP address and port in the case of UDP transport.

3.4.2 Channel Messages

Within a channel, messages from the owner to the member(s) sent to the channel's destination address are said to be sent *channel outbound*. Channel outbound messages are listed in Figure 4. Note that *Leave*, and the two *Wrapper* messages may also be sent to *potential* channel members in the join pending state during channel creation (see Section 3.5.1.2). The *Reliable Wrapper* and *Unreliable Wrapper* messages carry all session upstream and downstream traffic. All *session* level traffic is thus channel outbound.

- *Channel Params*
- *Leave*
- *Connect*
- *Disconnect*
- *Reliable Wrapper*
- *Unreliable Wrapper*

Figure 4: Channel Outbound Messages

Channel inbound messages are sent to the channel owner at the channel's source address. These messages are not sequenced, nor are they sent reliably. Channel inbound messages are listed in Figure 5.

- *Join Accept*
- *Join Refuse*
- *NAK*
- *Leaving*

Figure 5: Channel Inbound Messages

Channel reciprocal inbound messages are SDT response messages sent from a channel member to a channel owner on another channel. They are transported within outbound wrapped messages on a reciprocal channel. They apply as inbound messages to the channel identified in the "Association" field of the SDT Client block that contains them, see Section 4.4.6.1.

- *ACK*
- *Connect Accept*
- *Connect Refuse*
- *Disconnecting*

Figure 6: Channel Reciprocal Inbound Messages

3.5 Session and Channel Creation

Sessions are led, and channels are owned. A session has a single downstream sequenced channel and an upstream sequenced channel for each session member. The session leader owns the downstream channel. Each session member owns its upstream channel.

Sessions are created by the component that will lead them and are established on top of sequenced channels. Session membership is always initiated by the leader (there is no mechanism for a component to request membership in a session). All downstream traffic in a session is sent on a single channel owned by the session leader. When a component needs to create a session it shall either create a new channel or select a previously

created channel with an appropriate destination address for the session.[3] It shall also select the other channel parameters based on the purpose of the session and the characteristics of the system. If using an existing channel, the leader may desire to change the channel parameters to meet the needs of the new session. Channel parameters are described later (see Sections 4.4.1.2 and 4.5.2).

Each time a session leader starts-up it must assume that there may be components on the network that still consider themselves to be members of that leader's previous sessions. Obvious reasons for this are loss of power or interruption of physical network connectivity. Although using unique sessions numbers on equipment start-up is optional, to minimize incorrect system operation that may result from duplicate session numbers, components that lead sessions should use different session numbers each time they are initialized. This could be accomplished, for example, by saving the most recently generated session number to persistent storage and then incrementing it on restart, or through the use of the time of start-up to choose a session number. Note that since the session number is the channel number of the downstream channel (the channel carrying the session's downstream traffic), this means that start-up channel numbers are the actual quantities whose generation would be managed to avoid duplicates.

3.5.1 Channel Membership

Membership of components in sequenced *channels* is initiated by a handshaking operation that begins with the owner sending a *Join* message.

Termination of membership for a component in a sequenced channel is initiated either by the owner sending a *Leave* message to a member or by the member sending a *Leaving* message to the owner.

Membership of a component in a sequenced channel can also be terminated by timeouts (expiry), or unrecoverable data loss. These are error conditions and are not to be used for programmed termination.

3.5.1.1 Member Identifiers

When a component joins a channel, it is given a member identifier (MID) by the channel owner. MIDs are used for addressing within channels. The owner shall ensure that each member's MID is unique within the channel. The channel owner does not have a MID.

MIDs are 16-bit unsigned numbers. MID 0 (zero) is reserved and shall not be assigned. MID 65535 is reserved and means all session members. This places a limit of 65534 members per channel.

3.5.1.2 Channel Joining

A channel owner wishing a component to join a channel sends the ad-hoc *Join* message to the component. The *Join* specifies the channel parameters that the component is to use. A channel owner can expect one of three results from sending a *Join* message: a *Join Accept* response, a *Join Refuse* response, or no response. On sending the initial *Join*, a channel owner shall wait for a response according to the retry and timeout rules specified in Section 5.11. In the case of no response, the channel owner shall assume that the *Join* message was not received by the component and that the join process has failed.

When a potential channel member receives a *Join* message, it shall respond with a *Join Accept* message to the unicast source address of the channel being joined. (The *Join Accept* message is sent channel inbound.) At this point, the potential member is in the join pending state. This *Join Accept* message, from the potential member, contains a channel number that identifies the reciprocal channel that the member will couple with the channel being joined. This join pending state is maintained until an *Ack* message is sent by the potential member on the reciprocal channel. (The *Ack* message is sent channel reciprocal inbound.)

[3]The method for allocating the addresses is dependent on the transport protocol and is defined in the interoperability profile for SDT on that transport.

This *Ack* message shall be sent on the reciprocal channel immediately if it already exists. In the case where no reciprocal channel yet exists, it must be created before the *Ack* can be sent. Note that in this latter case, the *Ack* shall be sent immediately upon the receipt of the *Join Accept* for this reciprocal channel.

On receipt of a *Join Accept* message, a channel owner shall expect an *Ack* message from the member on the reciprocal channel indicated in the *Join Accept* message. If this *Ack* message is not received within the RECIPROCAL_TIMEOUT period (see Section 7.3) the *Join* process has failed. In this case, the owner shall send a *Leave* message to the failed member. This is to ensure that the member is not left in the joined state. Until it has received an *Ack from a potential member* the owner should recognize that a potential member may be in the join pending state and will only process a restricted subset of wrapped messages. On receipt of this initial *Ack,* the owner knows that the member is established in the channel sequence and will process the full set of wrapped messages as outlined in this standard. (Note that the owner must not wait for this *Ack* before sending its own *Ack* for the reciprocal channel as this would cause a deadlock condition.)

While in the join pending state, the potential channel member shall sufficiently process all wrappers received on the pending channel in order to track sequence numbers, to identify the reciprocal inbound *Ack* indicating that the channel join has completed within the RECIPROCAL_TIMEOUT period, and to leave the channel in response to a *Leave* message sent by the channel owner. All other messages on the pending channel, whether addressed to all members or directly to the pending member, shall be discarded until it has sent the *Ack* message via the reciprocal channel. Note that this message is typically sent as soon as a *Join Accept* has been received on the reciprocal channel. In the case where the reciprocal channel already exists, this message is sent immediately.

The reliable sequence number sent in the *Ack* shall be the highest reliable sequence number (respecting sequence wrap-around) either from the *Join* message or from any subsequent wrapper received while in the join pending state. This reliable sequence number becomes the ACK point for the member in the channel. Once this *Ack* is sent, the member shall process all subsequent channel messages as detailed in this specification until it leaves the channel as described below.

If a potential channel member is unable to join the channel (e.g., because of resource limitations or policy restrictions), it shall reply to the source address of the channel owner with *Join Refuse* indicating the appropriate reason code. If that potential member has previously sent a *Join Accept* and entered the join-pending state, but does not have a channel in the reverse direction and has not received a *Join Accept* for a channel it is trying to establish in the reverse direction, within the timeout rules discussed above, it shall send a *Join Refuse* with the appropriate reason code.

If a component receives a *Join* message for a channel of which it is already a member and that message specifies the same member ID as the prior *Join* (see Section 3.5.1.1), the member shall respond with *Join Accept*. This condition occurs if the owner does not receive the first *Join Accept* and re-sends the *Join*. If a member receives a *Join* message for a channel of which it is already a member and that message indicates a different member ID, then it shall respond with a *Join Refuse* with the reason code "Already Member".

3.5.1.3 Channel Leaving

A component wishing to leave a channel shall send a *Leaving* message to the source address of that channel. A channel owner may also send *Leave* to any member at any other time, to require it to leave the channel (owner initiated leave). Except in the case where a channel owner receives a *Leaving* message from a member, *Leave* is always the very last message sent to a member within a channel. Having sent a member the *Leave* message, the owner shall no longer respond to messages from that member, including *NAK* messages. This means that the owner will not resend the *Leave* message if the member *NAK*s it, however, the member will leave the channel after executing the *NAK* cycle retry logic and failing to receive the wrapper. In short, the owner shall cease to treat the now former member as a member of the channel.

On receipt of a *Leave*, a member shall send a *Leaving* message to the leader. *Leaving* is always the very last message a member sends within a channel. Having sent the *Leaving* message, the member shall cease to

operate as a member of the channel. In order to communicate within the channel again it needs to be requested to join the channel again, in the normal way as described in this Standard.

On receipt of a *Leaving* message from a member, the channel owner shall cease to send any messages to that member, and shall treat that component as having left the channel.

3.5.1.4 Channel Expiry

Channel expiry is a mechanism that prevents members from keeping a channel open indefinitely if a fault causes loss of communication from the owner.

Whenever a component becomes a member of a channel, it is given an expiry time specified in seconds. If at any time the member has received no correctly sequenced messages from the channel owner to any member of the channel for a period greater than the expiry time, the channel shall be considered to have expired. In choosing reasonable values for channel expiry time consideration should be given to required latency and the characteristics of the network link. Where networks contain links with the potential for high latency (e.g. wireless or internet segments), expiry times may need to be considerably longer. The expiry time also determines the values of other channel timing parameters (see Section 7.3 and relevant EPIs).

All communication depends upon a channel and its reciprocal channel. Channel pairs should have similar timing parameters. Channel expiry time should be matched to the expiry time of the reciprocal channel.

When a member determines that a channel has expired, it shall transmit a single *Leaving* message to the source address of the sequenced channel and shall operate as specified in Section 3.5.1.3. The owner of a channel shall ensure that outbound traffic within a channel is sufficiently frequent to prevent it from expiring in normal operation.

State diagrams for a potential channel owner and channel member components are shown in Figure 7, and Figure 8 below.

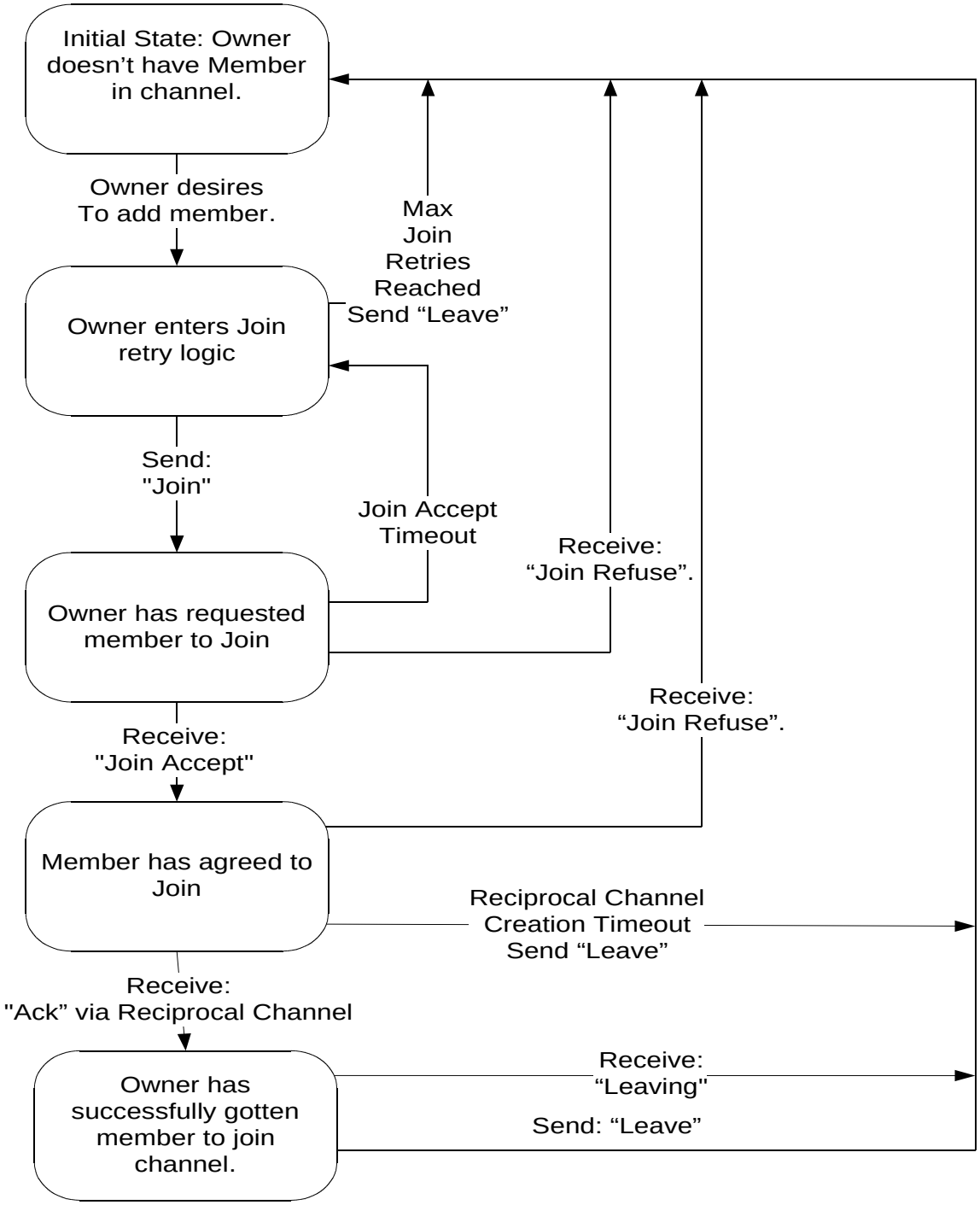


Figure 7: Component Channel Owner State Diagram

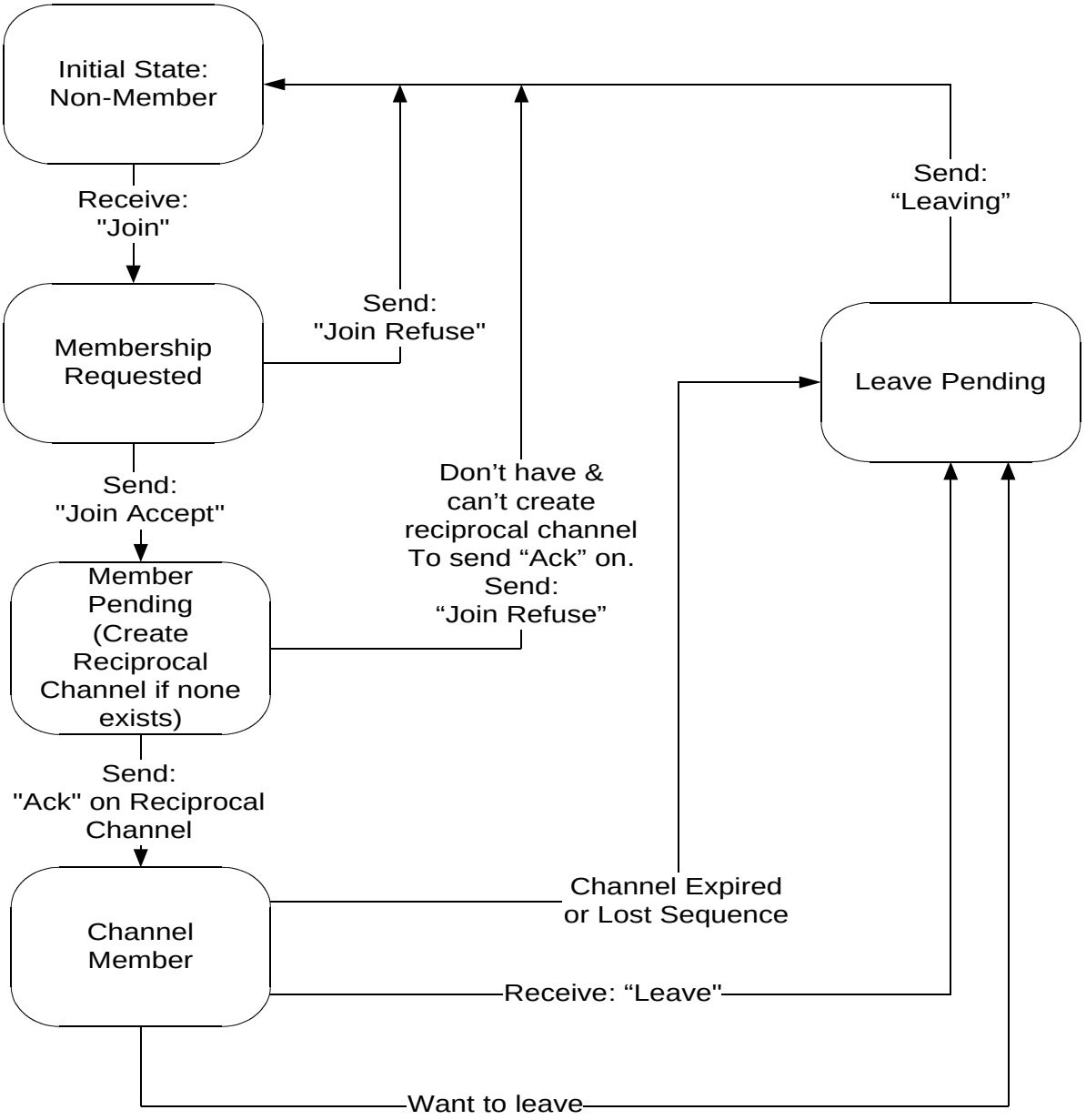


Figure 8: Component Channel Member State Diagram

3.5.2 Session Membership

At any time after a leader has created a session over a pair of reciprocal channels, other components may be connected to (become members of) the session, while existing members may disconnect from the session. The leader sends a *Connect* message to a potential member and the member responds with a *Connect Accept* message. A member may be disconnected from a session by either the leader sending *Disconnect* to the member or the member sending *Disconnecting* to the leader.

All members of a session are disconnected from the session if its downstream channel is terminated. Any member is disconnected from a session if its upstream channel is terminated. It is an error for components to be disconnected from a session by termination of the underlying channels. In normal operation members must be explicitly disconnected before the constituent channels may be closed.

A state diagram is shown in Figure 9 for a potential session connection component.

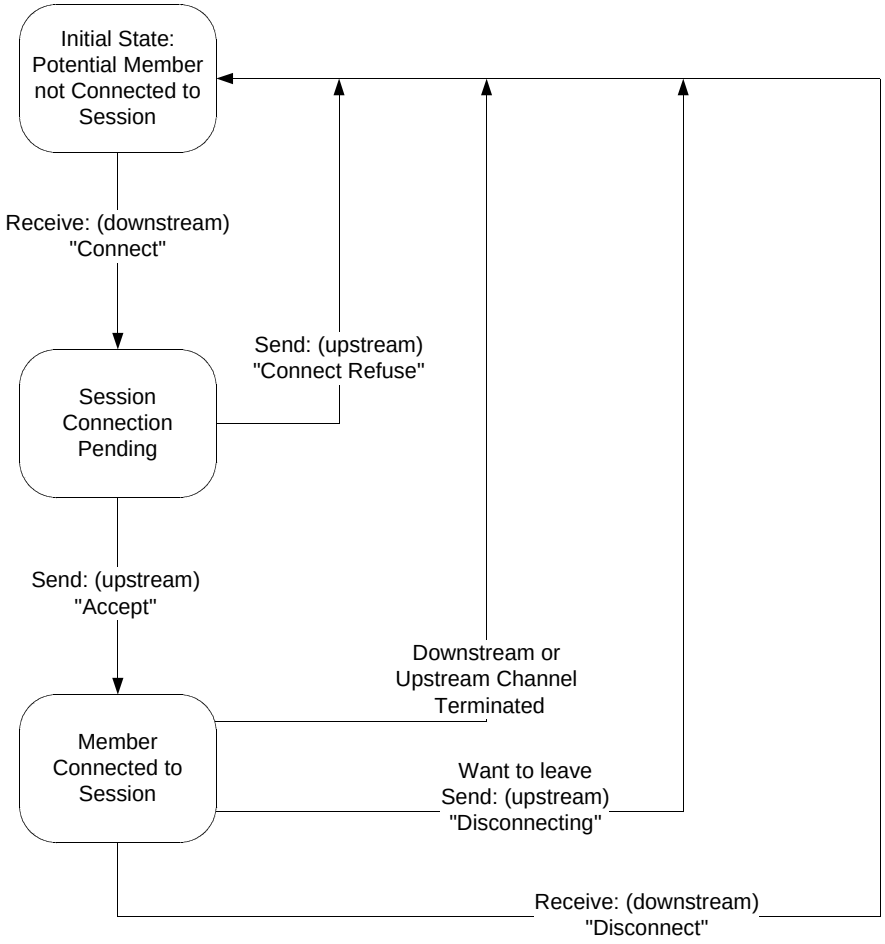


Figure 9: Potential Session Member State Diagram

3.5.3 Higher Layer Protocols

No higher layer protocol can be transported until a session has been started for that protocol. An SDT implementation can refuse a *Connect* by responding with *Connect Refuse*. It may do this for any protocol for which it has no high level client waiting, or for other reasons such as resource limitations or policy reasons. If a higher level PDU is received associated with a channel and protocol for which no session is started, the PDU shall be discarded.

3.6 Connection State of Channels and Sessions

An owner considers a sequenced channel *connected* to a particular member, so long as that member responds with *ACKs* as specified or until the membership is terminated by sending a *Leave* message to the member or until the membership is terminated by receiving a *Leaving* message from the member. It is considered connected by the members all the while there is traffic within the expiry time and sequence is not lost or until the membership is terminated by a *Leave or Leaving* message.

If the owner receives no *ACK* from a member after appropriate timeouts and retries then they shall send *Leave* and consider the member gone from that channel. Any reciprocal channel connections shall be removed when their reciprocal connection is lost.

If the member receives no traffic from the owner for the expiry time then they shall send *Leaving* to the owner and consider the channel closed. Any channel dependent upon that channel for its reciprocal channel shall also be closed.

A *session* is considered *connected* between both leader and member following the establishment of that session by the *Connect* and *Connect Accept* message exchange between them until it becomes *disconnected* by one of the following:

- it is explicitly ended by a *Disconnect* message.
- it is broken by its *downstream channel* [4]becoming disconnected, i.e., the session member no longer being a member of the downstream channel.
- it is broken by the *upstream channel* from the member to the leader becoming disconnected, i.e., the session leader no longer being a member of the upstream channel.

3.7 Session Disconnection and Channel Termination

When a leader wishes to terminate a session it shall ensure that the session has no connections by issuing *Disconnect* messages as necessary. *Disconnect* messages shall be sent reliably. When a member wishes to disconnect from a session it shall reliably send a *Disconnecting* message to the session leader.

When a channel owner wishes to terminate a channel it shall ensure there are no sessions on the channel as specified above and that the channel has no members by issuing reliable *Leave* messages as necessary.

It is an error to terminate a channel before disconnecting all sessions dependent upon it.

As stated earlier, a channel may also be terminated by timeout (expiry or lack of response). These are error conditions, and are not to be used intentionally to terminate a channel.

4 Messages

4.1 General

The definition and format of all PDUs and PDU blocks within SDT, including byte ordering and packing rules shall be as specified in this Standard and in [Arch].

SDT transports client protocol data within PDUs defined as "wrappers." Each wrapper may carry data for multiple client protocols and all such client protocols gain the benefits of message ordering, reliability, session management, and addressing. Several of SDT's own messages are carried in this way and are referred to in this specification as *SDT wrapped messages* or *SDT internal traffic.*

There are no SDT messages that may occur as both SDT base layer messages and SDT wrapped messages.

Each data block passed between SDT and the Root Layer Protocol shall contain a single PDU block composed of base layer SDT messages.

[4] The downstream channel is the channel owned by the session leader that carries downstream traffic for the session.

4.2 Sending and Receiving

In E1.17 systems, the underlying transport shall be the E1.17 Root Layer Protocol [Arch]. In the description of SDT messages references are made to sending messages, or replying to messages. Both these actions refer to taking one or more PDUs of the proper format, packing them into a PDU block and passing them to the Root Layer Protocol that then delivers the messages using some underlying transport (e.g., UDP) to its destination.

4.3 Addressing in SDT

In E1.17 every destination is a Component [Arch]. However, there is not a one-to-one relationship between components and underlying transport addresses (e.g., UDP address and port) and the Root Layer Protocol does not perform address filtering but simply sends to an underlying transport address. For this reason the PDU block might be delivered to other components as well as those intended. SDT has to anticipate this and must filter PDUs on reception by session identifier. Members must further filter individual PDUs (within wrappers) by MID (within each channel the MID identifies exactly one component).

4.4 SDT Base Layer Messages

SDT messages that occur in the PDU block directly transported by a lower layer transport protocol are referred to in this Standard as *SDT base layer messages*. The following SDT messages are base layer messages:

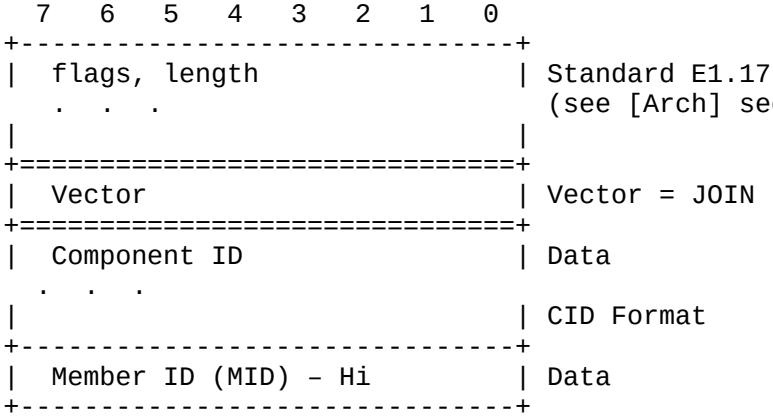
Join,
Join Accept,
Join Refuse,
Leaving,
NAK,
Reliable Wrapper,
Unreliable Wrapper,
Get Sessions, and
Sessions.

Of these messages, *Join, Get Sessions, and Sessions* are ad-hoc messages, that is, they are not sent within channels. These messages identify their intended recipient by CID. All other SDT messages shall be sent only within existing channels.

4.4.1 Join

A channel owner wishing a component to join a channel sends a *Join* message to the component to join. The *Join* specifies the channel parameters that the component is to use. *Join* is an ad-hoc message.

The packet containing the *Join* message shall originate from the source address of the sequenced channel to join and shall be addressed to the ad-hoc address advertised in [Discovery-IP].



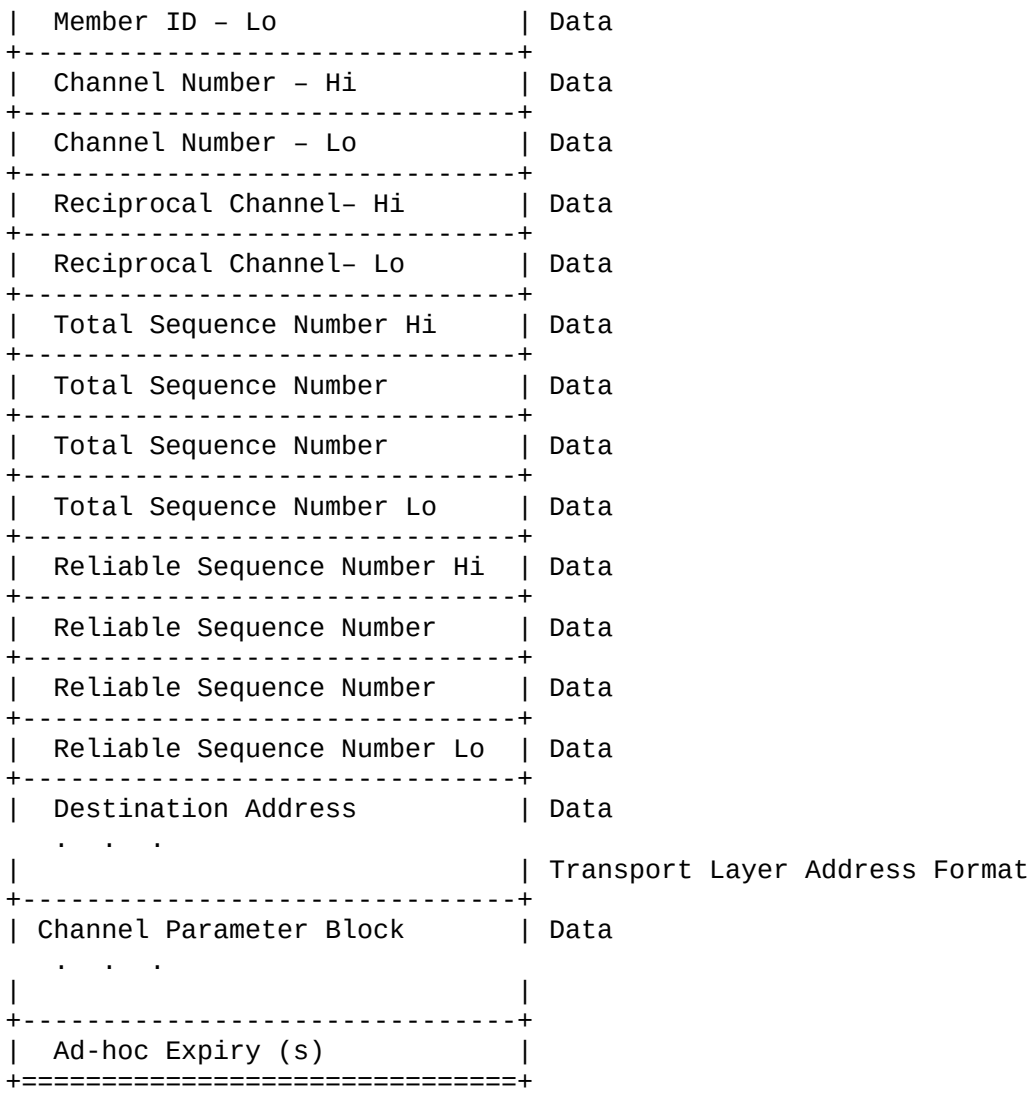


Figure 10: *Join* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Header:

No header fields.

Vector:

Vector = An octet containing the message type, JOIN.

Data:

Component ID = The CID of the component being asked to join the session.

Member ID = A Member ID, a MID, is the 16 bit identifier used to address all outbound SDT messages sent within a channel. This field contains the MID that the joining component will be addressed as within the channel being joined.

Channel Number = The channel number. See Section 3.3 for definition of Channel Number. The CID of the channel owner is found in the Root Layer Protocol PDU (see Section 2.6.1.2.2 of [Arch]). Channels shall not be assigned a Channel Number of 0 as this is reserved.

Reciprocal Channel = The reciprocal channel number that caused this *Join*. This field shall be 0 when a new channel pair is being initiated (i.e. no reciprocal channel exists).

Total Sequence Number = The total sequence number of the last wrapper PDU sent within the channel (see Section 5.2.1 for a definition of total sequence number).

Reliable Sequence Number = The reliable sequence number of the last reliable wrapper sent within the channel (see Section 5.2.2). Should the component join the channel, this field is used in determining where to start processing wrappers (see Section 3.5.1.2).

Destination Address = The transport layer specific address at which a channel owner communicates to a channel member or members (see Section 4.4.1.1). If the channel is unicast, the destination field shall contain SDT_ADDR_NULL and the transport protocol specific source address of the packet containing the corresponding *Join Accept* response shall be the destination address of the channel (the address at which the channel owner communicates to the channel member).

Channel Parameter Block = Fields as defined below in the Channel Parameter Block.

Ad-hoc Expiry = The number of seconds that the Ad-hoc Address, provided as the source address of the *Join* message, is valid.

Rules:

The *Join* message shall be sent from a valid Ad-hoc address for the component initiating the *Join.*

4.4.1.1 Transport Layer Address

The first octet of the Transport Layer Address specifies the address type. For any type of transport layer, the following shall indicate that no address is given: SDT_ADDR_NULL.

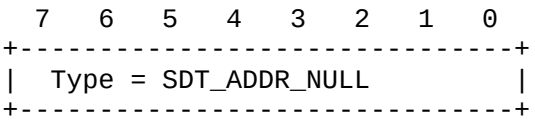


Figure 11: Null Transport Layer Address Format

Type = Type normally indicates the type of underlying address. In this case, type indicates that no address is supplied and says nothing about the actual type of underlying transport.

For example, IPv4 implementations use the following format for Transport Layer address format. For non-IPv4 transport layers or other IP specific information refer to the pertinent interoperability guide for the details of the transport layer address format.

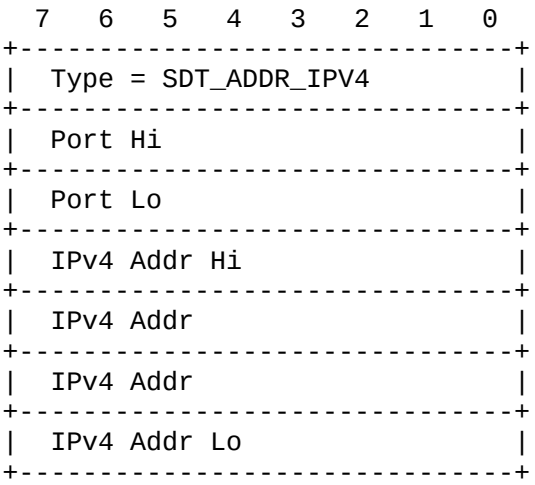


Figure 12: Example Transport Layer Address Format - IPv4

Type = indicates that this address is an IPv4 address.

Port = UDP port number, if the address is multicast this shall equal SDT_MULTICAST_PORT.
IPv4 Addr = IPv4 address (4 octets)

Additional address types may be added and shall be found in the pertinent interoperability profile.

4.4.1.2 Channel Parameter Block

The Channel Parameter Block is a block of fields found in the several message formats. It is defined here.

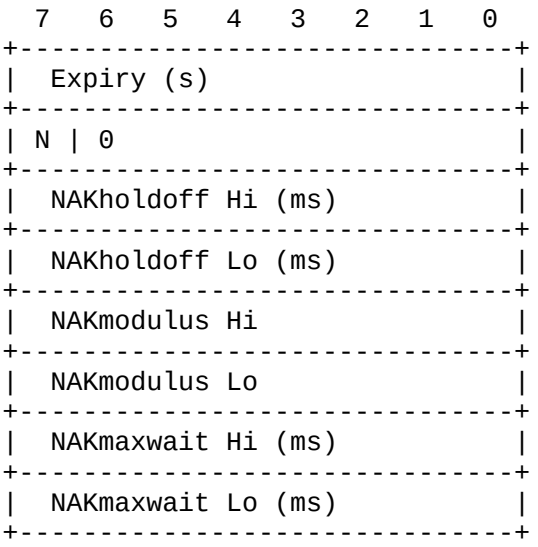


Figure 13: Channel Parameter Block Format

(s) = seconds
(ms) = milliseconds

Data:

Expiry = The number of seconds to tolerate no traffic from the channel owner before leaving the channel. The minimum value for Expiry shall be MIN_EXPIRY_TIME seconds (see Section 7.3).

N = N is the NAK_Outbound flag. If N is one then any future necessary NAKs shall be sent outbound on the channel. If N is zero then no NAK messages shall be sent to the destination address of the channel. See Section 5.7.1.

0 = 7 zero bits.

NAKholdoff (ms) = A parameter to be used in the calculation of a standoff time for sending NAKs. See Section 5.7.2.

NAKmodulus = A non-zero parameter to be used in the calculation of a standoff time for sending NAKs. See Section 5.7.2.

NAKmaxwait = The maximum number of milliseconds to wait before sending a NAK. See Section 5.7.2.

4.4.2 Join Accept

After a component has received a *Join* message and decides to join a channel, or has previously joined a channel, it shall reply to the owner with a *Join Accept* message. See Section 3.5.1.2.

The packet containing the *Join Accept* message may originate from any address, except in the case where the *Join* message specified SDT_ADDR_NULL for the channel's Destination Address. In that case it shall originate from the component's desired destination address for the channel. The packet containing the *Join Accept* message shall be addressed to source address of the channel it is joining.

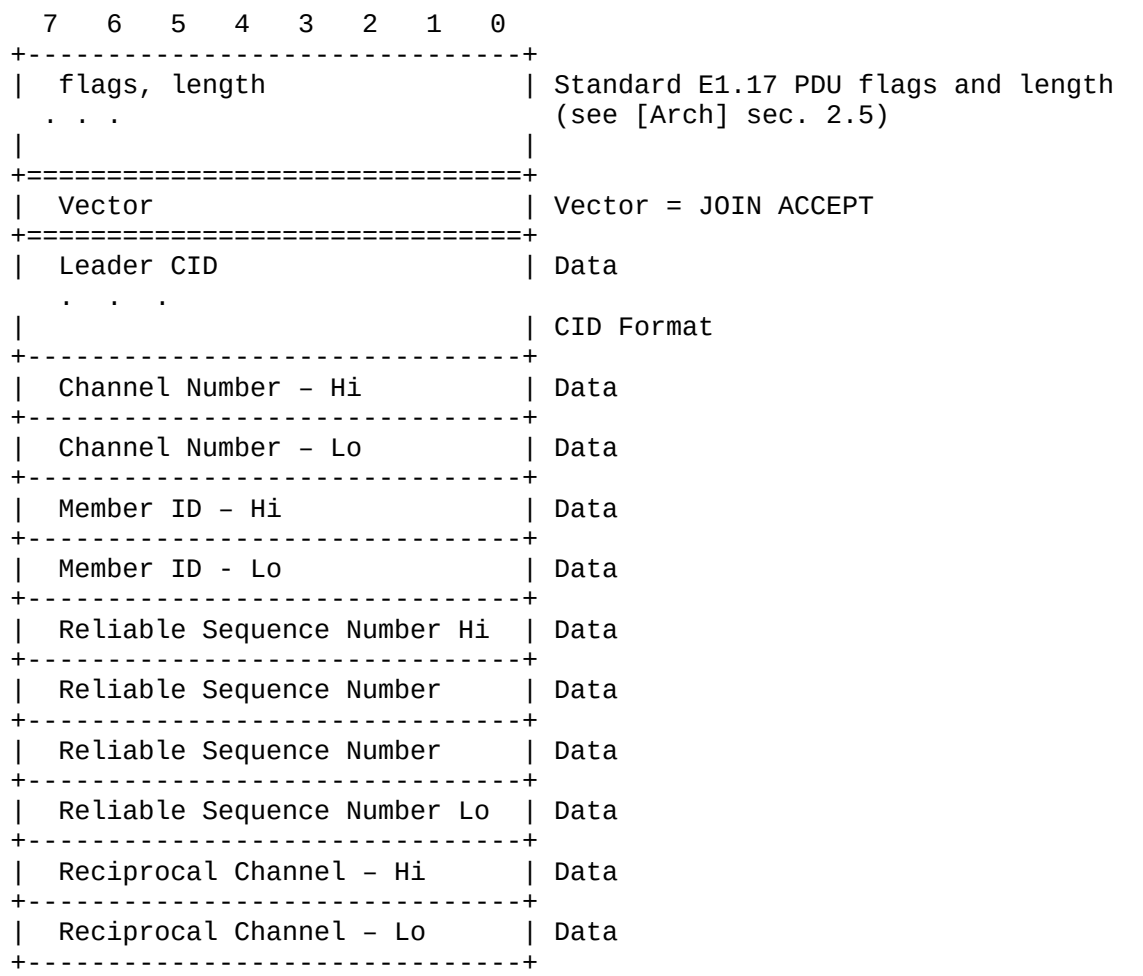


Figure 14: *Join Accept* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, JOIN ACCEPT.

Header:

No header fields.

Data:

Leader CID = The component ID of the component whose channel is being joined.

Channel Number. = The channel number assigned by the channel owner to the channel being joined.

Reciprocal Channel = The reciprocal channel number the new member will use to send reciprocal inbound messages for the channel being joined. This channel may already exist, or may be created after this message is sent.

Member ID = The 16 bit MID assigned to the joining component by the owner for the channel being joined.

Reliable Sequence Number = The last correctly sequenced reliable wrapper received, or in the case where no such wrappers have yet been received, this field shall contain the value of the Reliable Sequence Number provided in the *Join* message being responded to.

Rules:

In the case where the channel owner did not specify a destination address for the channel in the *Join* message (SDT_ADDR_NULL), the destination address for the channel shall be the transport specific source address of the packet carrying this message as received by the owner.

Join Accept messages shall be sent to the source address of the channel being joined.

4.4.3 Join Refuse

After a component has received a *Join* message and decides not to join a channel, it shall reply to the owner with *Join Refuse* message. The reason code shall indicate the reason the request was refused (e.g., insufficient resources, incompatible channel parameters).

The packet containing the *Join Refuse* message shall originate from the ad-hoc address of the component and shall have as its destination address the source address of the channel it is refusing to join.

```
  7   6   5   4   3   2   1   0
+-------------------------------+
|  flags, length                | Standard E1.17 PDU flags and length
  . . .                           (see [Arch] sec. 2.5)
|                               |
+===============================+
|  Vector                       | Vector = JOIN REFUSE
+===============================+
|  Leader CID                   | Data
   .  .  .
|                               | CID Format
+-------------------------------+
|  Channel Number –Hi           | Data
+-------------------------------+
|  Channel Number –Lo           | Data
+-------------------------------+
|  Member ID -Hi                | Data
+-------------------------------+
|  Member ID –Lo                | Data
+-------------------------------+
|  Reliable Sequence Number -Hi | Data
+-------------------------------+
|  Reliable Sequence Number     | Data
+-------------------------------+
|  Reliable Sequence Number     | Data
+-------------------------------+
|  Reliable Sequence Number -Lo | Data
+-------------------------------+
|  Refuse Code                  | Data
+-------------------------------+
```

Figure 15: *Join Refuse* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, JOIN REFUSE.

Header:

There are no header fields.

Data:

Leader CID = The component ID of the component whose channel is not being joined.

Channel No. = The channel number assigned by the channel owner to the channel not being joined.
Member ID = The MID assigned to the joining component by the owner of the channel not being joined.
Reliable Sequence Number = The value found in the Reliable Sequence Number field of the *Join* message that this message is responding to.
Refuse Code = The reason that the *Join* request is being refused.

Reason	Value	Definition
Nonspecific	1	Non-specific, non-SDT reason.
Illegal Parameters	2	Illegal channel parameters.
Low Resources	3	Insufficient resources.
Already Member	4	Multiple MIDs for single component.
Bad Address Type	5	Unrecognized transport layer address type.
No Reciprocal Channel	6	No upstream channel and can't create one. (for reciprocal inbound messages).
Only Unicast Supported	13	Only unicast channels are supported by this component.

Figure 16: *Join Refuse* Codes

4.4.4 Leaving

This is the last message sent upstream when a component leaves a channel. Having sent *Leaving,* the member shall not send or process any further messages within the channel, unless it joins the channel again (after receiving a *Join* message from the channel owner in the normal way).

The packet containing the *Leaving* message shall originate from any address and shall have as its destination address the source address of the channel it is leaving.

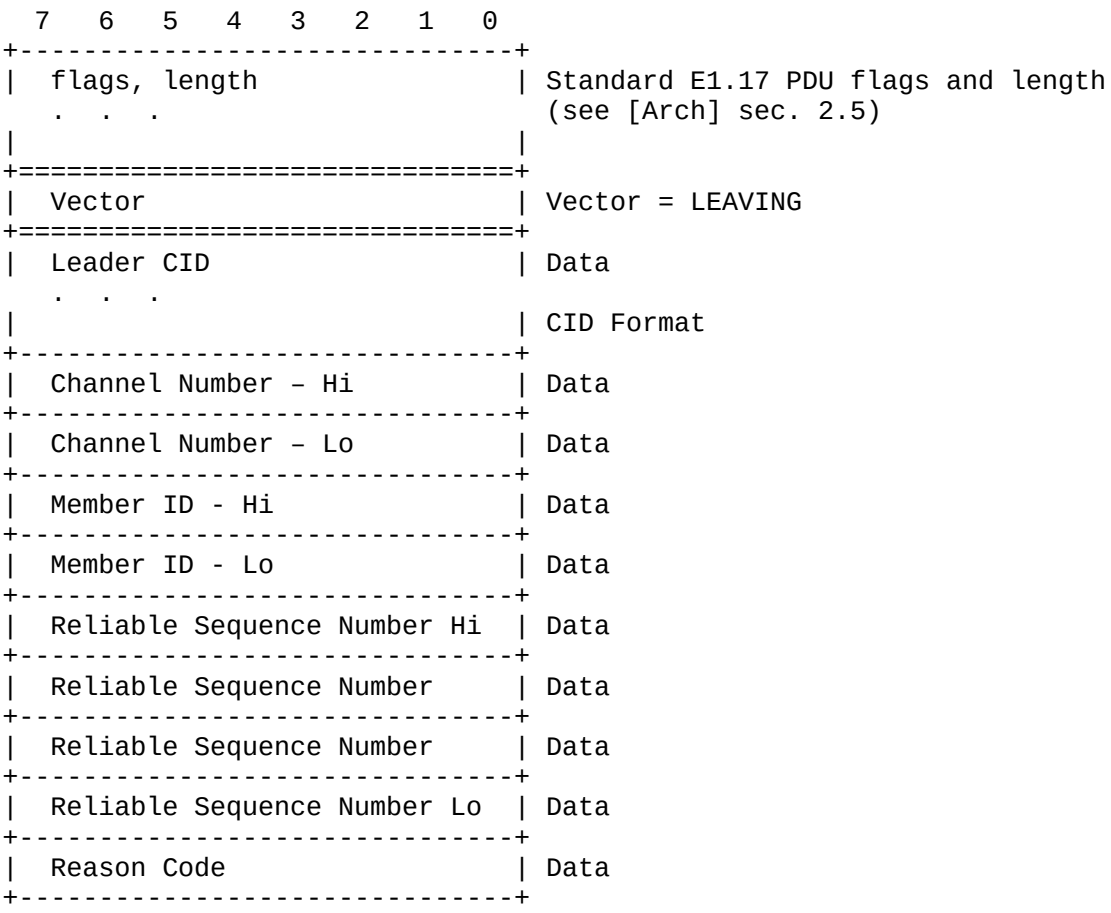


Figure 17: *Leaving* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, LEAVING.

Header:

There are no header fields.

Data:

Leader CID = The component ID of the component whose channel is being left.

Channel No. = The channel number assigned by the channel owner to the channel being left.

Member ID = The MID of the leaving component on the channel identified in the message.

Reliable Sequence Number = The reliable sequence number of the last correctly sequenced reliable wrapper received and processed on the channel identified in this message. If this message is in response to a *Leave* message, then this field shall have the value of the reliable sequence number on the wrapper that contained the *Leave*.

Reason Code = The reason the component is leaving the channel.

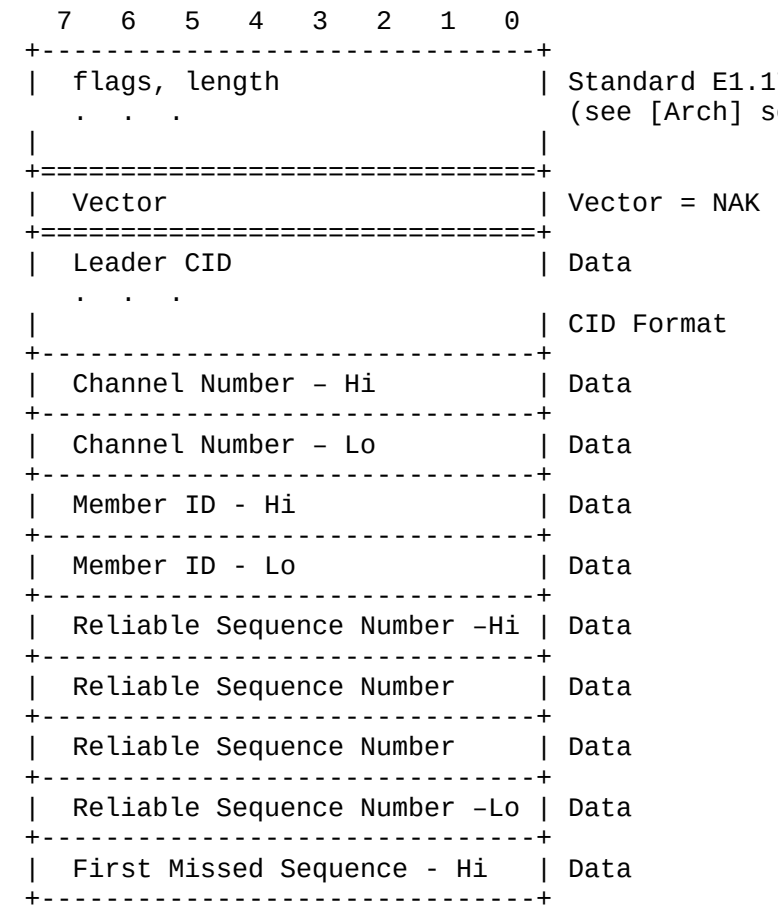
Reason	Value	Definition
Nonspecific	1	Non-specific, non-SDT reason.
No Reciprocal Channel	6	No upstream channel and can’t create one.
Channel Expired	7	Channel has expired.
Lost Sequence	8	Unrecoverable packets missed.
Saturated	9	Can’t keep up, processor saturation.
Transport Address Changing	10	Component changing transport layer address (e.g., IP number lease expired)
Asked to Leave	11	Asked to leave the channel.

Figure 18: *Leaving* Reason Codes

4.4.5 NAK

The mechanism for recovering missed reliable wrapper messages is to send a *NAK* (Negative Acknowledge) message to the leader requesting retransmission of any *Reliable Wrapper* messages that have been missed.

The packet containing the *NAK* message shall originate from any address and shall have as its destination address the source address of the channel. If the channel parameter NAK_Outbound is set to true, the NAK message shall also be sent to the destination address of the channel.



```
|  First Missed Sequence        | Data
+-------------------------------+
|  First Missed Sequence        | Data
+-------------------------------+
|  First Missed Sequence – Lo   | Data
+-------------------------------+
|  Last Missed Sequence - Hi    | Data
+-------------------------------+
|  Last Missed Sequence         | Data
+-------------------------------+
|  Last Missed Sequence         | Data
+-------------------------------+
|  Last Missed Sequence - Lo    | Data
+-------------------------------+
```

Figure 19 *NAK* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, NAK.

Header:

No header fields.

Data:

Leader CID = The component ID of the channel owner.

Channel No. = The channel number the NAK message applies to and is being sent on.

Member ID = The MID of the component reporting the missed wrappers.

Reliable Sequence Number = The ACK point of the component in this channel.

First Missed Sequence = The first reliable sequence number of the range to NAK.

Last Missed Sequence = The last reliable sequence number in the range of missed wrappers to identify as needing resending.

4.4.6 Reliable Wrapper and Unreliable Wrapper

An SDT wrapper message is a message whose purpose is to carry other messages. These other messages may be SDT protocol messages or messages of other protocols herein called client protocols. Wrapper messages are sequenced. A *Reliable Wrapper* message, with a vector of REL_WRAP, may be retransmitted if a channel member misses it and sends a NAK message to the channel owner. *Unreliable Wrapper* messages, with a vector of UNREL_WRAP, are not resent and cannot be negatively acknowledged.

The packet containing the *Wrapper* message shall originate from the source address of the channel and shall have as its destination address, the destination address of the channel.

```
  7   6   5   4   3   2   1   0
+-------------------------------+
|  flags, length                | Standard E1.17 PDU flags and length
   .  .  .                      |  (see [Arch] sec. 2.5)
|                               |
+===============================+
|  Vector                       | Vector = REL WRAP or UNREL WRAP
+===============================+
|  Channel Number –Hi           | Data
+-------------------------------+
|  Channel Number –Lo           | Data
+-------------------------------+
```

```
|  Total Sequence Number -Hi    | Data
+-------------------------------+
|  Total Sequence Number        | Data
+-------------------------------+
|  Total Sequence Number        | Data
+-------------------------------+
|  Total Sequence Number -Lo    | Data
+-------------------------------+
|  Reliable Sequence Number –Hi | Data
+-------------------------------+
|  Reliable Sequence Number     | Data
+-------------------------------+
|  Reliable Sequence Number     | Data
+-------------------------------+
|  Reliable Sequence Number –Lo | Data
+-------------------------------+
|  Oldest Available Wrapper –Hi | Data
+-------------------------------+
|  Oldest Available Wrapper     | Data
+-------------------------------+
|  Oldest Available Wrapper     | Data
+-------------------------------+
|  Oldest Available Wrapper –Lo | Data
+-------------------------------+
|  First Member to ACK - Hi     | Data
+-------------------------------+
|  First Member to ACK - Lo     | Data
+-------------------------------+
|  Last Member to ACK - Hi      | Data
+-------------------------------+
|  Last Member to ACK - Lo      | Data
+-------------------------------+
|  MAK Threshold - Hi           | Data
+-------------------------------+
|  MAK Threshold - Lo           | Data
+-------------------------------+
|  SDT Client Block             | Data
   .  .  .
|                               | See Section 4.4.7
+-------------------------------+
```

Figure 20: *Wrapper* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, REL WRAP or UNREL WRAP, indicating whether this PDU contains reliable or unreliable data.

Header:

No header fields.

Data:

Channel Number = The channel number assigned by the channel owner.

Total Sequence Number = The current total sequence number (see Section 5.2.1 for a definition of total sequence number).

Reliable Sequence Number = The current highest reliable sequence number sent on this channel (see Section 5.2.2).

Oldest Available Wrapper = The sequence number of the oldest reliable wrapper that the channel owner has available for resending.

First Member to ACK = The Member ID (MID) representing the small end of a numerical range of Member ID's that shall send ACK messages.

Last Member to ACK = The Member ID (MID) representing the large end of a numerical range of Member ID's that shall send ACK messages.

MAK Threshold = If the channel member has ACK'd within MAK Threshold reliable wrappers of this message, there is no need to ACK in response to this wrapper. If the ACK Threshold is 0, the channel member must always issue an ACK in response to receipt of this message.

SDT Client Block = This field is optional. Its presence or absence may be determined by examining the PDU length. See Section 4.4.7 for the definition of this PDU block.

Rules:

Wrapper messages shall only be sent outbound in channels.

MID value 0xffff (used for "all members") has no special meaning in MAK parameters. It does not mean all members ACK.

Recipients of this wrapper need not check First Member to ACK and Last Member to ACK for the special values 0 and 0xffff, but shall implement the following C algorithm (or any other algorithm that produces identical results) in which First Member to ACK, Last Member to ACK and Recipient Component's MID are all unsigned 16-bit numbers:

```
if ((First Member to ACK <= Recipient Component's MID) &&
        (Recipient Component's MID <= Last Member to ACK))
{
        // current component is subject to range and must send ACK, if the threshold value is exceeded;
}
```

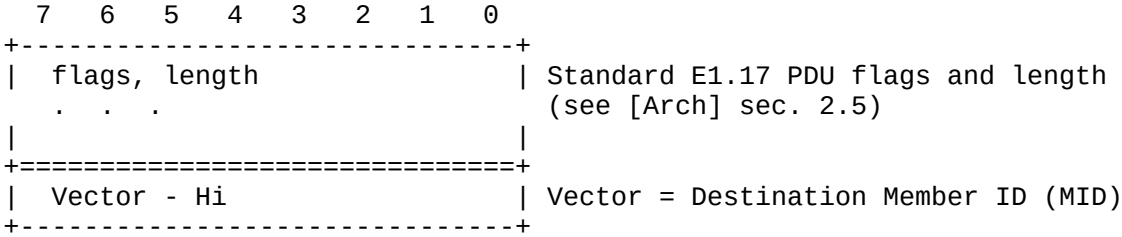
Channel owner shall not use MID 0 for First Member to ACK or Last Member to ACK.
Channel owner may use 0xffff so:

- range 1..0xffff means all members
- range 1..0xfffe means all members (0xfffe is largest legal MID)
- range X..Y means no members if X > Y
- range 0xffff..X means no members for any value of X (since no legal MID is >= 0xffff)

The preferred range to use for "no members" is 0xffff..0xffff since members using the above algorithm will only test the first MID value.

4.4.7 SDT Client Block

The SDT Client block is a PDU Block as defined in [Arch]. The PDUs within the SDT Client block shall all have the following format:



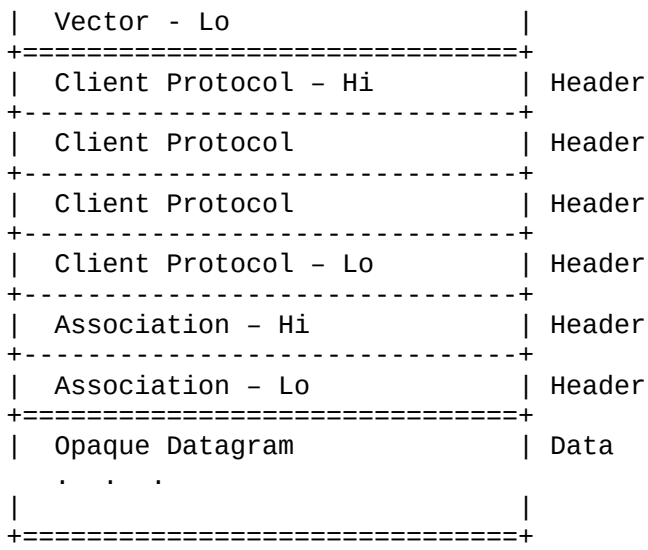


Figure 21: SDT Client Block PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = Destination Address = MID, indicating a single SDT channel member or join pending member (see Section 3.5.1.2) or all members (MID = 0xFFFF) to whom the contents of the data field are intended to be delivered.

Header:

Client Protocol = The protocol ID of the protocol that the opaque datagram is to be delivered to. For internal SDT traffic the value of this field is SDT_PROTOCOL_ID. This field, together with the Association field below and the CID of the sending component uniquely identifies the session that the datagram applies to.

Association = The value of this field indicates which channel this client block applies to. If it is 0, the datagram is outbound traffic on the current channel (it is session downstream traffic if the Client Protocol is not SDT). If it is non-zero, the datagram is inbound traffic for the channel whose channel number is the value of this field.

Data:

Opaque Datagram = For non-SDT protocols, SDT does not know the contents of this field. When the Client Protocol is SDT_PROTOCOL_ID, this field shall be interpreted as a PDU block composed only of certain SDT messages (SDT wrapped messages), see Section 4.5.

Rules:

The Opaque Datagram shall be discarded if the Client Protocol does not match a protocol ID for which a session has been connected.

If the Association field is non-zero and the Client Protocol is SDT_PROTOCOL_ID then the Opaque Datagram is Channel Reciprocal Inbound Traffic (*Ack, Connect Accept, Connect Refuse, or Disconnecting)* for the channel identified in the Association field.

If the Association field is non-zero and the Client Protocol is not SDT_PROTOCOL_ID then the Opaque Datagram is Session Upstream Traffic for the session identified by the Association field, and Client Protocol.

If the Association field is zero and the Client Protocol is not SDT_PROTOCOL_ID then the Opaque Datagram is Session Downstream Traffic for the session identified by the channel number the client block is sent on, and the Client Protocol.

If the Assocation field is zero and the Client Protocol is SDT_PROTOCOL_ID then the Opaque Datagram contains wrapped Channel Outbound Traffic (*Channel Params, Leave, Connect, or*

Disconnect). Note that other Channel Outbound Traffic consists of base layer, unwrapped messages.

4.4.8 Get Sessions

A component wishing to obtain diagnostic information about an E1.17 system may ask a component to report the list of sessions it has joined or is leading by sending it the *Get Sessions* message. This message and the response to it, *Sessions*, are provided to assist in intelligent troubleshooting of a system using SDT protocols and should not be used to build application functionality that requires complete or timely knowledge of session allocation and usage. *Get Sessions* is an ad-hoc message.

The packet containing the *Get Sessions* message shall originate from a valid address to which the reply must be sent. The packet shall be addressed to a Component's ad-hoc address advertised in [Discovery-IP].

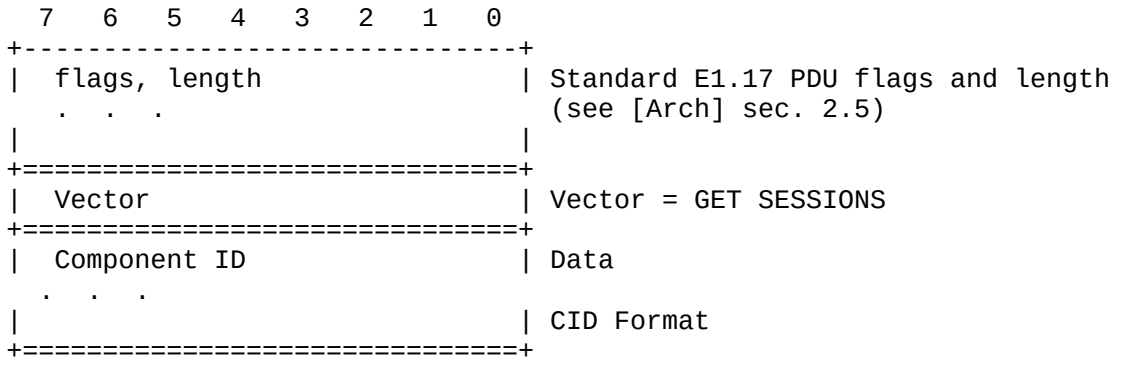


Figure 22 Get Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, GET SESSIONS.

Header:

No header fields.

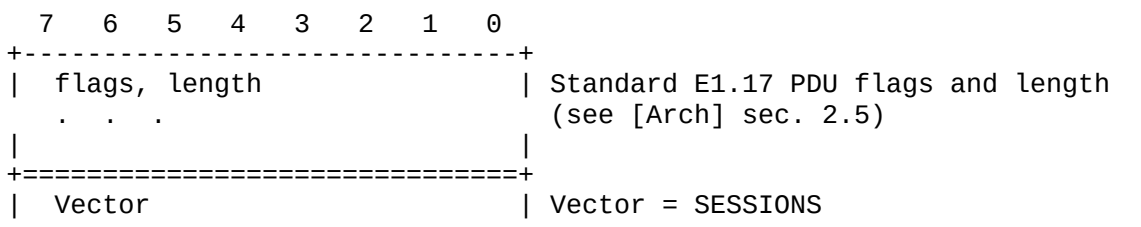
Data:

Component ID = The CID of the component being asked to list sessions it participates in.

4.4.9 Sessions

A component receiving a *Get Sessions* message shall respond with one or more *Sessions* messages to report on the sessions it is involved with. More than one Sessions message may be required to communicate a complete reply to a single *Get Sessions* message. *Sessions* is an ad-hoc message.

The packet containing the *Sessions* message shall originate from any valid address. The packet shall be addressed to the source address of the packet containing the *Get Sessions* message that this message is a reply to.



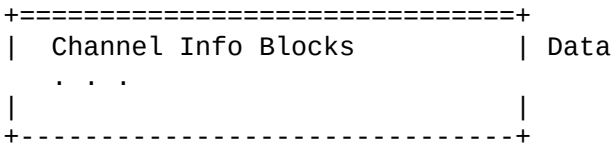


Figure 23 *Sessions* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SESSIONS.

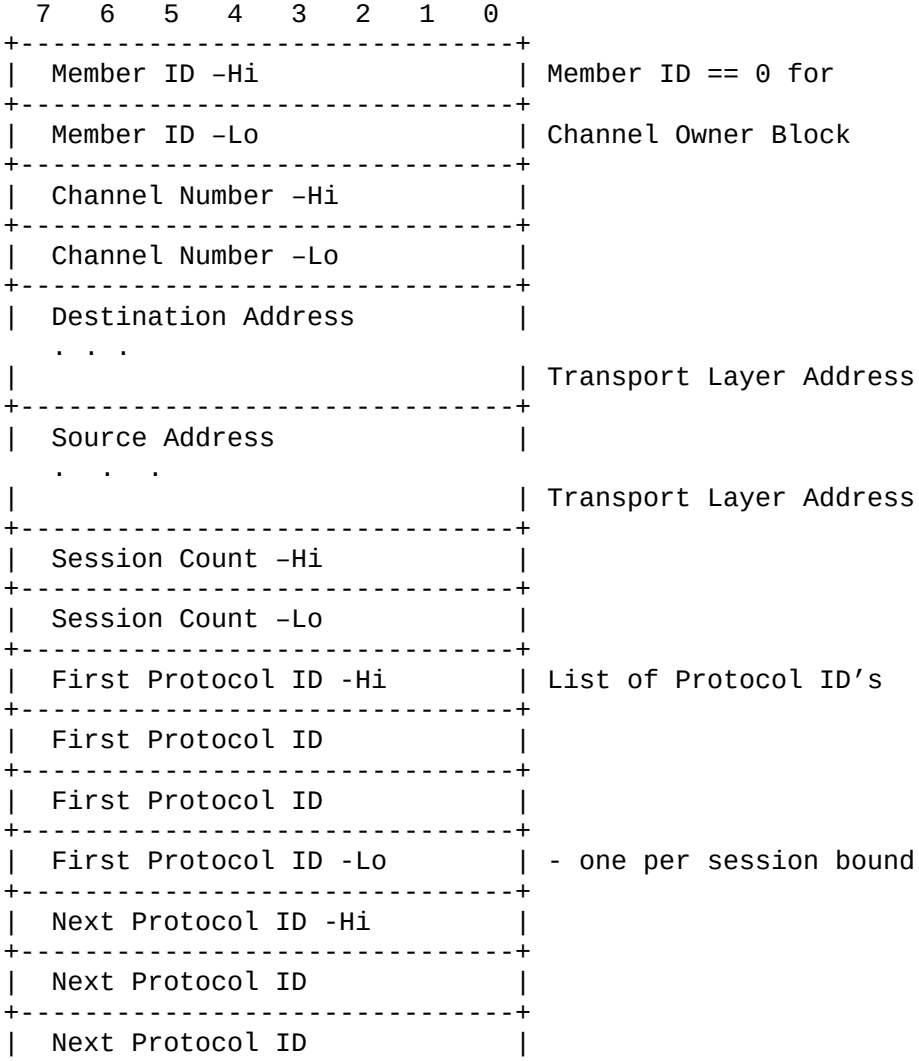
Header:

No header fields.

Data:

Channel Info Blocks= A series of Channel Owner and Channel Member Info Blocks describing the current channel and session state of the component.

4.2.8.1 Channel Owner Info Block



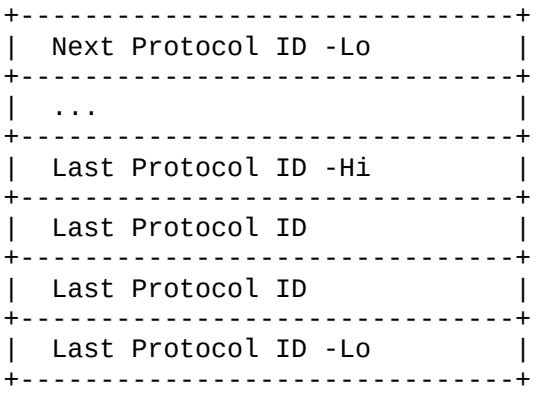
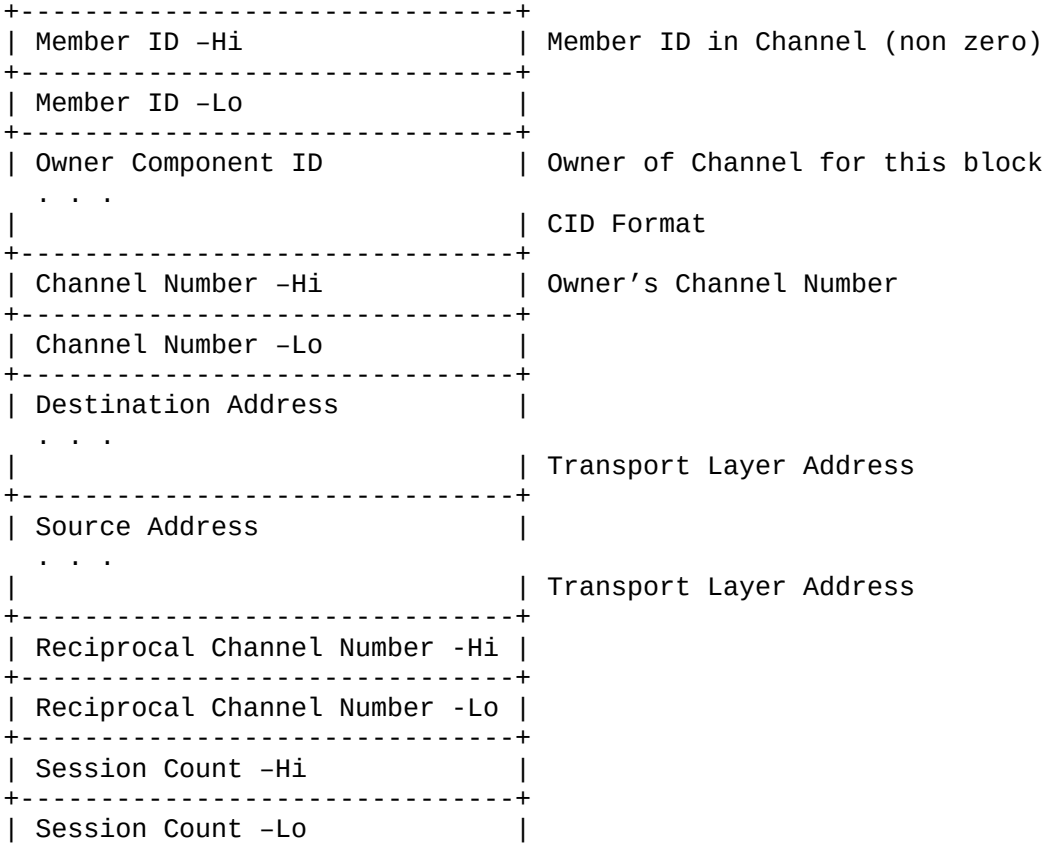


Figure 24 Channel Owner Channel Info Block Format

Member ID = 0 indicating that this is a channel owner info block.
Channel Number = The channel number assigned by the channel owner.
Destination Address = The transport specific address at which a channel owner communicates to a channel member or members. This field shall not contain SDT_ADDR_NULL.
Source Address = The transport specific address at which a channel member sends certain channel inbound messages to a channel owner. This field shall not contain SDT_ADDR_NULL.
Session Count = The number of sessions bound to the channel.
First Protocol ID, Next Protocol ID, …, Last Protocol ID = A list of protocol ID's, one for each session bound to the channel.

4.2.8.2 Channel Member Info Block



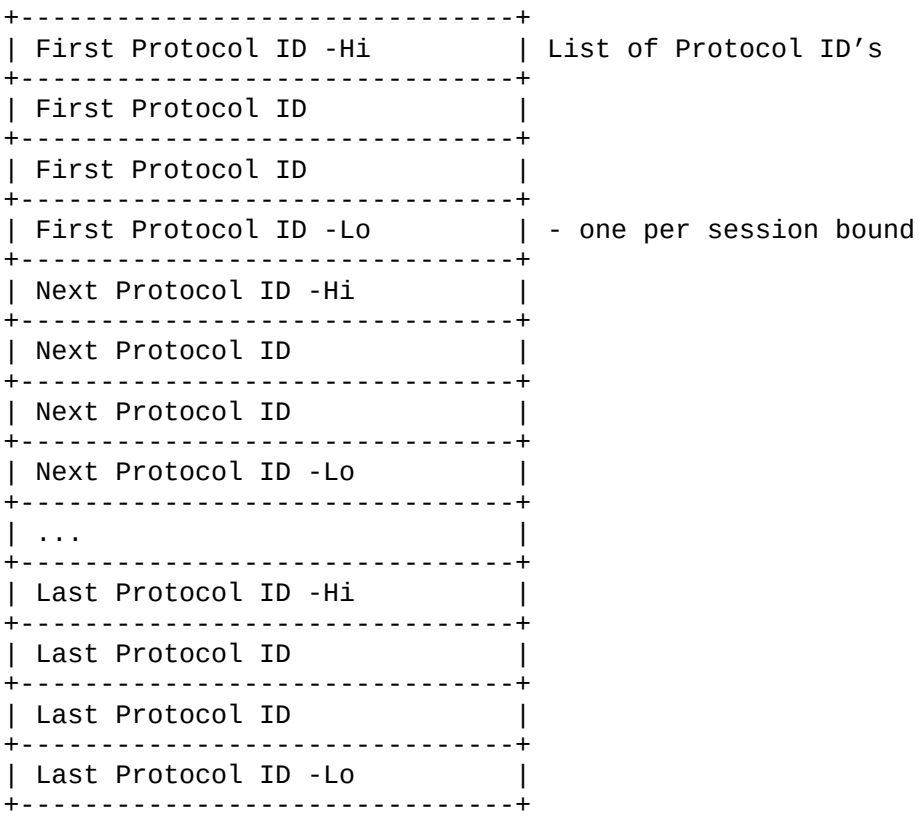


Figure 25: Channel Member Channel Info Block Format

Member ID = The non-zero Member ID indicates this is a channel member info block and the value of the field is the id of the member on the channel.

Owner Component ID = The CID of the component owning the channel that this info block is describing.

Channel Number = The channel number assigned by the channel owner.

Destination Address = The transport specific address at which a channel owner communicates to a channel member or members. This field shall not contain SDT_ADDR_NULL.

Source Address = The transport specific address at which a channel member sends certain channel inbound messages to a channel owner. This field shall not contain SDT_ADDR_NULL.

Session Count = The number of sessions bound to the channel.

First Protocol ID, Next Protocol ID, …, Last Protocol ID = A list of protocol ID's, one for each session bound to the channel.

Reciprocal Channel Number = The channel number of the channel that transports *SDT internal traffic*, e.g., ACK's, to the channel identified in the Channel Number field.

4.5 SDT Wrapped Messages

The contents of the Data field of an SDT Client Block PDU (the Opaque Datagram of 4.2.6.1) whose vector field is SDT_PROTOCOL_ID shall be a PDU Block containing *SDT wrapped message* PDUs only. The following SDT messages, and only the following messages, are defined to be *SDT wrapped messages*. The wrapper containing them provides channel context, sequencing and addressing. They shall be found exclusively inside of the SDT Client Block of SDT *Wrapper* messages:

Ack,
Channel Params,
Leave,
Connect,
Connect Accept,
Connect Refuse

Disconnect,
Disconnecting.

4.5.1 ACK

The *ACK* message is sent inbound to a channel owner by a member. It indicates the most recent *Reliable Wrapper* message that has been correctly received and processed in consecutive order. The first *ACK* sent from a member to the channel owner also indicates to the owner which reciprocal channel the member is using for SDT internal traffic for the channel.

A member shall send an *ACK* in response to the Must Acknowledge Parameter (MAK) fields in the wrapper messages subject to the criteria of 5.8.1.2.

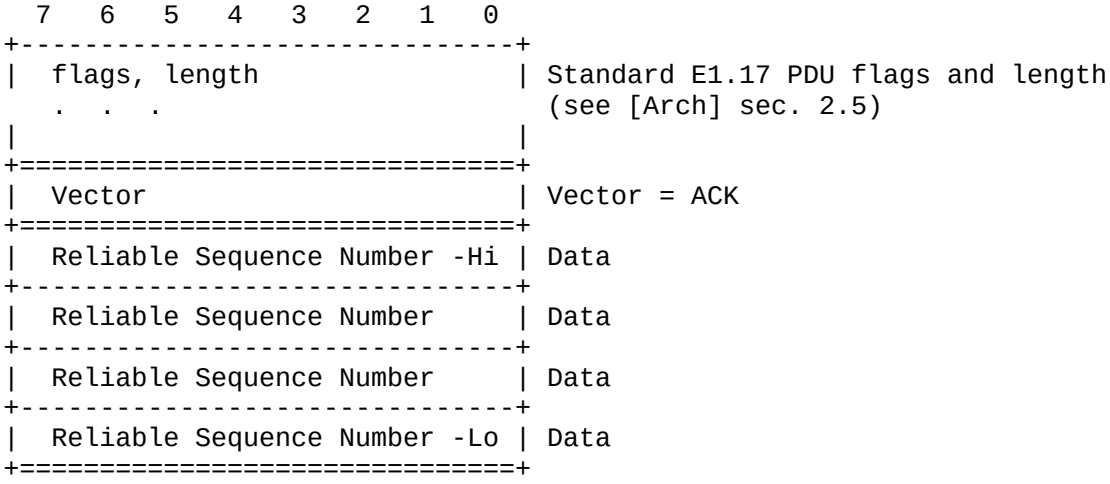


Figure 26: ACK Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, ACK.

Header:

No header fields.

Data:

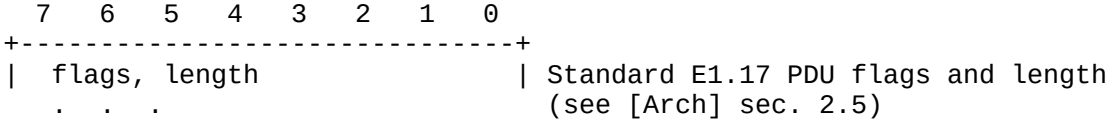
Reliable Sequence Number = The reliable sequence number of the last correctly sequenced reliable wrapper received in this channel. There shall be no reliable wrappers in this channel preceding the wrapper with this number that the component sending the ACK has not successfully received. This reliable sequence number is referred to as *the ACK point* of the component.

Rules:

When *ACK* messages are included with client protocol PDUs, for efficiency, they may be sent in reliable or unreliable wrappers, otherwise *ACKs* shall be sent in unreliable wrappers.

4.5.2 Channel Params

The *Channel Params* message is used to change all the channel parameters used by a particular member.



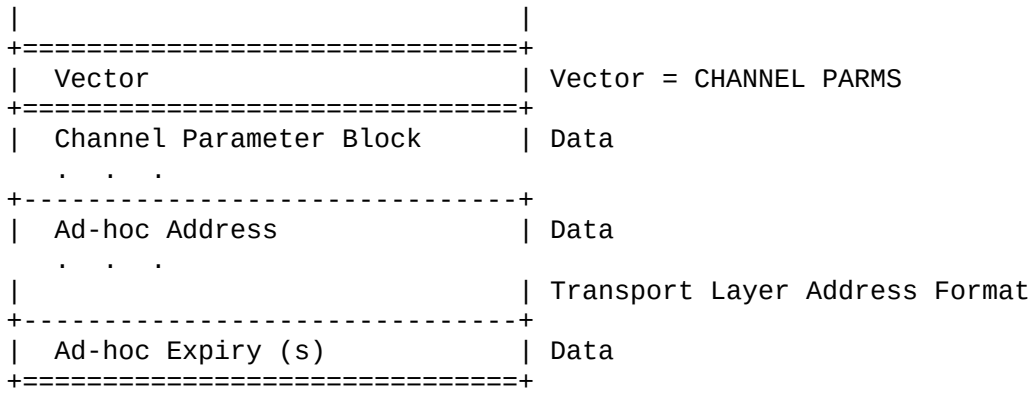


Figure 27 *Channel Params* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, CHANNEL PARAMS

Header:

No header fields.

Data:

Channel Parameter Block = Block of channel related parameters as defined in Section 4.2.1.2.

Ad-hoc Address = A transport layer specific address at which ad-hoc messages are accepted. If the ad-hoc address field contains SDT_ADDR_NULL then the source address of the packet containing this message (e.g., UDP source port and address) shall be used as the value for this field.

Ad-hoc Expiry = The number of seconds that the ad-hoc address, provided in the *Join* message, is valid.

Rules:

A component shall immediately begin using the new channel parameters or shall issue a *Leaving* message with reason *Non-specific* (see Section 4.4.4) and leave the channel.

This message shall be sent reliably.

4.5.2 Leave

The channel owner may send a *Leave* message to a member requiring it to leave the channel.

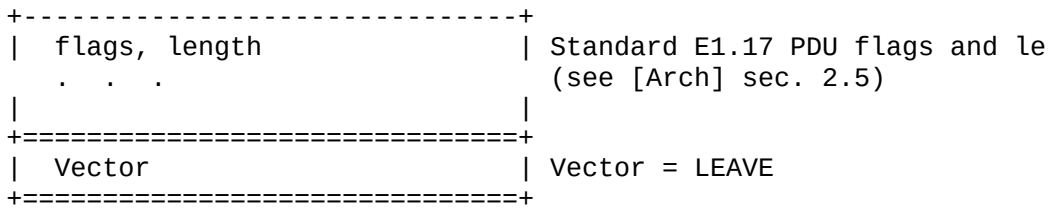


Figure 28 *Leave* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, LEAVE.

Header:

No header fields.

Data:

No data fields.

Rules:

This message shall be sent reliably. (See also Section 3.5.1.3.)

4.5.3 Connect

The channel owner may request that a member connect to a session by sending the *Connect* message to that member. The session, if connected, will carry the specified protocol. The session downstream traffic shall be on the channel carrying the *Connect*, while session upstream traffic shall be on a reciprocal channel owned by the member.

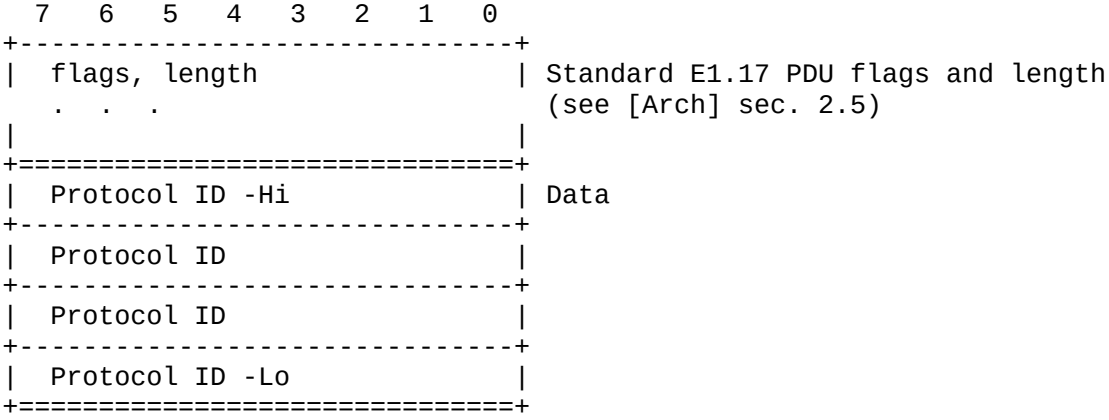


Figure 29: *Connect* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, CONNECT.

Header:

No header fields.

Data:

Protocol ID = The protocol ID of the protocol the session carries.

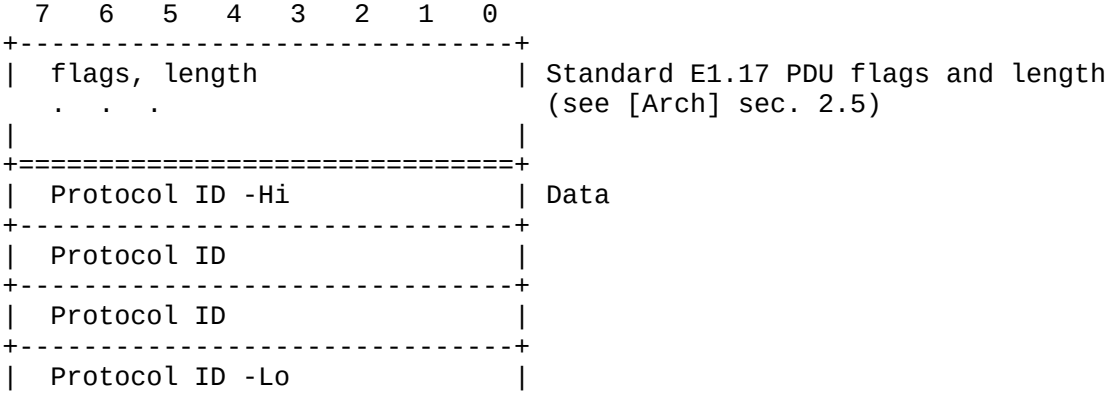
Rules:

Connect shall be sent reliably.

The Association field of the SDT Client Block containing *Connect* shall be 0.

4.5.4 Connect Accept

A channel member may send a *Connect Accept* message to the channel owner to accept a session connection request for the specified Protocol ID.



```
+================================+
```

Figure 30: *Connect Accept* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, CONNECT ACCEPT.

Header:

No header fields.

Data:

Protocol ID = The protocol ID of the protocol the session carries.

Rules:

Connect Accept shall be sent reliably.

The Association field of the SDT Client Block containing *Connect Accept* shall be the channel number of the reciprocal channel on which the *Connect* was received.

4.5.5 Connect Refuse

A channel member may send a *Connect Refuse* message to the channel owner to reject a session connection request.

```
  7   6   5   4   3   2   1   0
+--------------------------------+
|  flags, length                 | Standard E1.17 PDU flags and length
   .  .  .                         (see [Arch] sec. 2.5)
|                                |
+================================+
|  Protocol ID -Hi               | Data
+--------------------------------+
|  Protocol ID                   |
+--------------------------------+
|  Protocol ID                   |
+--------------------------------+
|  Protocol ID -Lo               |
+--------------------------------+
|  Reason Code                   | Data
+================================+
```

Figure 31: *Connect Refuse* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, CONNECT REFUSE.

Header:

No header fields.

Data:

Protocol ID = The protocol ID of the protocol the session carries.

Reason Code = The reason the member is refusing the session connection.

```
-------------------------------------------------------------------------
Reason       | Value | Definition
-------------------------------------------------------------------------
Nonspecific  |   1   | Non-specific, non-SDT reason.
```

```
Saturated     |   9   | Can’t keep up, processor
              |       |  saturation.
No Recipient  |   12  | Component does not support protocol ID.
-------------------------------------------------------------------
```

Figure 32: *Connect Refuse* Reason Codes

Rules:

Connect Refuse shall be sent reliably.

The Association field of the SDT Client Block containing *Connect Refuse* shall be the channel number of the reciprocal channel on which the *Connect* was received.

4.5.6 Disconnect

A channel owner may send a *Disconnect* message to a channel member to initiate disconnection of that member from a session on that channel.

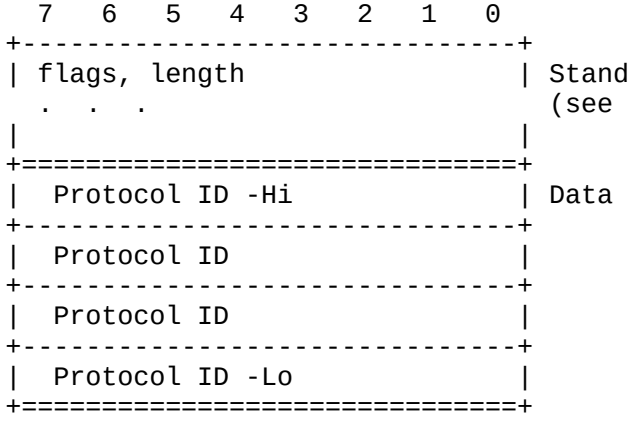


Figure 33: *Disconnect* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, DISCONNECT.

Header:

No header fields.

Data:

Protocol ID = The protocol ID of the protocol the session to be disconnected carries.

Rules:

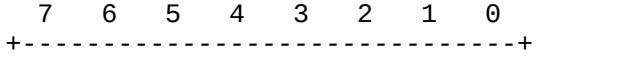
This message shall be sent reliably.

Any session messages received after this message has been sent shall be discarded.

The Association field of the SDT Client Block containing *Disconnect* shall be 0, i.e., this message is sent as an outbound message on the session’s downstream channel.

4.5.7 Disconnecting

A channel member may send a *Disconnecting* message to a channel owner to initiate disconnection of that member from a session on the owner’s channel.



```
| flags, length                 | Standard E1.17 PDU flags and length
  .  .  .                       (see [Arch] sec. 2.5)
|                               |
+===============================+
| Protocol ID -Hi               | Data
+-------------------------------+
| Protocol ID                   |
+-------------------------------+
| Protocol ID                   |
+-------------------------------+
| Protocol ID -Lo               |
+-------------------------------+
| Reason Code                   | Data
+===============================+
```

Figure 34: *Disconnecting* Message PDU Format

Standard PDU Fields:

Flags = LVHD indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the standard 12 bit or the extended 20 bit length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, DISCONNECTING.

Header:

No header fields.

Data:

Protocol ID = The protocol ID of the protocol the session to be disconnected carries.

Reason Code = The reason the component is leaving the channel.

Reason	Value	Definition
Nonspecific	1	Non-specific, non-SDT reason.
Channel Expired	7	Channel has expired.
Lost Sequence	8	Unrecoverable packets missed.
Saturated	9	Can’t keep up, processor saturation.
Transport Address Changing	10	Component changing transport layer address (e.g., IP number lease expired).

Figure 35: Disconnecting Reason Codes

Rules:

This message shall be sent reliably.

Any session messages received after this message has been sent shall be discarded.

The Association field of the SDT Client Block containing *Disconnecting* shall be the channel number of the reciprocal channel on which the *Connect* was received.

5 Reliability

Sessions obtain their reliability by using channels for their communication.

There are several distinct elements to reliability within the SDT channel:

Outbound Reliability Outbound reliability ensures that channel members will detect missing messages from the owner and request re-transmission (if the owner has buffered those messages so they are available for retransmission).

Online Status Online status ensures that the owner knows that the members are all present and receiving data.

Ordering Use of sequence numbers ensures that all members of a channel process messages in the order intended by the leader.

Outbound reliability, online status, and ordering are all achieved by a combination of numbering (sequencing) each outbound message, with members positively acknowledging that reliable messages have been received at the request of the leader and negatively acknowledging when they detect a break in the sequence indicating that they have missed reliable messages.

Channels are unidirectional and do not provide communications transportation for message responses. Responses within a session are sent on a separate reciprocal channel managed by the responding component.

5.1 Wrapping Messages

SDT's two wrapper messages (*Reliable Wrapper* and *Unreliable Wrapper*) are used to transport all client protocol data (and also *SDT wrapped messages*).

5.1.1 Difference between Reliable and Unreliable Wrappers

An unreliable wrapper is ordered but is only transmitted once by the owner. There is no mechanism to request retransmission. A reliable wrapper is kept (buffered) for some period after first transmission. It may be retransmitted during that period, if requested by a member.

5.1.2 Mixing Reliable and Unreliable Outbound Messages

The provision of both *Reliable Wrapper* and *Unreliable Wrapper* messages for outbound messages allows free intermixing of reliable and unreliable messages within the channel, at the choice of the channel, on a message by message basis by wrapping the messages in the desired wrapper message. For example, where the owner of a channel is transmitting a stream of rapidly changing updated values of the same data there is little use for reliability because as each new value is received, the old value is no longer relevant and there is no point in retransmitting it. However, if the last value in the stream is transmitted reliably, then the owner will be sure that all members have arrived at the correct final state. SDT provides for both reliable and unreliable outbound messages to be freely interleaved within a channel such that their ordering is preserved.

5.2 Sequencing Mechanism

Both of SDT’s wrappers for outbound traffic (*Reliable Wrapper* and *Unreliable Wrapper*) contain both a *reliable sequence number* and a *total sequence number*.

Examination of these two sequence numbers allows ordering of all messages and detection of missing or mis-ordered messages (a mis-ordered message is one whose sequence number precedes that of the last message processed, e.g., a retransmission). Note that because sequence numbers roll over when reaching maximum, the recent history of sequence numbers is required to determine if a sequence number of lesser value is before or after a current sequence number of greater value.

In conjunction with other channel parameters described below, these numbers also allow each member to determine which wrappers have been buffered by the owner and are available for retransmission.

Each member shall keep a record of both sequence numbers of the most recently correctly received and processed[5] wrapper. These stored sequence numbers shall be updated with the new values taken from each wrapper that is processed.

5.2.1 Total Sequence Number

The total sequence number is an unsigned 32 bit number. It is incremented by 1 for every wrapper (*Reliable Wrapper* or *Unreliable Wrapper*) within the channel. When the total sequence number increments past maximum (0xFFFFFFFF) it wraps back to 0.

5.2.2 Reliable Sequence Number

The reliable sequence number is an unsigned 32 bit number. It is incremented by 1 for every *Reliable Wrapper* sent (outbound) within the channel. When the reliable sequence number increments past maximum (0xFFFFFFFF) it wraps back to 0.

5.2.3 Correctly Sequenced Wrappers

On receipt of a new wrapper of either type, if the total sequence number is the immediate successor to the stored previous value, then this wrapper is correctly sequenced and shall be processed as soon as possible.

The member's record of the previous values of both reliable and unreliable sequence numbers shall be updated after the wrapper has been identified as ready for processing but before it is processed. Thus if any of the embedded PDUs refer to these values, the ones from the current wrapper shall be used.

Care shall be taken in implementing algorithms for checking sequence numbers to ensure that tests perform correctly when the sequence numbers wrap around from Maximum to Zero. The terms *precede* and *succeed* used in relation to sequence numbers shall be defined as follows:

Using 32 bit two's complement signed arithmetic, for two sequence numbers *A* and *B*.[6]
A precedes *B* if $A - B < 0$
A equals *B* if $A - B = 0$
A succeeds *B* if $A - B > 0$
A is the immediate successor of *B* if $A - B = 1$

5.2.4 Retransmitted or Mis-ordered Wrappers

On receipt of a new wrapper of either type, if the total sequence number precedes or is equal to the stored previous value, then this wrapper is a retransmitted or mis-ordered one and shall be discarded. This ensures that wrappers are processed only once and are processed in the order intended by the channel.

5.2.5 Missing Wrappers

If the total sequence number succeeds the stored previous value by more than one, then one or more wrappers have been missed. In this case the member shall now detect whether any of the missed wrappers were reliable by examination of the reliable sequence number and the wrapper type as follows:

5.2.5.1 If the newly received wrapper is *Reliable Wrapper* and the Reliable Sequence Number is the immediate successor to the stored previous value, then all missing wrappers are unreliable and cannot be recovered, so process this wrapper and update sequence numbers (see Section 5.3).

---

[5]Processing a wrapper means unpacking and interpreting the embedded PDUs. Within this Standard the term is understood to include wrappers that have been accepted and passed on for processing even if they have not yet been fully processed (e.g. because of multitasking or queuing implementations).

[6] The sequence numbers are unsigned quantities. For the purposes of the comparisons outlined here, they are interpreted as signed quantities. This appears incorrect, but in fact yields the most efficient implementation.

5.2.5.2 If the newly received wrapper is *Reliable Wrapper* and the Reliable Sequence Number succeeds the stored previous value by more than one, then reliable wrappers have been missed, so possibly NAK the missing sequences.

5.2.5.3 If the newly received wrapper is *Reliable Wrapper* and the Reliable Sequence Number precedes or is equal to the stored previous value, then a sequencing error has occurred, so discard this wrapper.

5.2.5.4 If the newly received wrapper is *Unreliable Wrapper* and the Reliable Sequence Number is equal to the stored previous value, then all missing wrappers are unreliable and cannot be recovered, process this wrapper and update sequence numbers (see Section 5.3).

5.2.5.5 If the newly received wrapper is *Unreliable Wrapper* and the Reliable Sequence Number succeeds the stored previous value, then reliable wrappers have been missed, so possibly NAK the missing sequences.

5.2.5.6 If the newly received wrapper is *Unreliable Wrapper* and the Reliable Sequence Number precedes the stored previous value, then a sequencing error has occurred, so discard this wrapper.

5.2.6 Sequencing Errors

If a sequencing error occurs as defined in Section 5.2.5 either the data has become corrupted or the channel has formed the wrapper incorrectly. The wrapper is discarded with no further processing – if the condition persists, then the channel will expire.

5.3 Missing Unreliable Wrapper Messages

Unreliable Wrapper messages cannot be recovered. If *all* missing wrappers are *Unreliable Wrapper's*, the member shall continue processing with the newly received wrapper, updating its sequence records accordingly. If missing *Unreliable Wrapper* messages subsequently arrive having been somehow delayed, the rules of 5.2.4 ensure that they are discarded rather than being processed out of sequence.

Members may take additional internal action if required based on the knowledge that messages have been missed (e.g., logging the errors) but any such action is outside the scope of this Standard and is not visible at the protocol level.

5.4 Missing Reliable Wrapper Messages

If *any* of the missing wrappers are *Reliable Wrapper* messages, then the member shall assess whether these are recoverable or not.

The following sections specify the mechanisms for assessing recoverability, recovering missing *Reliable Wrapper* messages and actions to take if messages are not recoverable.

5.5 Unavailable Wrappers

Each wrapper has a field *Oldest Available Wrapper* that indicates the sequence number of oldest *Reliable Wrapper* message that is still being buffered by the leader and is available for retransmission. The owner shall buffer all *Reliable Wrapper* messages including this one, and up to and including the current one. *Reliable Wrapper* messages that precede the *Oldest Available Wrapper* are no longer available and shall not be retransmitted.

The *Oldest Available Wrapper* may change from wrapper to wrapper, and even on wrapper retransmission due to *NAK* messages. The most up-to-date value available shall always be examined before attempting to recover missing wrappers. If the *Oldest Available Wrapper* is the same as the current wrapper, then no previous wrappers are available.

5.6 Unrecoverable Reliable Wrappers – Lost Sequence

If one or more missed *Reliable Wrapper* messages cannot be recovered because the channel no longer buffers them, the member has lost the sequence. The member shall issue *Leaving* message with a reason of Lost Sequence.

5.7 Recovery of Missing Reliable Wrappers – NAKs

The mechanism for recovering missed wrappers is to send a *NAK* (Negative Acknowledge) message to the channel requesting retransmission of any *Reliable Wrapper's* that have been missed. However in a large channel, if a wrapper is lost on transmission, and all members were to try to *NAK* at once, the network could be overloaded by a flood of *NAK* messages (this is called a *NAK* implosion). To avoid this undesirable situation, SDT uses *NAK* suppression: if one member sees another member *NAK* the missing wrapper, then it need not send a *NAK* of its own.

5.7.1 NAK Flag

Each member has a boolean parameter for the channel related to *NAK* processing:
NAK_Outbound indicates if *NAK* messages shall be sent outbound to the members.
This flag is used to enable or disable *NAK* suppression. Guidance on how the owner shall set the *NAK* flag is given in 5.10.1.

5.7.2 NAK Algorithm – Member

Having detected that one or more *Reliable Wrapper's* are missed and are recoverable the member starts *NAK* processing.

5.7.2.1 While awaiting retransmission of missed *Reliable Wrapper's*, the current wrapper and any subsequently received may be saved for later processing. (These wrapper messages should be saved if possible in the interests of system performance.) They shall not be processed at this time.

5.7.2.2 A member with scarce resources may discard some or all of these wrappers instead of saving them but this causes extra network traffic and is discouraged. Any *Reliable Wrapper* messages that are discarded shall be included in the *NAK* range for retransmission.

5.7.2.3 Before sending a NAK, the member waits for a standoff time given by the algorithm
T = min (NAKmaxwait, ((seqNo + MID) % NAKmodulus) * NAKholdoff),
where % means modulo arithmetic, NAKmaxwait is the maximum number of milliseconds to wait before sending, NAKmodulus sets a range for the standoff time, NAKholdoff is the scaling factor for the number of milliseconds to wait, and seqNo is the last reliable sequence number correctly received (as sent in the NAK PDU). If it is desired that no more than one NAK occur per NAKholdoff time, NAKmodulus should be larger than the highest MID. The NAKModulus shall not be zero. All three variables are sent in the *Join* message, and can be modified by the *Channel Params* message. See Table 4 for recommended values for NAKmaxwait (NAK_MAX_TIME), and NAKholdoff (NAK_HOLDOFF_INTERVAL).

5.7.2.3.1 If during this standoff time the member receives the missing wrappers then the *NAK* shall not be sent. The member shall resume normal operation, processing the messages in strict sequence.

5.7.2.3.2 If during this standoff time the member receives a *NAK* from another member that includes the same sequence numbers it is missing, then the *NAK* shall not be sent. The member then, at the end of its standoff time, continues *NAK* processing from step 5.7.2.4 as though it had sent the *NAK* itself.

5.7.2.3.3 If during this standoff time new wrappers are received, they shall be examined for the latest state of the *Oldest Available Wrapper* value and if some missing wrappers are no longer available for retransmission, the member shall cease *NAK* processing and commence leaving the channel as specified in Section 5.6.

5.7.2.4 At the end of the standoff time the member transmits a *NAK* indicating some or all of the *Reliable Wrapper* messages it needs re-transmitted. The range of *Reliable Wrapper* messages missing may change during *NAK* processing owing to new or retransmitted wrappers arriving during the various timeouts. The *NAK* is sent to the channel source address. Subsequently, if indicated by the NAK_Outbound channel parameter flag, a *NAK* is also sent to the channel destination address.

5.7.2.5 Having sent a *NAK* (or detecting that another member has sent the *NAK*), a member waits for NAK_TIMEOUT for the missing *Reliable Wrapper* message(s) to be resent.

5.7.2.5.1 During this waiting time if the member receives some or all of the missing *Reliable Wrapper* messages then the member shall process any that follow the last processed wrapper in strict sequence. Once all missing *Reliable Wrapper* messages have been processed, the member resumes normal processing starting with all wrappers that had been saved pending retransmission of missing ones – again in strict sequence.

5.7.2.5.2 If during this waiting time new wrappers are received, they shall be examined for the latest state of the *Oldest Available Wrapper* value and if some missing wrappers are no longer available for retransmission, the member shall cease *NAK* processing and commence leaving the channel as specified in Section 5.6.

5.7.2.6 If at the end of this waiting time there are still wrappers missing and NAK_ MAX_RETRIES hasn't been reached, the member resumes from step 5.7.2.2 and potentially resends its *NAK*. This resending is subject to the same suppression rules and is modified to remove any of the missing message(s) that have been received. When NAK_MAX_RETRIES is reached and there are still wrappers missing the component shall leave the channel with reason *Lost Sequence*.

5.7.3 NAK Algorithm – Owner

In normal operation, the owner continues to transmit sequenced wrappers on the assumption that they are correctly received until a *NAK* is received for one or more wrappers.

5.7.3.1 On receipt of a *NAK* the owner shall retransmit the requested wrappers if they are available. If multiple wrappers are re-transmitted this shall be done in strict sequence. Negatively acknowledged wrappers shall be re-transmitted before any further wrappers in the sequence.

5.7.3.2 If some requested *Reliable Wrapper* messages have been discarded, the owner may transmit those that are still buffered or may choose not to retransmit any of them – on the assumption that the member is already in a state of lost sequence.

5.7.3.3 If not retransmitting those wrappers that are still available, the owner shall immediately transmit at least one wrapper to ensure that members are informed of the current state of *Oldest Available Wrapper* and do not continue to *NAK* wrappers that are no longer available. The wrapper transmitted shall be either the most recent *Reliable Wrapper* message buffered, or any new wrapper that is available for immediate transmission, or an empty *Unreliable Wrapper* message.

5.7.3.4 Whenever wrappers are retransmitted, their *Oldest Available Wrapper* field shall be modified if necessary to show which wrappers are truly available. The owner shall not simply resend the value that was current at the time that the wrapper was originally transmitted.

5.7.3.5 On receiving a *NAK* for a wrapper that is to be retransmitted, the owner shall disregard subsequent *NAK* messages that arrive for the same wrapper within NAK_blanktime milliseconds of the first. A *NAK* for the same wrapper arriving after that time shall be treated as a new request for re-transmission. See Table 4 for the recommended value for NAK_blanktime (NAK_BLANKTIME).

A flowchart of a compliant wrapper processing and NAK algorithm implementation is shown in Annex A.

5.8 Positive Acknowledge – MAK and ACK

The *NAK* mechanism allows members to detect and request retransmission of missing *Reliable Wrapper* messages but in the absence of *NAK* messages, the leader cannot tell whether a member is still there. This is resolved by the *MAK– ACK* (Must Acknowledge – Acknowledge) exchange.

MAK is sent implicitly by the owner within the wrapper message and requests a member respond with an *ACK*. *ACK* is sent *transport inbound* by a member to inform the owner of its current point in the sequence.

5.8.1 MAK

The owner may indicate a *MAK* (Must Acknowledge) in a wrapper message at any time. The wrapper indicating the *MAK* identifies the current reliable sequence number and the wrapper contains a specification of the MIDs of members that are required to respond.

Since members are free to *ACK* at any time (see below), it is possible that some of the specified members have already sent an *ACK* very recently. To avoid extra *ACK* messages in this situation the *MAK* also carries a threshold number. Members that have already sent *ACK* messages up to within this number of wrappers preceding the current one may ignore the *MAK* and avoid extra traffic.

The “threshold sequence number” is the current reliable sequence number minus the threshold number. Each member shall keep a record of the most recent reliable sequence number it has acknowledged.

5.8.1.1 On receipt of an implicit *MAK* message from the leader, each member shall determine whether it’s most recently acknowledged reliable sequence number succeeds the threshold sequence number.

5.8.1.2 If a member’s most recently acknowledged reliable sequence number does not succeed the threshold sequence number, then it shall send an *ACK* immediately.

Use of the threshold allows members who have already acknowledged recent messages to avoid doing so again. If the threshold is zero the member shall always send an *ACK* message.

MAK messages are always implicit in sequenced wrappers (reliable or unreliable). The processing of a wrapper will trigger *NAK* processing if its sequence indicates a reliable wrapper has been missed.

5.8.1.3 If the implicit *MAK* message is the *only* reason for sending a wrapper, then that wrapper shall be an *Unreliable Wrapper.* This prevents members requesting the retransmission of implicit *MAK* messages.

5.8.2 ACK

The *ACK* message is sent *transport inbound* (on a reciprocal channel) by a member. It indicates, to the owner, the most recent *Reliable Wrapper* that has been correctly received and processed in consecutive order.

A member shall send an *ACK* in response to an implicit *MAK* subject to the criteria of 5.8.1.2.

A member may send an *ACK* message at any time whether or not an implicit *MAK* was indicated.

5.8.3 Off-line Status

Implicit *MAK* messages (the MAK parameters in wrapper messages) are used by the leader to determine if the member is on-line and keeping up. However, the leader cannot determine whether a member is off-line on the basis of a single missed *ACK*.

A member is defined as being off-line if no inbound message has been received from that member in response to MAK_MAX_RETRIES consecutive *MAK* messages with a minimum waiting time of MAK_TIMEOUT milliseconds after each one. Note that an explicit *ACK* response is not required; any inbound message that communicates the last sequence number is sufficient to indicate on-line status (e.g., *NAK*).

On deciding that a member is off-line the leader shall send a single *Leave* message to the member in case it is still processing messages.

5.8.4 NAK implies an ACK

Since the *NAK* message identifies the last *Reliable Wrapper* message correctly received by a member it carries an implied *ACK*.

They shall be treated as an *ACK* by both the owner and the member. The owner shall update its tables accordingly with the current state of that member while the member shall also update its value of most recent reliable sequence number acknowledged.

5.9 Message Verification and Full Reliability

Once a *Reliable Wrapper* message has been acknowledged by *all* channel members, it is said to have been *verified*. A verified wrapper may be unconditionally discarded from the owner's buffers.

If the owner buffers all *Reliable Wrapper* messages until they have been verified, the channel is said to be *fully reliable*.

5.10 Channel Usage Guidelines

5.10.1 NAK Flag and Where to Send NAK Messages

The setting of the *NAK* flag is guided by two general principles:

5.10.1.1 Outbound *NAK*s make *NAK* suppression work and so reduce the likelihood and scale of *NAK* implosion.

5.10.1.2 Only inbound *NAK*s actually cause re-transmission unless the owner is also receiving on the outbound or channel destination address.

These two points indicate that in general, *NAK* messages may be sent both inbound and outbound. However, a number of special cases apply.

5.10.1.3 In a unicast channel where there is only one member, *NAK* suppression is meaningless and *NAK_Outbound* shall be False.

5.10.1.4 If the channel is small and negative acknowledgement is infrequent, then *NAK* implosions may not be a significant burden and *NAK_Outbound* may be False.

5.10.1.5 In a fast and small network where the owner responds rapidly to *NAK* messages, the random back off before sending a *NAK* may mean that a missing packet is likely to be re-transmitted before many members have sent *NAK* messages and so *NAK_Outbound* may be false. If choosing to implement this functionality, the owner should implement some reasonable algorithm that detects and minimizes NAK implosions.

5.10.2 Acknowledgement Algorithms

The use of *MAK* and *ACK* depend on the needs of the channel and the purpose and character of the channel. More frequent implicit *MAK* messages give lower on-line status latency but take more bandwidth. Whatever the *MAK* rate, the owner is required to ensure that the resultant *ACK* traffic is spread across time.

For example in a channel of 100 members, where the owner is sending an average of 50 packets outbound each second, it is not recommended to set the *MAK* range to include all members, once per second. Instead the owner may require *ACK* messages from just two members with every outbound packet and that would ensure that the inbound *ACK* traffic is well spread out.

There is a trade-off that balances the rate at which members send *ACK* messages back to the owner and the size of the channel on one side, against network load and latency on the other. The owner should generally set a

MAK range that identifies only a subset of the members and is required to change the subset in a cycle that spreads *ACK* traffic evenly (rather than being bunched together at one point). Acknowledgement algorithms may need to take into account channel specific requirements for example: certain members may have higher importance and may need to send *ACK* messages more frequently.

5.10.3 Levels of Reliability

For full outbound reliability in a channel the owner must buffer all outgoing outbound messages until they are verified (5.9). However, in a channel with a long *ACK* cycle this may require excessive buffering. By managing the *ACK* cycle independently from the buffer depth, the channel may operate with intermediate levels of reliability and online status latency.

If the owner buffers only a few messages there is a very good probability in many networks that any lost messages may be detected and negatively acknowledged well before those messages would be verified. Thus without full reliability and with no *ACK* messages at all, the reliability of outbound messages in the presence of occasional and random message corruption may be greatly improved. Also missed messages may be detected (but not retransmitted) even after some of them have been discarded. The presence of the *Oldest Available Wrapper* field in messages allows members to know instantly whether a missed *reliable* message is recoverable.

5.10.4 Online Status and Retransmission Latency

Missing wrappers are not detected until subsequent wrappers are sent, which may be some time, while for online status, the main criterion is the period of the ACK cycle that varies with both the length of the cycle in messages and with the frequency with that messages are sent. For a channel in which messages are only sent occasionally but in which short retransmission latency on missed packets is important, the owner may choose to send empty wrappers indicating non-empty MAK ranges to keep the latency reasonably short.

5.11 Ad-hoc Reliability

Ad-hoc messages (e.g., *Join, GetSessions, and Sessions*) have no automatic reliability. Normally these messages are part of a direct exchange. In these cases reliability is provided by an increasing timeout and retransmission. The timeout starts at the value AD_HOC_TIMEOUT (see Section 7.3) and doubles for each retry. When a response to an ad-hoc SDT message is not received within the computed timeout the sender of the original message may retry up to AD_HOC_RETRIES times in an attempt to get an answer.

Also, SDT does not specify timeouts for other protocols using root level PDUs.

6 SDT Message Summary

Channel Data Wrappers	Symbol	Contains PDU Block	Sent to Ad-hoc Address in SDT Base layer PDU	Sent to Destination Address in SDT Base layer PDU	Sent to Destination Address in Wrapper PDU	Sent to Source Address in SDT Base layer PDU
Reliable Wrapper	REL WRAP	✓		✓		
Unreliable Wrapper	UNREL WRAP	✓		✓		
Channel Membership						
Join	JOIN		✓			
Join Refuse	JOIN REFUSE					✓
Join Accept	JOIN ACCEPT					✓
Leave	LEAVE				✓[4]	
Leaving	LEAVING					✓
Channel Parameters						
Channel Params	CHANNEL PARAMS				✓[4]	
Reliability & Sequencing						
Acknowledge	ACK				✓[4]	
Negative Acknowledge	NAK			✓[3]		✓
Session Membership						
Connect	CONNECT				✓	
Connect Accept	CONNECT ACCEPT				✓[4]	
Connect Refuse	CONNECT REFUSE				✓[4]	
Disconnect	DISCONNECT				✓	
Disconnecting	DISCONNECTING				✓[4]	
Client Protocols						
Client protocol PDUs		✓[2]			✓	
Diagnostic						
Get Sessions	GET SESSIONS		✓			
Sessions[1]	SESSIONS					

Notes:
1 Sent to source address of packet containing *Get Sessions* message.
2 Depends on client protocol specification.
3 Sent to channel downstream for NAK suppression.
4 Sent outbound on reciprocal channel, but association field indicates intended inbound channel.

Table 1: SDT Message Summary

7 Protocol Symbolic Parameters

7.1 Protocol Code

Symbol	*Value*	*Definition*
SDT_PROTOCOL_ID	1	PDU protocol value

Table 2: Protocol Code

7.2 PDU Vector Codes

Symbol	*Value*	*Definition*
REL WRAP	1	PDU vector value
UNREL WRAP	2	PDU vector value
CHANNEL PARAMS	3	PDU vector value
JOIN	4	PDU vector value
JOIN REFUSE	5	PDU vector value
JOIN ACCEPT	6	PDU vector value
LEAVE	7	PDU vector value
LEAVING	8	PDU vector value
CONNECT	9	PDU vector value
CONNECT ACCEPT	10	PDU vector value
CONNECT REFUSE	11	PDU vector value
DISCONNECT	12	PDU vector value
DISCONNECTING	13	PDU vector value
ACK	14	PDU vector value
NAK	15	PDU vector value
GET SESSIONS	16	PDU vector value
SESSIONS	17	PDU vector value

Table 3: PDU Vector Codes

7.3 Timeouts and Retries

All times & intervals are specified in milliseconds unless otherwise specified.

Symbol	***Value***	***Definition***
MAK_TIMEOUT	MAK_TIMEOUT_FACTOR *Channel Expiry Time (see Section 3.5.1.4).	After a leader sends an implicit "*MAK*" message, it shall wait this time for a response.
MAK_TIMEOUT_FACTOR	See interoperability profile	This factor is multiplied by the channel expiry to determine the MAK_TIMEOUT.
MAK_MAX_RETRIES	See interoperability profile	A leader resends the wrapper indicating a "*MAK*" message this many times after timeout before assuming member has left the session.
AD_HOC_TIMEOUT	See interoperability profile	Initial time for first retry of an ad-hoc SDT message (see Section 5.11)
AD_HOC_RETRIES	See interoperability profile	Maximum number of times to retry an ad-hoc message
RECIPROCAL_TIMEOUT	RECIPROCAL_TIMEOUT_FACTOR_FACTOR*Channel Expiry Time (see Section 3.5.1.4).	Time to allow for receipt of initial "*ACK*" message while in join pending state.
RECIPROCAL_TIMEOUT_FACTOR	See interoperability profile.	This factor is multiplied by the channel expiry to determine the RECIPROCAL_TIMEOUT.
MIN_EXPIRY_TIME	See interoperability profile	The minimum number of seconds to use for the Expiry parameter of a channel. The minimum value that an interoperability profile may specify shall be 1 second.
NAK_TIMEOUT	NAK_TIMEOUT_FACTOR * Channel Expiry Time (see Section 3.5.1.4).	How long to wait after sending a NAK message before restarting NAK processing
NAK_TIMEOUT_FACTOR	See interoperability profile	This factor is multiplied by the channel expiry to determine the NAK_TIMEOUT.
NAK_MAX_RETRIES	See interoperability profile	Maximum number of times to retry a NAK message.
NAK_HOLDOFF_INTERVAL	See interoperability profile	Recommended scaling factor for the number of milliseconds to wait before sending a NAK message
NAK_MAX_TIME	See interoperability profile	Recommended maximum number of milliseconds to wait before sending a NAK message
NAK_BLANKTIME	See interoperability profile	Recommended value for time after receiving a NAK message during which further matching NAKs are assumed to be duplicates

Table 4: Timeouts and Retries

7.4 Other Symbolic Parameters

Symbol	***Value***	***Definition***
SDT_MULTICAST_PORT	See interoperability profile	All multicast communication shall use

		this standard port.

Table 5: Standard Port

Reason Codes	***Value***	***Definition***
Nonspecific	1	Non-specific, non-SDT reason.
Illegal Parameters	2	Illegal channel parameters.
Low Resources	3	Insufficient resources.
Already Member	4	Multiple MIDs for single component.
Bad Address Type	5	Unrecognized transport layer address type.
No Reciprocal Channel	6	No upstream channel and can't create one (for reciprocal inbound messages).
Channel Expired	7	Channel has expired.
Lost Sequence	8	Unrecoverable packets missed.
Saturated	9	Can't keep up, processor saturation.
Transport Address Changing	10	Component changing transport layer address (e.g., IP number lease expired).
Asked to Leave	11	Asked to leave the channel.
No Recipient	12	Component does not support protocol ID.
Only Unicast Supported	13	Only unicast channels are supported by this component.

Table 6: Reason Codes

Address Type	***Value***	***Definition***
SDT_ADDR_NULL	0	Address is not present (0 octets).
SDT_ADDR_IPV4	1	Address specified is in IP v4 format
SDT_ADDR_IPV6	2	Address specified is in IP v6 format

Table 7: Address Specification Types

Annex A: Compliant Wrapper Processing Algorithm

The following figures show one possible Standard-compliant wrapper processing algorithm. Compliant implementations are not expected to exactly implement the algorithm as shown in these figures. The figures do help to point out how the different aspects of channel set up and message sequences impact where and when a NAK message it sent.

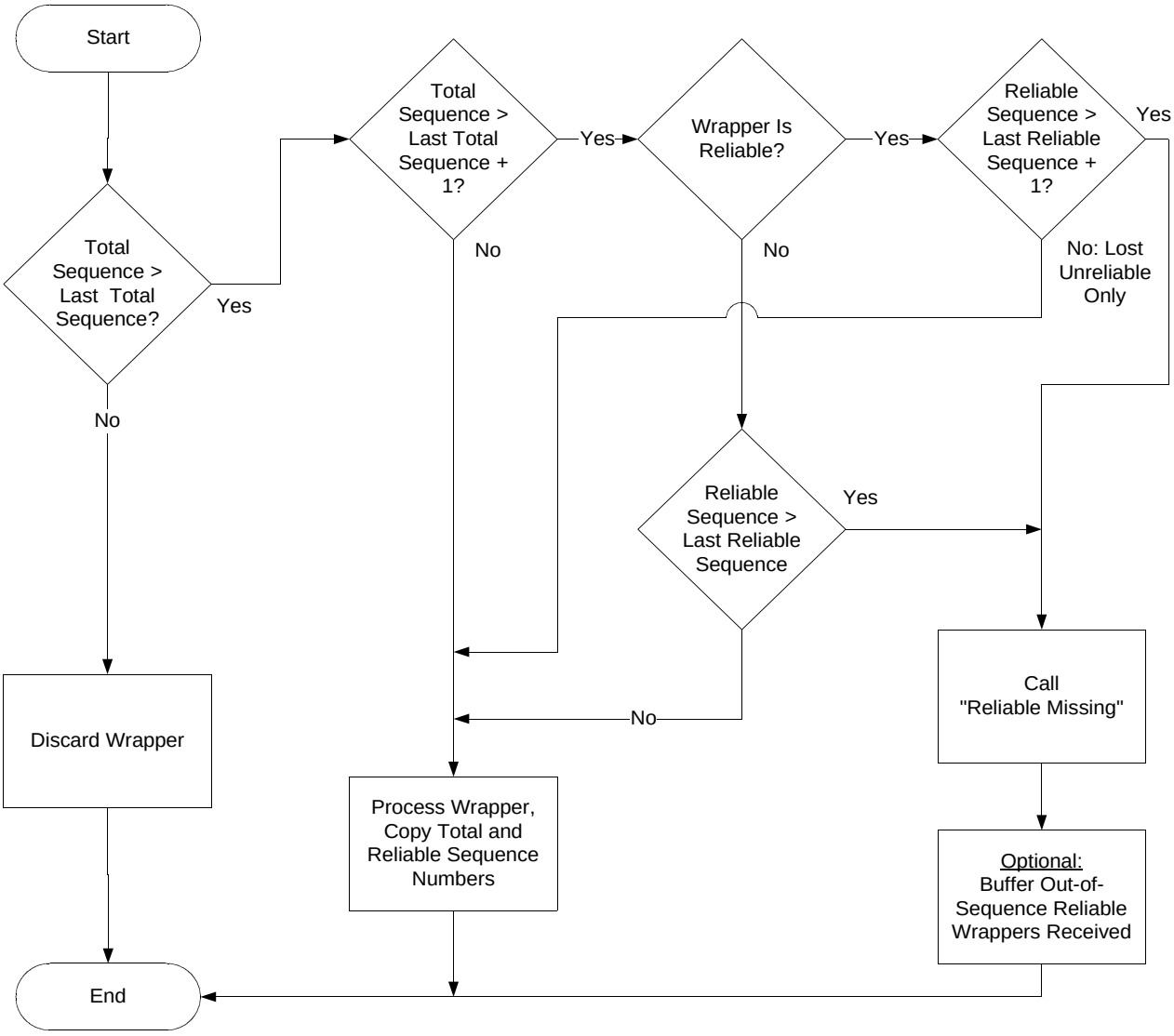


Figure 36: Example Wrapper Processing Algorithm Flowchart

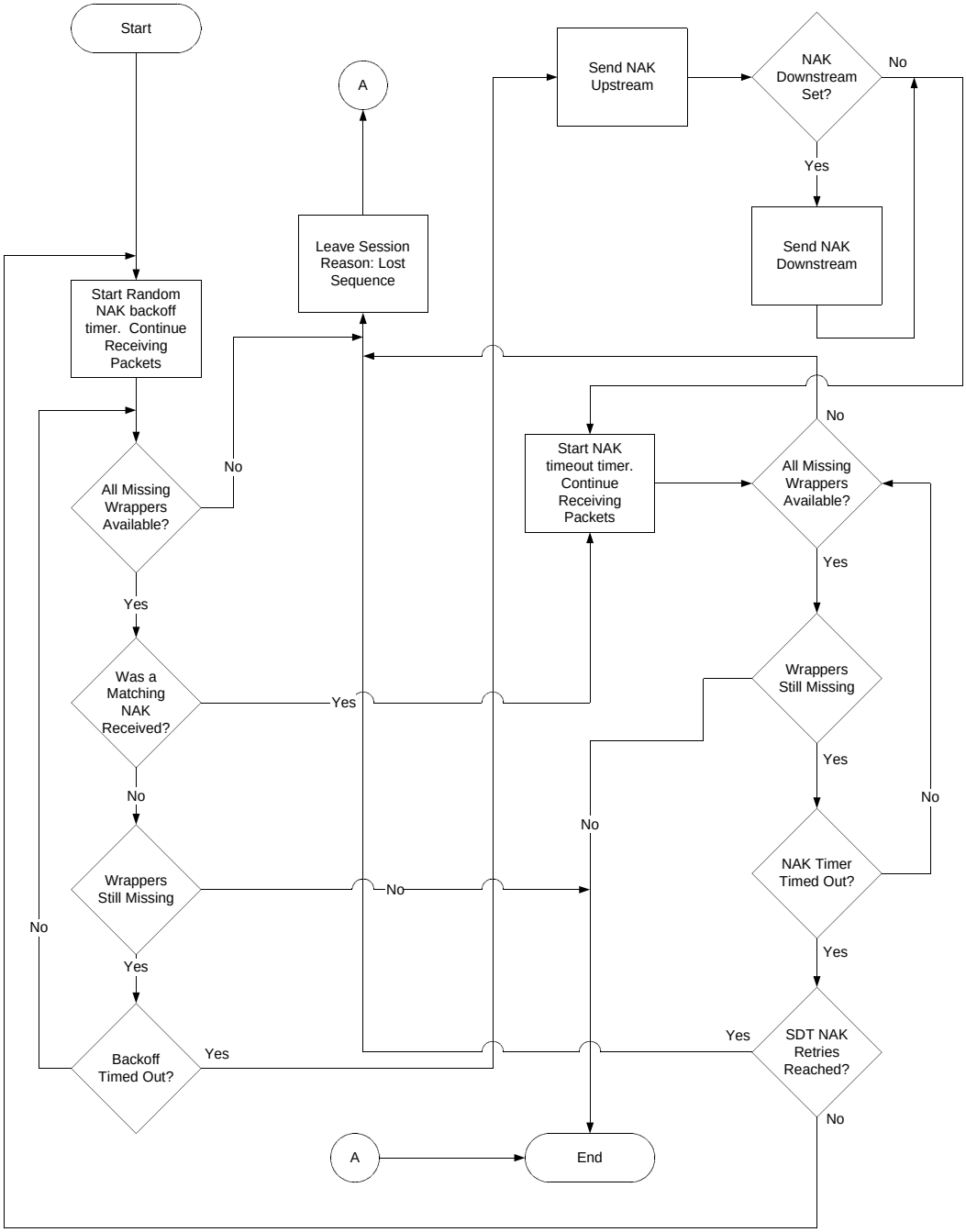


Figure 37: Example NAK Algorithm Flowchart, Subroutine “Reliable Missing”

Annex B: E1.17 Definitions

access protocol: The network or datalink protocol used to access and control the devices described by a Device Description (e.g. DMP for a DMP device). DDL may be adapted to many different access protocols both within and outside of E1.17 and a single device may support multiple access protocols.

ad hoc: Communication between two components that does not occur within a session. Ad hoc messages are passed directly as enclosing layer PDUs and have no reliability or ordering mechanism.

appliance: In DDL an appliance is a piece of equipment described by a root device and all its children and descendents. In DMP systems an appliance corresponds to a component that exposes one or more devices (since the rules require that all devices are descendants of a single root device). See also root device.

canonical form: Many specifications allow a variety of forms or methods for some items. This is usually useful in the interests of flexibility or ease of processing (for instance by removing the need for extensive checking). However, there may be times when it is required to have exactly one way of representing each case and in this situation a canonical form specifies which of a set of choices must be used such that each possible case has exactly one unambiguous representation. For example, the floating point numbers 12E5 and 1.2E6 are identical. Many specifications dictate that the normalized form with exactly one non-zero digit before the decimal point (1.2E6) be used as the canonical form.

CID: Component Identifier. A UUID [UUID] identifying a particular component.

client protocol: A (higher layer) protocol whose messages are transported by the current one. e.g. DMP is a client protocol of SDT, while SDT is a client protocol of the ACN Root Layer Protocol and the Root Layer Protocol is a client of UDP (or other transport). See also transport protocol.

component: The process, program or application corresponding to a single ACN endpoint. All messages in ACN are sent and received by a component. See [Arch] for a more complete definition.

control session: Any session in DMP's transport protocol (e.g. SDT) in which a DMP controller performs property manipulation on devices (i.e. sends control messages).

controller (DDL): A machine or program that is used to control devices described by DDL. The controller is connected to the device(s) via one or more data links or networks using one or more access protocols.

controller (DMP): An ACN component capable of retrieving and setting DMP properties in other components using DMP.

DCID: Device class identifier. This is a unique identifier UUID for a Device Class. All devices have a DCID which they share with all other devices of their class. The DCID may be used as a key to recognize or retrieve the device description for a device class.

DDL: Device Description Language. See [DDL].

device (DMP): A device is part of a Component that exposes properties that may be manipulated by a controller using DMP (a component may contain zero or more devices). A device is always of a Device Class and has an associated DCID and device description.

device (DDL): A device is an entity that may be monitored and controlled by means of a network or datalink and where a DDL description is available describing how to do this. A device includes all those devices (sub-devices) contained within it (device is a recursive term).

device class: The set of devices that are all described by the same device description.

device description: The complete textual definition of a class of devices. Device descriptions are written in E1.17 Device Description Language (DDL). Because of the recursive definition of a device in DDL, a Device Description is also defined recursively and may describe an entire piece of equipment or a part of it.

downstream channel: The channel owned by a session leader and used to communicate to session members.

downstream traffic: Data sent from the session leader to members within the session using the downstream channel.

EPI: ESTA Profile for Interoperability – Synonymous with “Interoperability Profile”. A brief document that specifies a particular usage or method to be used within ACN systems to ensure interoperability within a particular well defined context. The use of EPIs allows interoperable implementations to be defined for specific environments without restricting the applicability of ACN to other different environments. EPIs can and often do reference other EPIs as part of their specification.

event session: Any session in DMP's transport protocol (e.g. SDT) in which a DMP device sends event messages to report the value of one or more properties.

expose: Features or functionality of a component that may be discovered by the E1.17 discovery mechanisms and are then potentially accessible over the network are said to be exposed by that component. e.g. the term is applied to properties and devices accessible via DMP.

interoperability profile: See EPI.

IPv4: Internet Protocol version 4.

IPv6: Internet Protocol version 6.

message: A valid PDU.

MID: Member Identifier that represents a component within a session. MIDs are unique within a session.

ordering: A mechanism that ensures that all messages within a session are processed by the members in the order intended by the leader.

packet: A PDU block sent as a distinct unit in the underlying protocol (e.g. a UDP datagram).

PDU: Protocol Data Unit is a single message (or command). A PDU may contain a PDU block with further embedded PDUs.

PDU block: A contiguous set of PDUs (messages) within a packet or containing PDU.

property (DMP): A property is a single setting within a device. (e.g. intensity) Properties are addressed within a component.

property (DDL): A property is one aspect or facet describing an entity. e.g. a dimmer might have a property of intensity, while the intensity property could itself have a property of maximum level.

proxy: A proxy is a DMP component that exposes a set of properties that are then mapped to other properties within this or other components. Proxies may be used for many purposes in a DMP system (e.g. they can be used to connect two systems, change the interface to devices, control access to devices, and act as splitters and routers).

reliable message: Any message (PDU) transmitted inside the reliable sequenced wrapper of SDT.

root device: An instance of a device described in DDL that has no parent device. See also appliance.

root layer PDU: A PDU contained within the PDU block at the packet level.

semantic: A term common in linguistics, computer science, logic and formal languages (e.g., DDL) that identifies the meaning of an item or construct as distinct from its syntax (the term grammar is loose and is sometimes taken to include semantics but more often excludes it).

session: A session has a single leader and zero or more session members. The leader communicates to members using the downstream address. Members respond to the leader on the upstream address. A unique session identifier identifies a session.

session identifier: The combination of the session leader's CID, a leader assigned session number and the protocol ID of the protocol carried in the session. This unambiguously identifies one particular session.

session leader: A component that creates, configures, and manages a session. The leader is the only component to send downstream and receive upstream within a session. (With the exception that members may send NAK messages downstream when configured to do so by the leader).

session member: A component that has formally joined a session. Members receive downstream and send upstream (as a special case, members may also send NAK messages downstream).

transport frame: The structure that is used by the underlying protocol communication layer to send and receive E1.17 Root Layer Protocol packets. It includes the E1.17 Packet and any header and/or other enclosing contextual information. (e.g. the header and payload of a UDP/IP datagram)

transport protocol: A (lower layer) protocol the current one uses for transporting its messages (e.g. SDT is the transport protocol for DMP etc.). Also see client protocol.

unreliable message: Any message (PDU) not sent using the SDT reliability mechanism (See reliable message). Unreliable messages sent in a session are still ordered. Reliability shall be provided by other means if it is required for these PDUs (e.g. by timeouts or by separate protocols at other layers).

upstream channel: The channel owned by a session member and used to communicate to the session leader.

upstream traffic: Packets sent from members to the leader within the session using the upstream channel.

wrapper: A PDU containing an embedded PDU block that may contain arbitrary PDUs for other protocols (and/or for SDT itself).

Annex C: References

Normative

[ACN] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ANSI E1.17 *Entertainment Technology - Architecture for Control Networks - The edition current when this Standard is approved.*

[Arch] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ESTA TSP *Entertainment Technology – Architecture for Control Networks*. “ACN” Architecture. - *The edition current when this Standard is approved.*

[DDL] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ESTA TSP *Entertainment Technology - Architecture for Control Networks*. Device Description Language. - *The edition current when this Standard is approved.*

[Discovery-IP] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ESTA TSP *Entertainment Technology - Architecture for Control Networks*. EPI 19. Discovery on IP networks. - *The edition current when this Standard is approved.*

[UUID] Internet Engineering Task Force (IETF) [http://ietf.org/]. RFC 4122 [http://ietf.org/rfc/rfc4122.txt]. P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. July 2005.

Informative

[SDT] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ESTA TSP *Entertainment Technology - Architecture for Control Networks*. Session Data Transport Protocol. - *The edition current when this Standard is approved.*

[DMX512] Entertainment Services and Technology Association, since 1 January 2011 "PLASA North America" [http://tsp.plasa.org/]. ANSI E1.11-2008. *USITT DMX512-A - Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories*. 2008.