ETSI TS 102 127 V6.6.0 (2006-04)

Technical Specification

Smart Cards;
Transport protocol for CAT applications;
Stage 2
(Release 6)

Reference

RTS/SCP-T0015r3

Keywords

protocol, smart card, transport

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Smart Card Platform (SCP).

The contents of the present document are subject to continuing work within TC SCP and may change following formal TC SCP approval. If TC SCP modifies the contents of the present document, it will then be republished by ETSI with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x: the first digit:

0 early working draft;

1 presented to TC SCP for information;

2 presented to TC SCP for approval;

3 or greater indicates TC SCP approved document under change control.

y: the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z: the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The present document defines the stage two description of the Card Application Toolkit Transport Protocol (CAT_TP), for CAT applications based on TS 102 223 [2].

The present document contains the core functionalities for the CAT_TP between two CAT_TP entities hosting for instance on a UICC and on a remote entity.

The CAT_TP described in the present document is based on RDP version 2 as specified in RFC 908 [4] and RFC 1151 [5].

The present document describes, according requirements defined in TS 102 124 [1]:

- The core functionalities of CAT_TP (data structures, state diagrams, protocol procedures, etc.).

- Usage of CAT_TP on top of the Bearer Independent Protocol (BIP).

The following items are out of the scope of the present document:

- The specific implementation of an API.

- Anything dealing with the security above CAT_TP.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to an TC SCP document, a non-specific reference implicitly refers to the latest version of that document in the same Release as the present document.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

[1]   ETSI TS 102 124: "Smart Cards; Transport Protocol for UICC based Applications; Stage 1 (Release 6)".

[2]   ETSI TS 102 223: "Smart cards; Card Application Toolkit (CAT) (Release 7)".

[3]   IETF RFC 793 (1981): "Transmission Control Protocol".

[4]   IETF RFC 908 (1984): "Reliable Data Protocol".

[5]   IETF RFC 1151 (1990): "Version 2 of the Reliable Data Protocol (RDP)".

[6]   ETSI TS 102 221: "Smart cards; UICC-Terminal interface; Physical and logical characteristics (Release 7)".

[7]   ISO/IEC 10731 (1994): "Information technology - Open Systems Interconnection - Basic Reference Model - Conventions for the definition of OSI services".

3 Definitions, symbols, abbreviations and coding conventions

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

Bearer Independent Protocol (BIP): mechanism by which the TE provides the UICC with access to the data bearers supported by the terminal and the network, as defined in TS 102 223 [2]

CAT_TP client: entity which initiates a CAT_TP link to the CAT_TP server, and applies during the connection phase only

NOTE: It could be on the UICC or on the remote entity.

CAT_TP entity: entity able to open a CAT_TP link, exchange CAT_TP PDUs, and close the CAT_TP link

CAT_TP link: logical link between CAT_TP entities over which CAT_TP PDUs are exchanged

CAT_TP Port: this 16-bit identifier is used to identify the CAT_TP upper layer process

CAT_TP server: entity which receives a CAT_TP link establishment request from a CAT_TP client, and applies during the connection phase only

NOTE: It could be on the UICC or on the remote entity.

CAT_TP service data unit: in the reference model for OSI, amount of information whose identity is preserved when transferred between peer (N+1)-layer entities and which is not interpreted by the supporting (N)-layer entities

NOTE: Here (N)-layer is the CAT_TP layer.

lower layer: within consideration a (N)-layer, the lower layer is the just below layer which is the (N-1)-layer

NOTE: For the CAT_TP, the lower layer may be the Physical layer.

physical link: composed of the Bearer Independent Protocol channel between the UICC and the TE and a bearer specific link between the TE and the remote entity

upper layer: within consideration a (N)-layer, the upper layer is the just above layer which is the (N+1)layer

NOTE: For the CAT_TP, the upper layer may be the Application layer.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

'0' to '9' and 'A' to 'F' are the sixteen hexadecimal digits.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACK	ACKnowledgement
BIP	Bearer Independent Protocol
CAT	Card Application Toolkit
CAT_TP	Card Application Toolkit Transport Protocol
FFS	For Further Study

LSB   Least Significant Bit
MSB   Most Significant Bit
OSI   Open System Interconnection
PDU   Protocol Data Unit
RFU   Reserved for Further Use
SDU   Service Data Unit
TE   Terminal Equipment

3.4   Coding conventions

For the purposes of the present document, the following coding conventions apply:

All lengths are presented in bytes, unless otherwise stated. Each byte is represented by bits b8 to b1, where b8 is the Most Significant Bit (MSB) and b1 is the Least Significant Bit (LSB). In each representation, the leftmost bit is the MSB.

All bytes specified as RFU shall be set to '00' and all bits specified as RFU shall be set to 0.

4   Description

The CAT_TP is a transport protocol providing the UICC with a reliable data transmission with a remote entity. Main actors of the environment are the UICC, the TE and the remote entity.
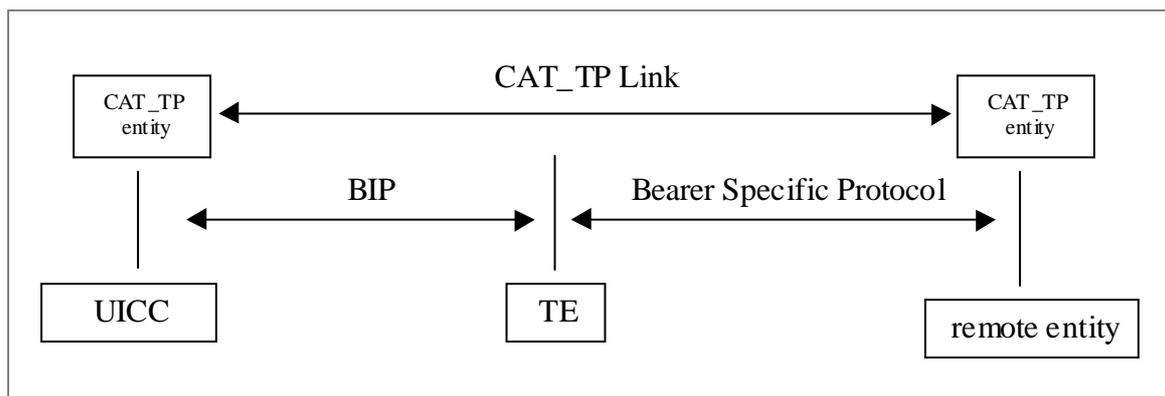
Figure 1: Environment description

The CAT_TP protocol provides following functionalities:

- CAT_TP provides a full-duplex communications channel between the two ports of each transport connection.

- CAT_TP reliably delivers all upper layer data and reports any failure to the upper layer.

- CAT_TP attempts to detect and discard all damaged and duplicated PDUs.

- CAT_TP provides sequenced delivery of SDUs. Out of sequence delivery of SDUs is FFS.

- CAT_TP segments large SDUs into PDUs on a CAT_TP sending entity and re-assembles segmented PDUs into SDUs on a CAT_TP receiving entity.

The CAT_TP ensures an end to end reliable data communication between the UICC and a remote entity over UDP/IP or other networks.
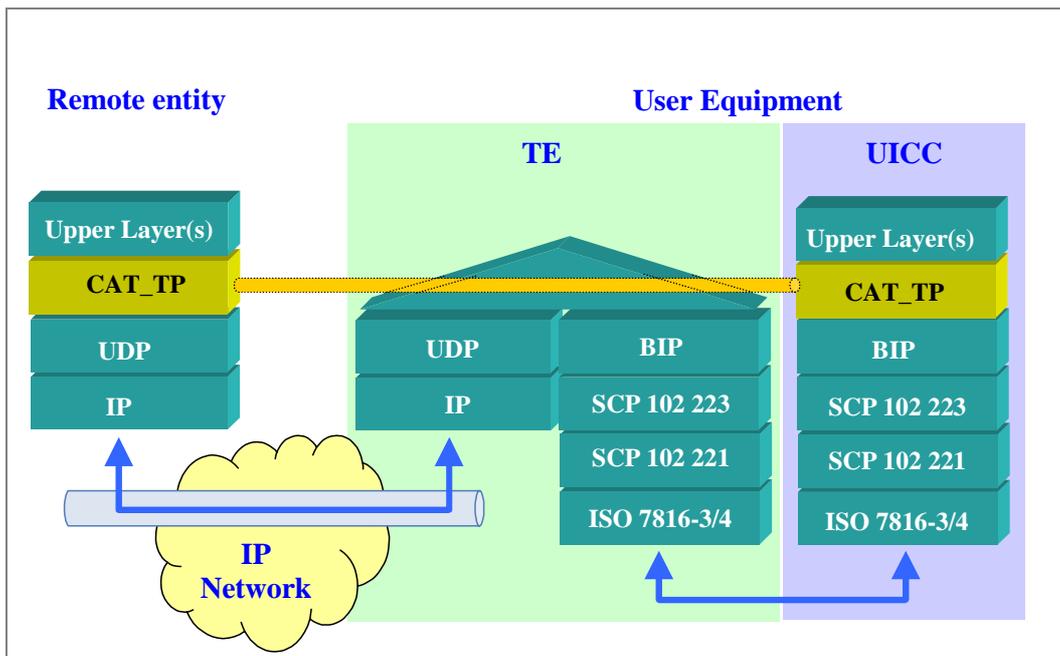
Figure 2: CAT_TP layer position

5 CAT_TP layer

This clause describes the core protocol. The CAT_TP layer is composed of two entities.
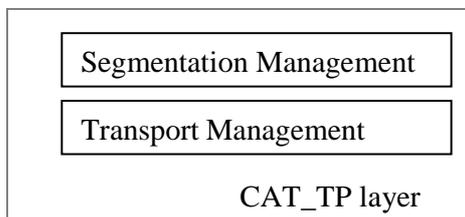
Figure 3: CAT_TP layer description

5.1 Data communication

Data flows through a CAT_TP connection in the form of PDUs. Each CAT_TP PDU is packaged as a CAT_TP header and one or more octets of data.

CAT_TP is able to fragment a large user message (CAT_TP SDU) into smaller CAT_TP PDUs and re-assemble the message on the receiving end. At the CAT_TP level, outgoing PDUs are queued as input to the lower layer as soon as they are created. Each PDU is held by the sending CAT_TP entity until it is acknowledged by the remote host.

Incoming PDUs are queued as input to the upper layer. PDUs are acknowledged when they have been accepted by the receiving CAT_TP entity. The receiving end of each connection, at the connection establishment phase, specifies the "maximum PDU size" it will accept. A CAT_TP sending entity having to send a block of data larger that this "maximum PDU size" leads to a segmentation of the data. CAT_TP will abort a connection with an RST PDU if an incoming PDU contains more data than the maximum acceptable PDU size.

CAT_TP shall deliver SDUs in sequence to the upper layer.

5.2 Segmentation management

5.2.1 Segmentation and re-assembly

The segmentation is a function used by a layer N because of system capabilities (physical transmission, etc.). The layer N has a N-SDU to transmit and may have to split it into several N-PDUs. On the reception side, the layer N has to re-assemble N-PDUs into the initial N-SDU. Here is a presentation of one layer exchanging PDUs.
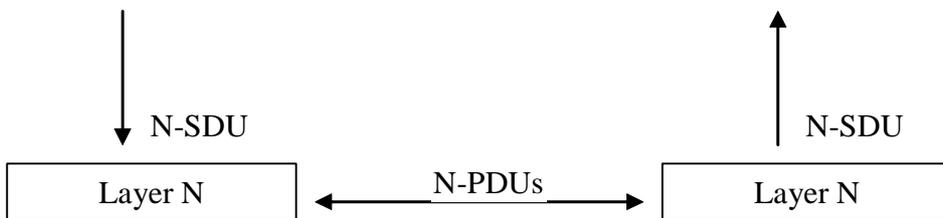
Figure 4: PDUs exchange between two Layers N

5.2.1.1 N-SDU and N-PDU when segmentation is needed
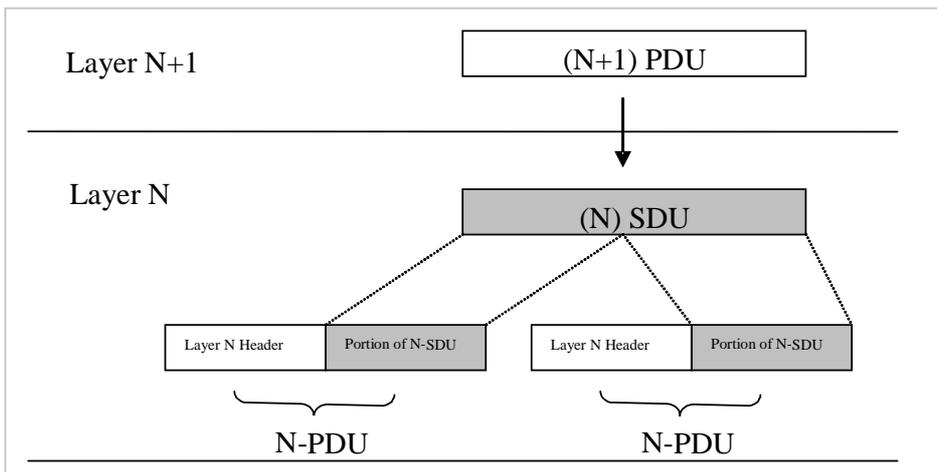
Figure 5: N-SDU and N-PDU with segmentation

5.2.1.2 N-SDU and N-PDU when no segmentation
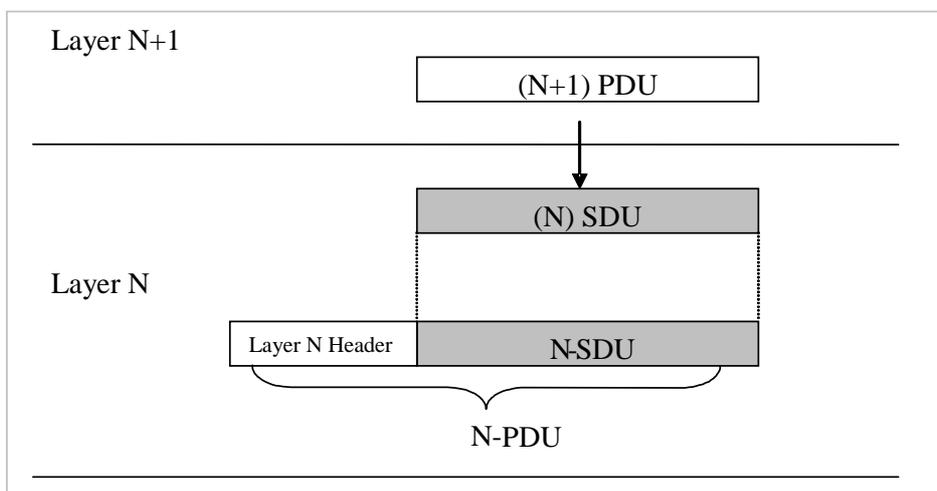
Figure 6: N-SDU and N-PDU with no segmentation

5.2.2   Limitations

The main limitation is provided by the buffer size of the Terminal Equipment. This parameter shall be taken into account during the segmentation and the re-assembly mechanism.

To ease the reassembly, all PDUs of one segmented SDU shall have the same length, except possibly the last one, which may be shorter. The PDUs may be smaller than the maximum supported PDU size announced by receiver (SND_PDU_SIZE_MAX in clause 5.3.1.4).

Another limitation is that the CAT_TP header shall not be segmented.

5.2.3   CAT_TP segmentation management

The task of the CAT_TP Segmentation Manager is to split SDUs of exceeding length into two or more PDUs that will each fit into one packet of the CAT_TP Transport Manager.

Only one bit, named SEG, is required to indicate fragments that are not single PDUs or the last fragment. This bit constitutes the CAT_TP Segmentation header. Note that for the sake of simplicity and space, the CAT_TP Transport Manager integrates this one bit header into its own header.

SEG is defined as follows:

- SEG = 0: this is a single packet or the last fragment.

- SEG = 1: this is a non-last fragment.

Thus SEG behaves like a follow bit - it indicates "more data for this packet is following".
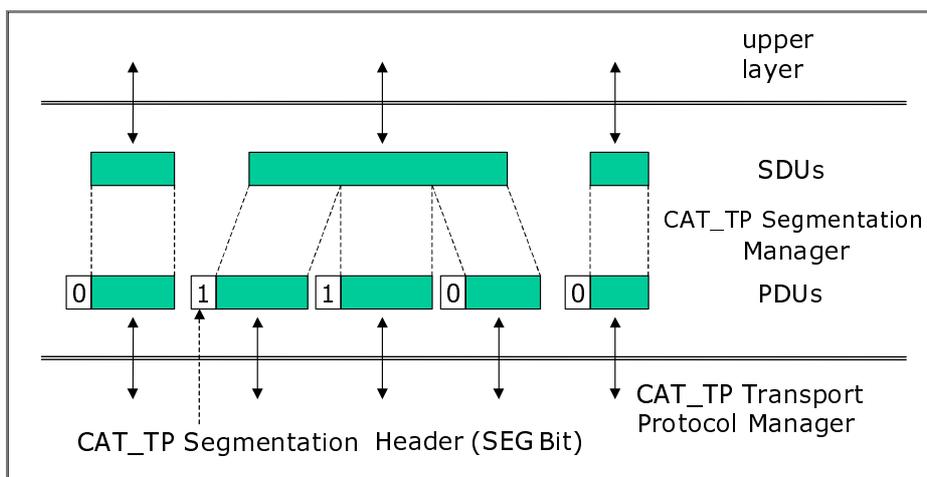
Figure 7: Segmentation description

Protocol operation at sending side:

- In the case that the CAT_TP Segmentation Manager discovers that the SDU to be sent is longer than the maximum PDU length, the SDU is segmented. The CAT_TP Segmentation Manager adds the SEG bit headers and passes the PDUs to the CAT_TP Transport Manager. SDUs shorter or equal to the maximum PDU length are passed on directly with SEG = 0.

Protocol operation at the receiving side:

- Once a PDU with SEG = 1 is received, this and all possibly following PDUs with SEG = 1 are assembled including the first following PDU with SEG = 0. All those PDUs are concatenated and the reassembled SDU is passed to the upper layer. A non-segmented PDU is indicated by having itself SEG = 0 and following a PDU with SEG = 0. This PDU is passed immediately to the upper layer.

One requirement that this segmentation mechanism works is a reliable underlying protocol that delivers PDUs in sequence. CAT_TP Transport Manager fulfils this requirement.

5.3 Transport management

5.3.1 CAT_TP connection management

CAT_TP is a connection-oriented protocol in which each connection acts, for an upper layer, as a full-duplex communication channel between two CAT_TP entities. CAT_TP PDUs from a sender are directed to a port on the destination entity. A connection is uniquely identified with the two 16-bit source and destination port identifiers and with the source and destination network identities.

5.3.1.1 Opening a connection

There are two modes for opening a connection.

- The active mode puts a CAT_TP entity into the SYN-SENT state. The CAT_TP client shall know the CAT_TP server port to be specified in the request. The CAT_TP client may specify its own port otherwise a dynamic allocation, in the "allocable" range, shall be done.

- The passive mode puts a CAT_TP entity into the LISTEN state, during which it passively listens for an open request from a remote CAT_TP entity. The CAT_TP server, which perform the passive connection request, shall specify its own port. The CAT_TP server may optionally specify a CAT_TP client port to accept a connection request from. In that case, an arriving connection request shall match to the specified CAT_TP port.

5.3.1.2 Ports

Valid port numbers range is from 1 to 65 535 (decimal). There are two types of ports: "well-known" ports and "allocable" ports. "Allocable" ports are, for instance, used during an active connection request to identify the CAT_TP client. "Well-know" ports are, for instance, used during a passive connection request to identify the CAT_TP server.

"Well know" ports are in the range from 1 to 1 023. Allocable ports are in the range from 1 024 to 65 535. A CAT_TP entity shall not use the same local port number for an active and a passive request simultaneously.

5.3.1.3   Connection states

A CAT_TP connection progresses into several states during its lifetime. The states are shown in the figure 8. Figure boxes represent CAT_TP states and arrows between them represent state change direction. The event that triggers the state change is described above the line and the resulting output is described in figure 8.
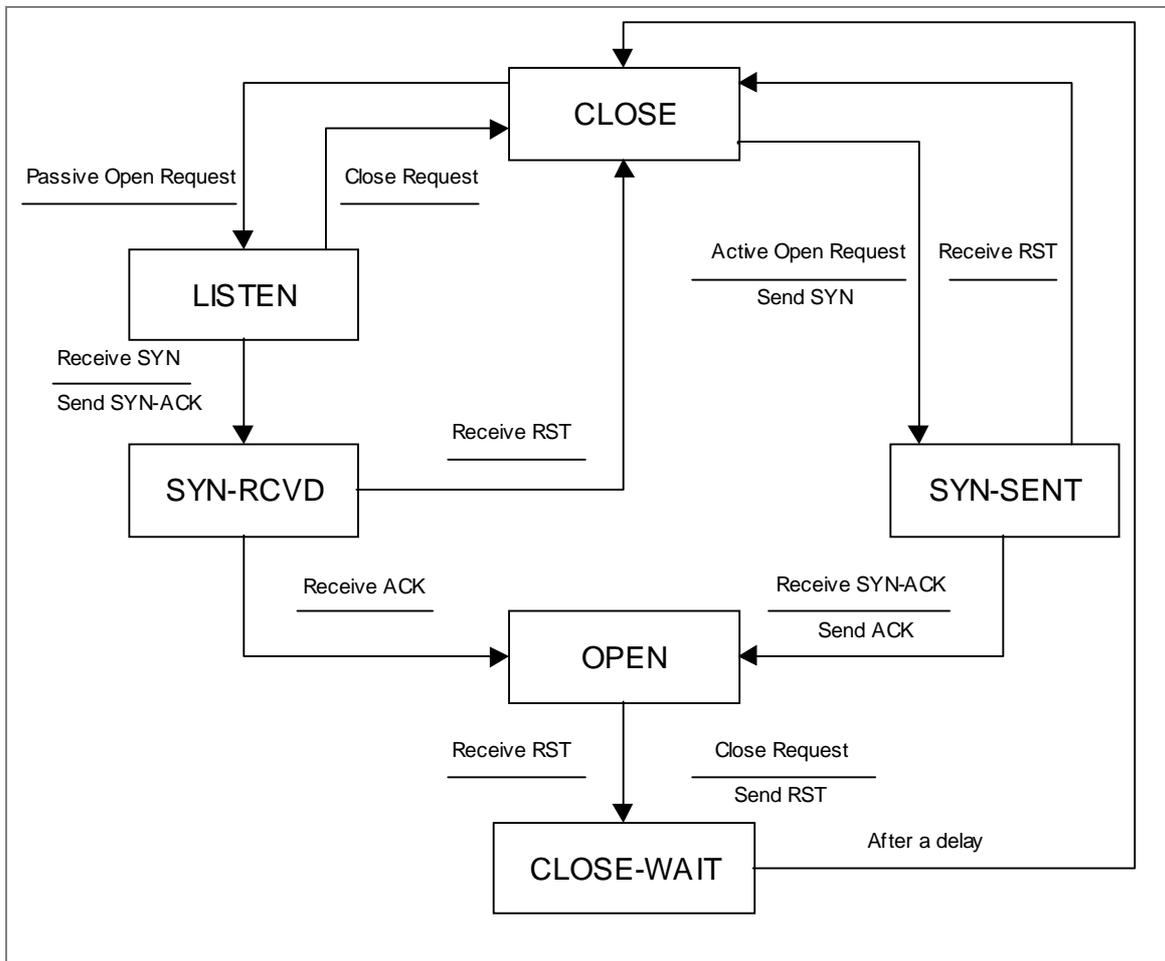
Figure 8: CAT_TP functional state machine

5.3.1.3.1   CLOSED state

The CLOSED state is valid when no connection exists between two ports. If no connection exists at all, the CLOSED state PDU arrival events processing is optional.

NOTE: As soon as one connection is in any state different from CLOSED, the CAT_TP layer shall react to every incoming PDU. For PDUs addressing connection currently not existing or ports not open, the CLOSED state applies and they shall be handled according to it.

5.3.1.3.2   LISTEN state

The LISTEN state is entered after a passive Open request. A connection record is created and the CAT_TP server waits for an active connection request reception.

5.3.1.3.3   SYN-SENT state

The SYN-SENT state is entered, by a CAT_TP entity, after having processing an active Open request. A connection record is allocated, an initial sequence number is generated, and a SYN PDU is sent to the CAT_TP server. The CAT_TP client then waits for a CAT_TP server acknowledgment (SYN-ACK PDU).

5.3.1.3.4 SYN-RCVD state

SYN-RCVD state is reached from the LISTEN state when a SYN PDU requesting a connection is received from a CAT_TP client. In reply, the CAT_TP server shall generate an initial sequence number for its side of the connection, and then sends the sequence number and an acknowledgement of the SYN PDU to the CAT_TP client. It then waits for an acknowledgement.

5.3.1.3.5 OPEN state

The OPEN state exists when a connection has been established. This is the result of a three way handshake:

- SYN.

- SYN-ACK.

- ACK.

between a CAT_TP client and a CAT_TP server. In this state, the two entities are able to begin a full-duplex communication.

5.3.1.3.6 CLOSE-WAIT state

The CLOSE-WAIT state is entered from either a Close request or from the reception of a RST PDU. In that case a CAT_TP entity shall wait for a delay period without initiating any activity on the connection and then enter the CLOSE state.

5.3.1.4 Connection record

The variables that define the state of a connection are stored in a connection record maintained for each connection. A connection shall contain the following parameters.

5.3.1.4.1 STATE

The current state of a connection may be OPEN, LISTEN, CLOSED, SYN-SENT, SYN-RCVD, and CLOSE-WAIT.

5.3.1.4.2 Variables for CAT_TP sending activity

SND_INI_SEQ_NB: This is the initial sequence number for the sending activity. This shall be the sequence number that was previously sent in the SYN PDU.

SND_NXT_SEQ_NB: This is the sequence number of the next PDU that is to be sent.

SND_UNA_PDU_SEQ_NB: This is the sequence number of the oldest sent PDU unacknowledged.

SND_PDU_SIZE_MAX: This is the largest PDU size that may be sent.

SND_SDU_SIZE_MAX: This is the largest SDU size that may be sent.

SND_WIN_SIZE: This is the current number of PDUs that can be sent, counting from **SND_UNA_PDU_SEQ_NB-1**.

5.3.1.4.3 Variables for CAT_TP receiving activity

RCV_INI_SEQ_NB: This is the initial sequence number for the receiving activity. This shall be the sequence number that was previously received in the SYN PDU.

RCV_CUR_SEQ_NB: This is the sequence number of the last PDU received correctly and in sequence.

RCV_OUT_OF_SEQ_PDU_SEQ_NBs: This is the sequence numbers that have been received out of sequence.

RCV_PDU_SIZE_MAX: This is the largest PDU size that can be received.

RCV_SDU_SIZE_MAX: This is the largest SDU size that can be received.

RCV_WIN_SIZE: This is the number of PDUs that can be received, counting from SND_UNA_PDU_SEQ_NB-1.

The values for maximum PDU/SDU sizes may differ between receiving and sending activity of one CAT_TP connection.

5.3.1.4.4 Variables from current PDU

CUR_PDU_SEQ_NB: This is the sequence number of the PDU currently being processed by a CAT_TP entity.

CUR_PDU_ACK_NB: This is the acknowledgement number in the PDU currently being processed by a CAT_TP entity.

CUR_PDU_WIN_SIZE: This is the window size of the PDU currently being processed by a CAT_TP entity.

5.3.1.4.5 Variables from SYN PDU

SYN_PDU_SIZE_MAX: This is the maximum PDU size (in octets) accepted by a CAT_TP entity.

SYN_SDU_SIZE_MAX: This is the maximum SDU size (in octets) accepted by a CAT_TP entity.

5.3.1.5 Closing a connection

The closing of a connection can be initiated by a Close request or by the reception of a RST PDU. In the case of the Close request, the CAT_TP entity shall send a RST PDU to the other side of the connection and then enter the CLOSE-WAIT state for a period of time. While in the CLOSE-WAIT state, CAT_TP shall discard PDUs received from the other side of the connection. When the time-out period expires, the connection record is deleted and the connection is terminated.

> NOTE: For a normal ending of data transmission, this simple connection closing facility requires that CAT_TP upper layer determines that all data has been reliably delivered before requesting a close of the connection.

5.3.1.6 Detecting an Half-Open and/or inactive connection

If one side of a connection crashes, the connection may be left with the other side still active. This situation is termed to be an half-open connection. A CAT_TP entity can verify the status of a connection by sending a NUL PDU. An inactive connection is defined to be a connection that has no outstanding/unacknowledged PDUs, that has no PDUs to be processed and that has not had any traffic for some period of time. To minimize network overhead, verification of such connections should only be done when necessary to prevent a deadlock situation. In case of inactive connection or half-open connection, the active CAT_TP side shall reset the connection.

5.3.2 Reliable Communication

CAT_TP implements a reliable message service through several mechanisms:

- insertion of sequence numbers;

- checksums into PDUs;

- positive acknowledgement of PDU receipt;

- timeout management for lost PDUs and retransmission of missing PDUs.

5.3.2.1 Sequence number

Each CAT_TP PDU transporting data has a sequence number that uniquely identifies it among all other PDUs in the same connection. The initial sequence number is chosen when the connection is opened and is selected by reading a value from a monotonically increasing clock.

Each time a PDU requiring an acknowledgment (i.e. ACK with Data, SYN, NUL) is sent, the sequence number is incremented by one once this PDU has been sent (that is a post-increment).

PDUs requiring no acknowledgment (i.e. ACK without data, ACK/EACK without data and RST) do not increment the sequence number.

The SYN PDU is always sent with a unique sequence number, the initial sequence number.

5.3.2.2 Checksum

Each CAT_TP PDU contains a checksum to allow the receiver to detect damaged PDUs. CAT_TP shall use the 16-bit TCP checksum, which is specified on page 16 of RFC 793 [3].

The checksum field is the 16-bit one's complement of the one's complement sum of all 16 bit words in the header and data. If a PDU contains an odd number of header and data octets to be check-summed, the last octet shall be padded on the right with zeros to form a 16 bit word for checksum purposes. The pad shall not be transmitted as part of the PDU. While computing the checksum, the checksum field itself shall be replaced with zeros.

In the TCP mechanism, the checksum also covers a 96 bit pseudo header which has conceptually to be prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length, thus giving the TCP a protection against misrouted PDUs. Concerning CAT_TP, this pseudo-header shall not be used.

5.3.2.3 Positive acknowledgement of PDUs

CAT_TP may be used above reliable as well as unreliable physical layers. In order to deal with this latter case, CAT_TP shall use positive acknowledgement and retransmission of PDUs for guaranteeing the delivery of PDUs in this kind of unreliable environment.

Each PDU containing data and the SYN and NUL PDUs are acknowledged when they are correctly received and accepted by the destination host. PDUs containing only an acknowledgement are not acknowledged. Damaged PDUs are discarded and are not acknowledged. PDUs are retransmitted when there is no timely acknowledgement of the PDU by the destination host.

CAT_TP allows two types of acknowledgement:

- A cumulative acknowledgement is used to acknowledge all PDUs up to a specified sequence number. This type of acknowledgement can be sent using fixed length fields within the CAT_TP header. Specifically, the ACK control flag is set and the last acknowledged sequence number is placed in the Acknowledgement Number field.

- The extended or non-cumulative acknowledgement allows the receiver to acknowledge PDUs out of sequence. This type of acknowledgement is sent using the EACK control flag and the variable length fields in the CAT_TP PDU header. The variable length header fields are used to hold the sequence numbers of the out-of-sequence acknowledged PDUs.

The type of acknowledgement used is simply a function of the order in which PDUs arrive. Whenever possible, PDUs are acknowledged using the cumulative acknowledgement PDU. Only out-of-sequence PDUs are acknowledged using the extended acknowledgement option.

5.3.2.4 Retransmission timeout

PDUs may be lost in transmission for two reasons:

- they may be lost or damaged due to the effects of the loss transmission media;

- they may be discarded by the receiving CAT_TP.

The positive acknowledgement policy requires the receiver to acknowledge a PDU only when the PDU has been correctly received and accepted.

To detect missing PDUs, the sending CAT_TP entity shall use a retransmission timer for each SYN, NUL and data PDU transmitted. When an acknowledgement is received for a PDU, the timer is cancelled for that PDU and the PDU is removed from the retransmission queue. If the timer expires before an acknowledgement is received for a PDU, that PDU is retransmitted and the timer is restarted.

NOTE: The retransmission timers should be set to values approximating the round trip time of a CAT_TP PDU in the network.

In addition to retransmission timers, CAT_TP shall implement a counter for the number of retries. Once a maximum number of retries has been reached, the connection shall be considered unusable and shall be closed by sending a RST PDU. The maximum value of retries shall be configurable.

5.3.3　Flow control and window management

CAT_TP employs a simple flow control mechanism that is based on a window for the PDUs the receiver is ready to accept.

Together with each Acknowledgement Number, the acknowledging entity shall announce its current acceptance window. The value specified for this parameter should be based on the amount and size of buffers that the CAT_TP is currently willing to allocate to a connection. The particular CAT_TP implementation can set the parameter to a value that is optimal for its buffering scheme.

The sender is then allowed to send all PDUs within this window without receiving any new window announcement. The highest Sequence Number the sender is allowed to send is calculated by adding the Window Size to the Acknowledgement Number. This constitutes the right window border. A right window border shall never be reduced once it was announced. A sender shall ignore a lower right border value it receives, which might occur if an older ACK PDU is delayed. A Window Size of zero is permitted, effectively closing the communication, which might be required when the receiver is temporarily out of resources. Whenever resources are available again, the new window size shall be announced in a new ACK PDU.

When a received PDU has a sequence number that falls within the acceptance window, it shall be acknowledged. If the sequence number is equal to the left-hand edge (i.e. it is the next sequence number expected), the PDU is acknowledged with a cumulative Acknowledgement (ACK). If the sequence number is within the acceptance window but is out of sequence, it is acknowledged with a non-cumulative acknowledgement (EACK).

The CAT_TP module in the transmitting host sends PDUs until all PDUs allowed by the currently defined window are sent.

Once this limit is reached, the transmitting CAT_TP module may only send a new PDU after it receives a window announcement that increases the right window border.

5.4　Events processing

This clause describes the sequence for processing events. It is not intended to suggest a particular implementation. Only the peer to peer behaviour between the protocols implemented in the CAT_TP entities is normative.

The following are the kinds of events that may occur:

- User Requests: Open, Send, Receive, Close.

- Arriving PDU: A PDU arrives.

- Timeouts: Retransmission timeout, close-wait timeout.

This description is given with SDL diagrams; user request processing always terminates with a return to the caller, with a possible error indication. Error responses are given as a status code. A response is also possible, in PDU arrival events case, indicated by the term "signal".

Processing of arriving PDUs usually follows this general sequence: the sequence number is checked for validity and, if valid, the PDU is queued and processed in sequence-number order. For all events, unless a state change is specified, CAT_TP remains in the same state.

Notation used for following diagrams:

Action: An underlined word is used to define a specific action (e.g. send, discard, start).

<PDU Type>: Between <> brackets are indicated the flag(s) to set for a PDU (e.g. <SYN> or <SYN, ACK>, etc.) and the fields values (e.g. <SEQ = SND_NXT_SEQ_NB>, etc.).

SEQ: sequence number field.

ACK: acknowledgment number field.

WIN_SIZE: window size field.

SDU_SIZE_MAX: SDU maximum size field.

PDU_SIZE_MAX: PDU maximum size field.

: This symbol indicates that the diagram continues on the following figure (the closest next one).

5.4.1 Upper layer events

The following scenarios describe the processing of events requested by the upper layer.

5.4.1.1 Open request

Figure 9 describes the process of an "Open request" in the "OPEN", "LISTEN", "SYN-SENT" or "SYN-RCVD" state.
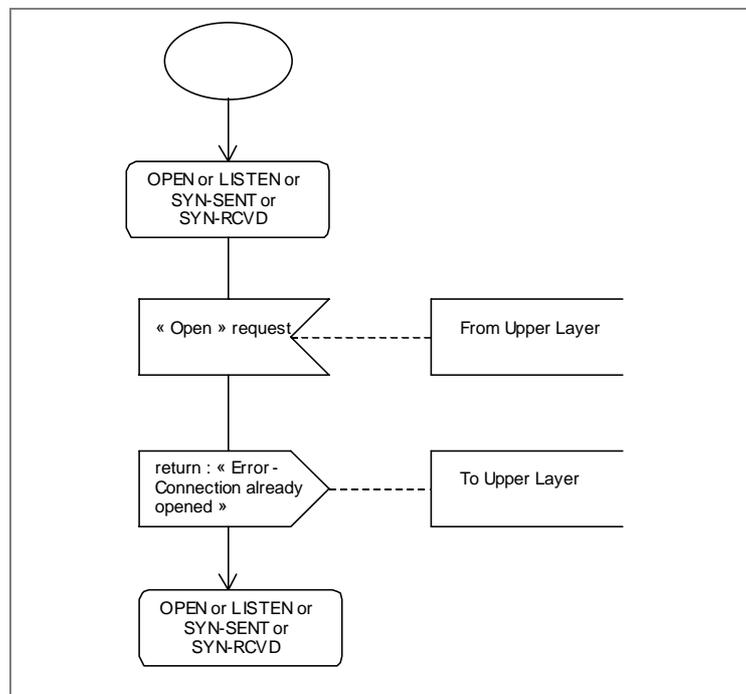
Figure 9: Open Request in OPEN, LISTEN, SYN-SENT, SYN-RCVD state

Figure 10 describes the process of an "Open request" in the "CLOSE-WAIT" state.
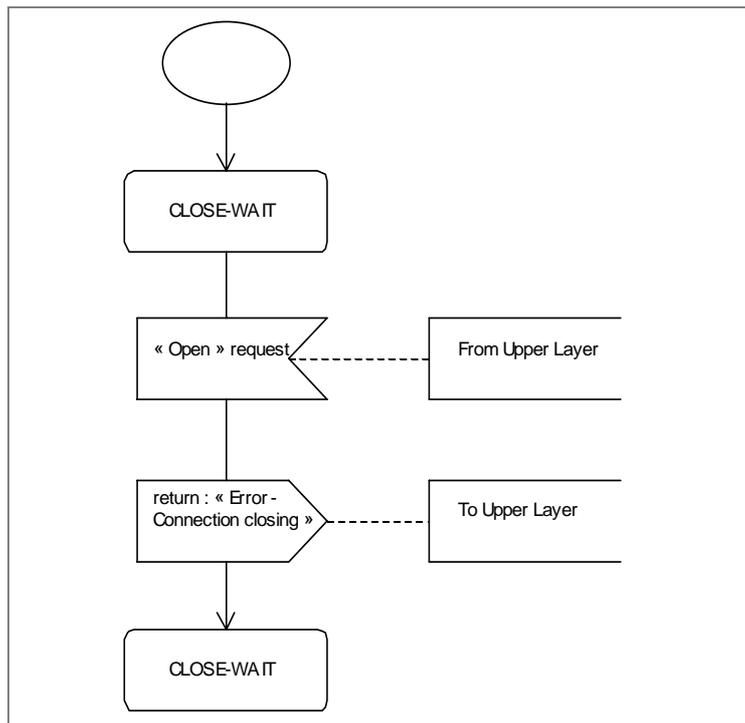
Figure 10: Open request in CLOSE-WAIT state

Figure 11 describes the process of an "Open request" in the "CLOSE" state.

Figure 11: Open request in CLOSED state

5.4.1.2 Close request

Figure 12 describes the process of a "Close request" in the "OPEN" state.
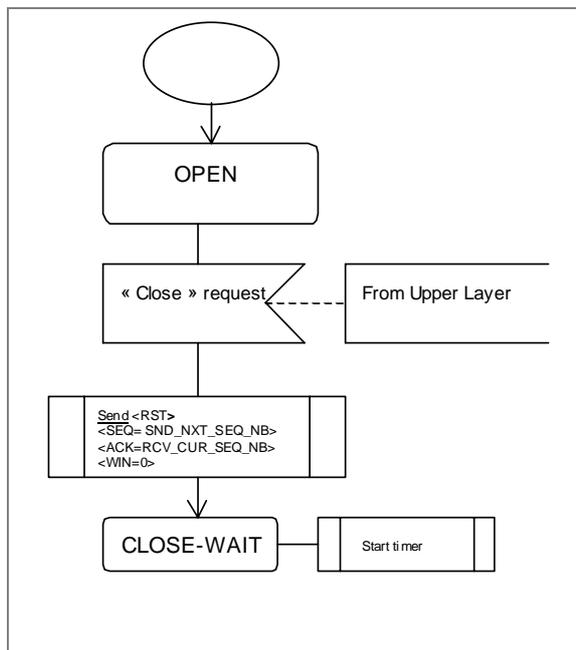
Figure 12: Close request in Open state

Figure 13 describes the process of a "Close request" in the "CLOSE" state.

Figure 13: Close request in CLOSE state

Figure 14 describes the process of a "Close request" in the "CLOSE-WAIT" state.
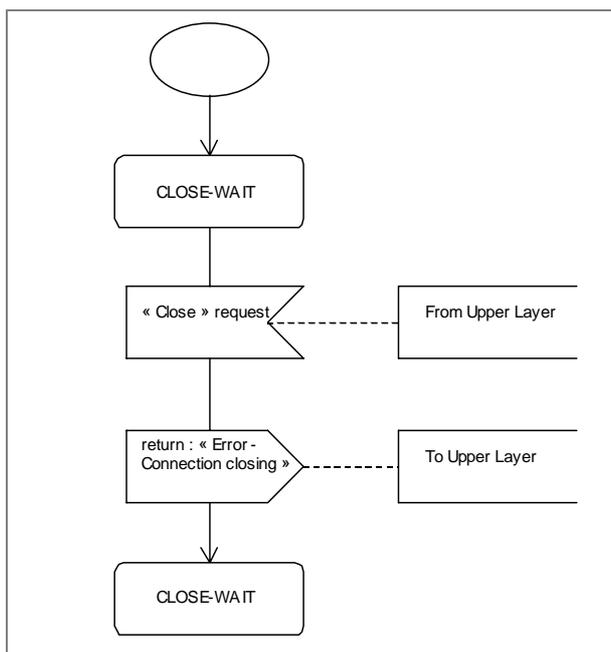
Figure 14: Close request in CLOSE-WAIT state

Figure 15 describes the process of a "Close request" in the "SYN-SENT" or "SYN-RCVD" state.
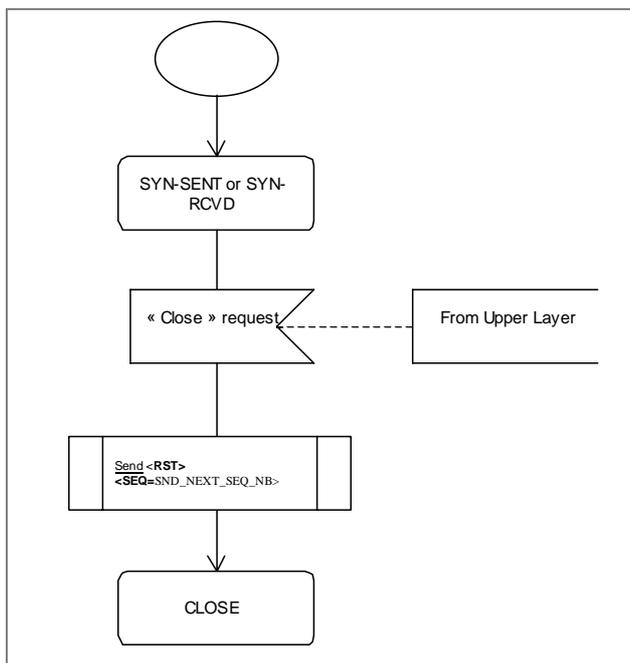
Figure 15: Close request in SYN-SENT or SYN-RCVD state

Figure 16 describes the process of a "Close request" in the "LISTEN" state.
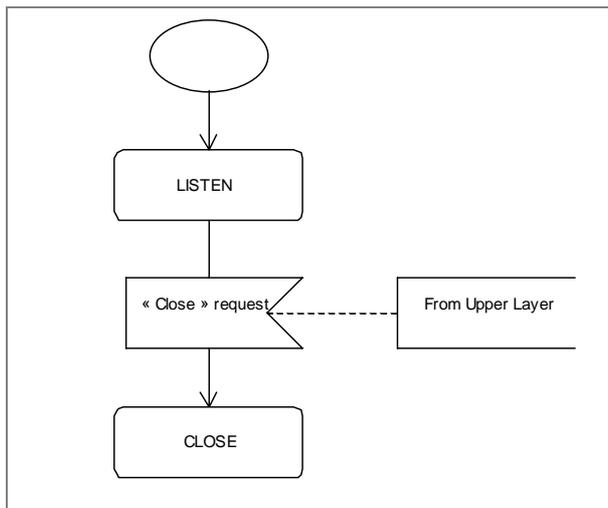
Figure 16: Close request in LISTEN state

5.4.1.3 Receive request

Figure 17 describes the process of a "Receive request" in the "OPEN" state.
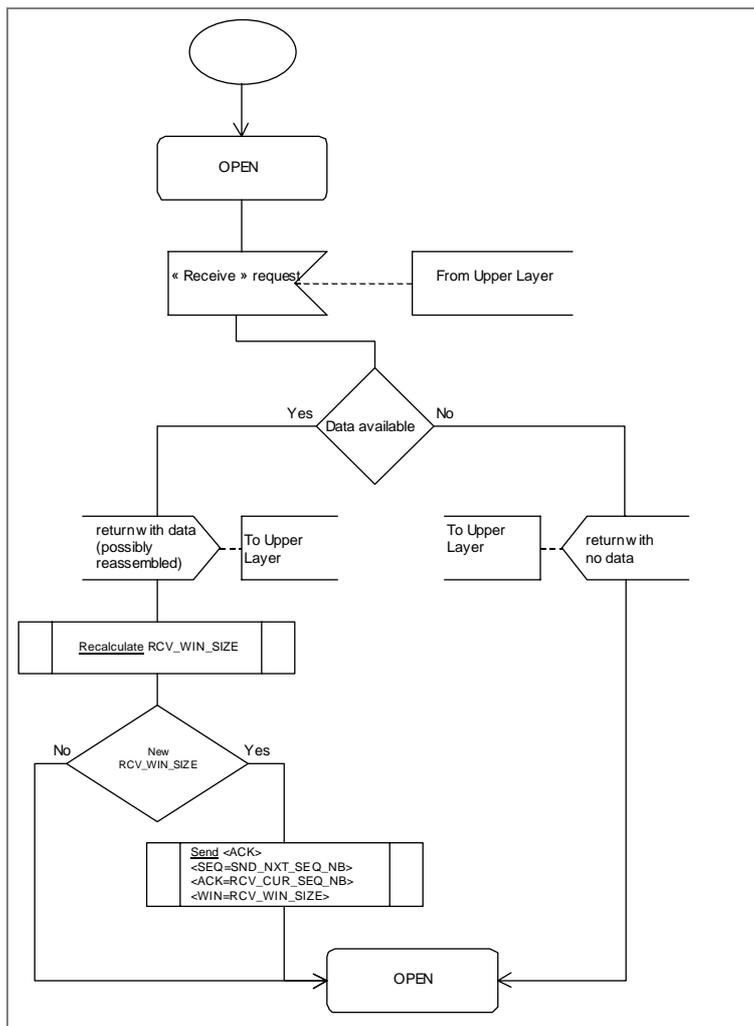
Figure 17: Receive request in OPEN state

Figure 18 describes the process of a "Receive request" in the "CLOSE" state.
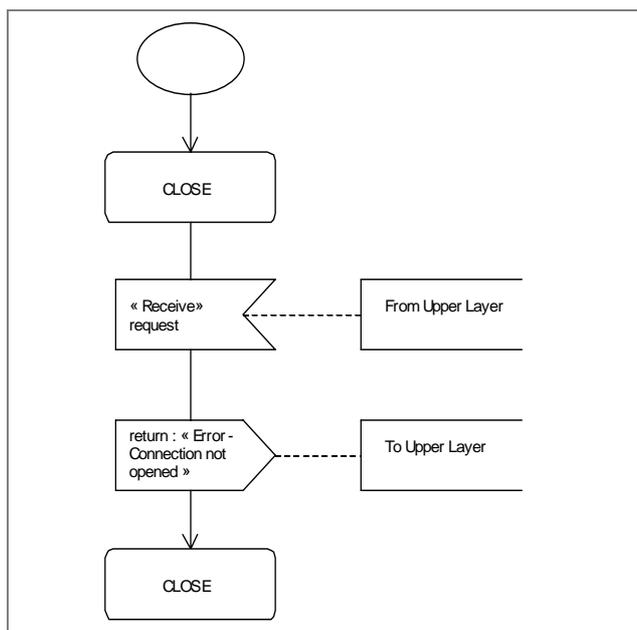
Figure 18: Receive request in CLOSE state

Figure 19 describes the process of a "Receive request" in the "CLOSE-WAIT" state.
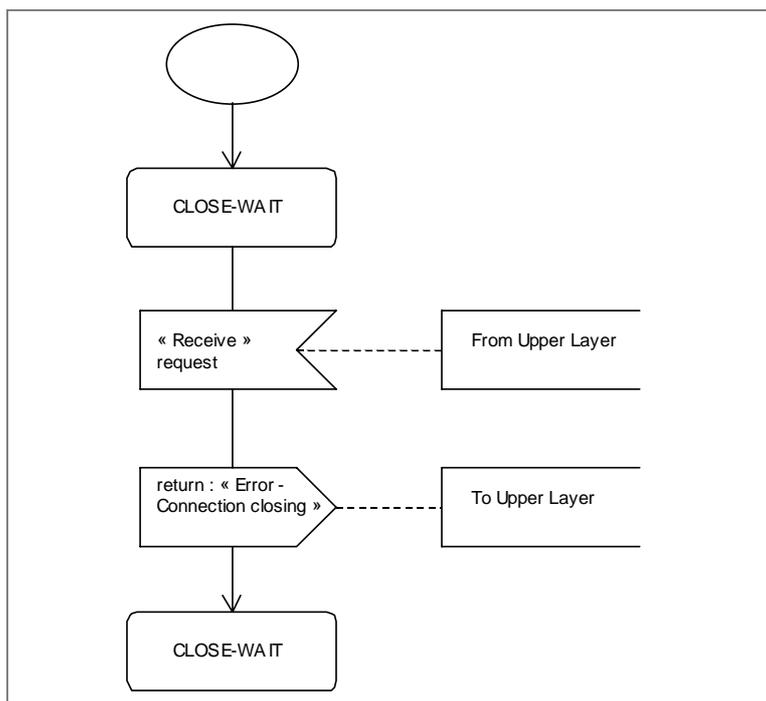
Figure 19: Receive request in CLOSE-WAIT state

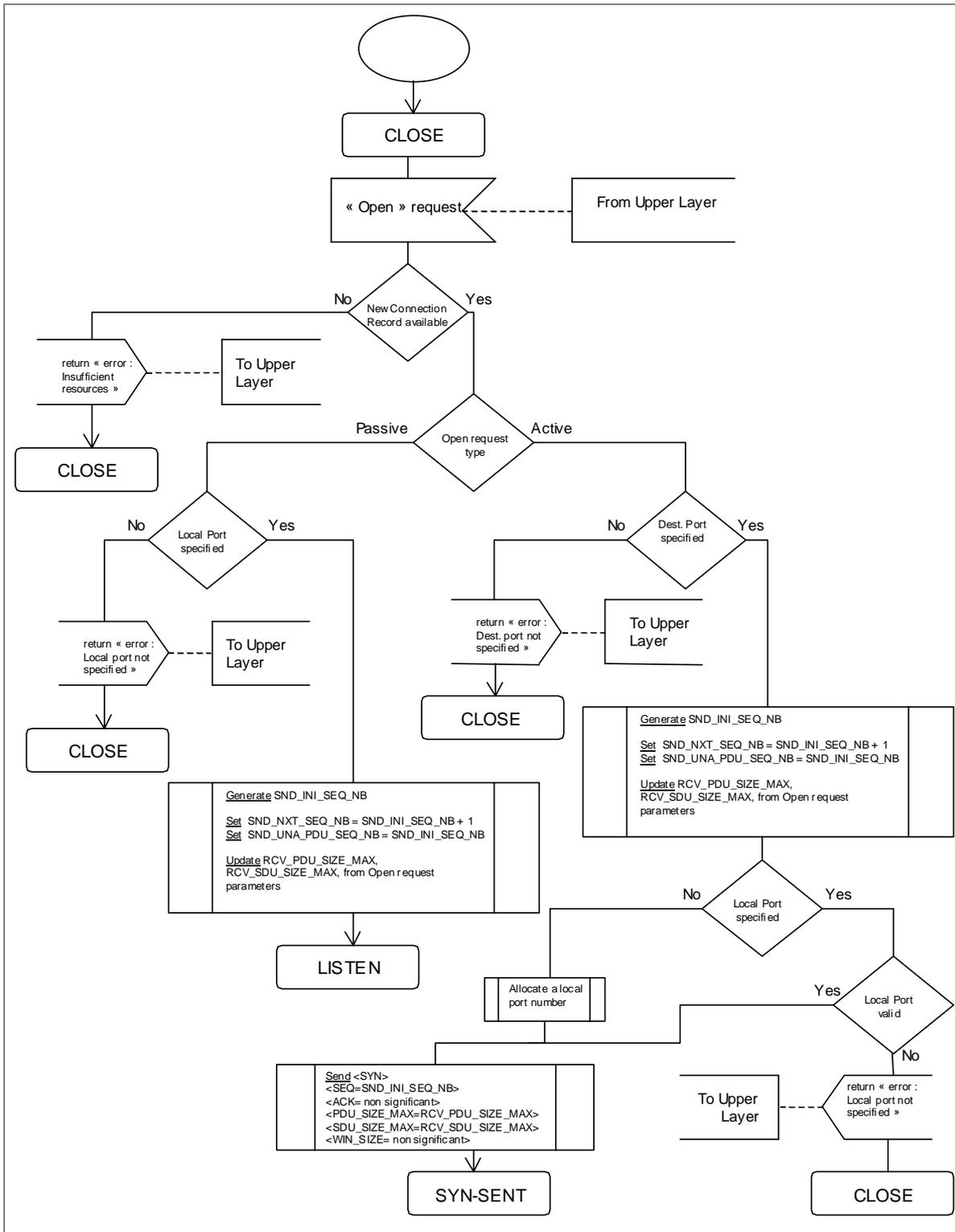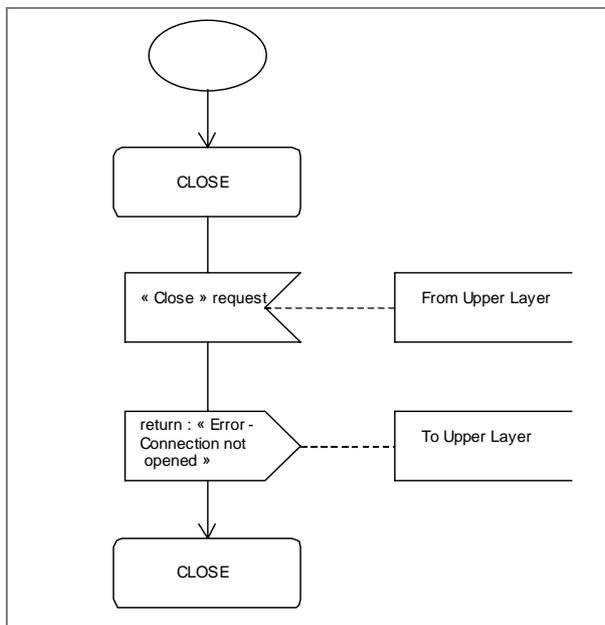Figure 20 describes the process of a "Receive request" in the "LISTEN", "SYN-SENT" or "SYN-RCVD" state.
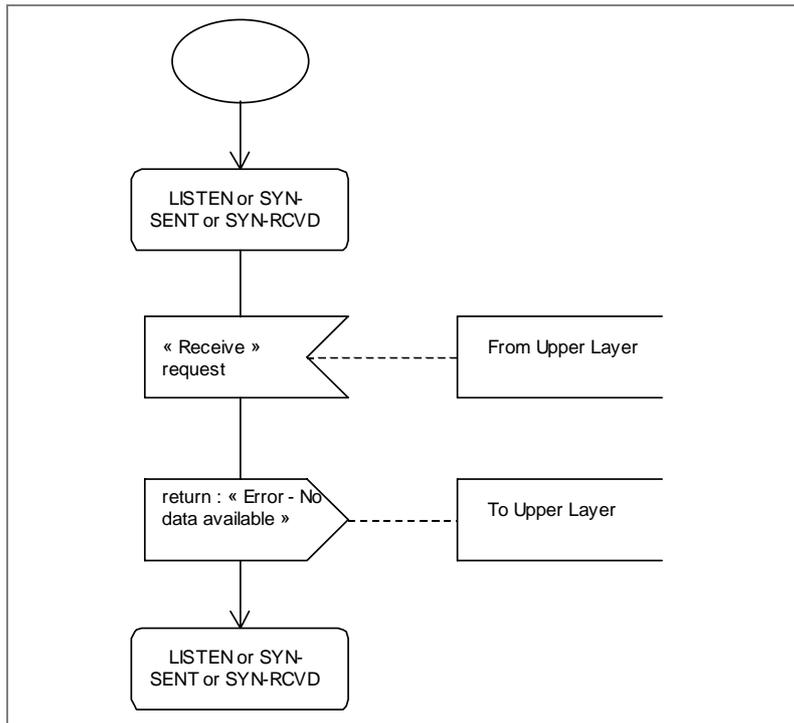
Figure 20: Receive request in LISTEN, SYN-SENT or SYN-RCVD state

5.4.1.4 Send request

If segmentation is needed, the given data will be checked (buffer size, etc.) then several PDUs will be produced and submitted to the transmission queue if it is accepted.

The following scenario describes the handling of a single PDU by the transmission queue.

Figure 21 describes the process of a "Send request" in the "OPEN" state.

The description does not take any reception activity happening in parallel into account. However, EACK PDUs that are generated by receive events may be merged with data PDUs generated by the sending activity, resulting in an EACK PDU containing data.
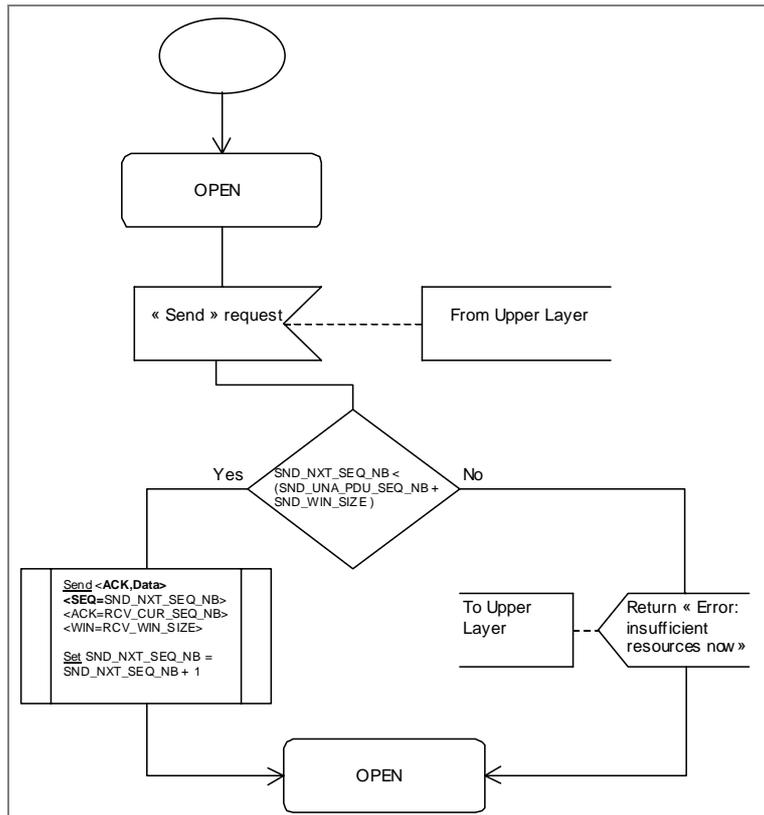
Figure 21: Send request in OPEN state

Figure 22 describes the process of a "Send request" in the "CLOSE", "LISTEN", "SYN-SENT" or "SYN-RCVD" state.
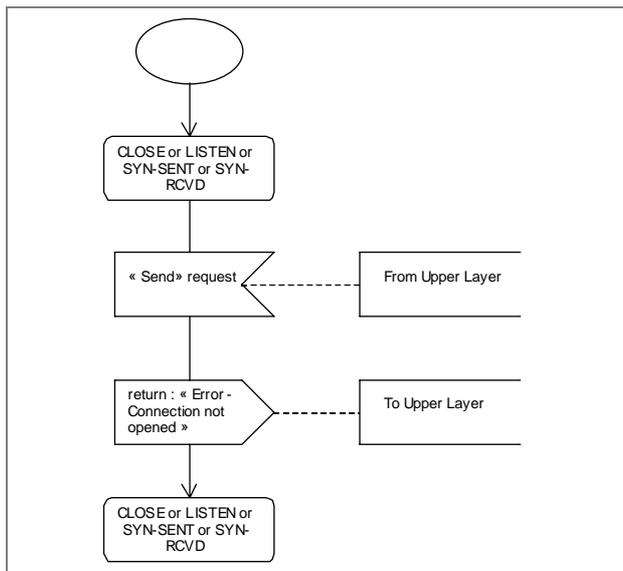
Figure 22: Send request in CLOSE, LISTEN, SYN-SENT or SYN-RCVD state

Figure 23 describes the process of a "Send request" in the "CLOSE-WAIT" state.
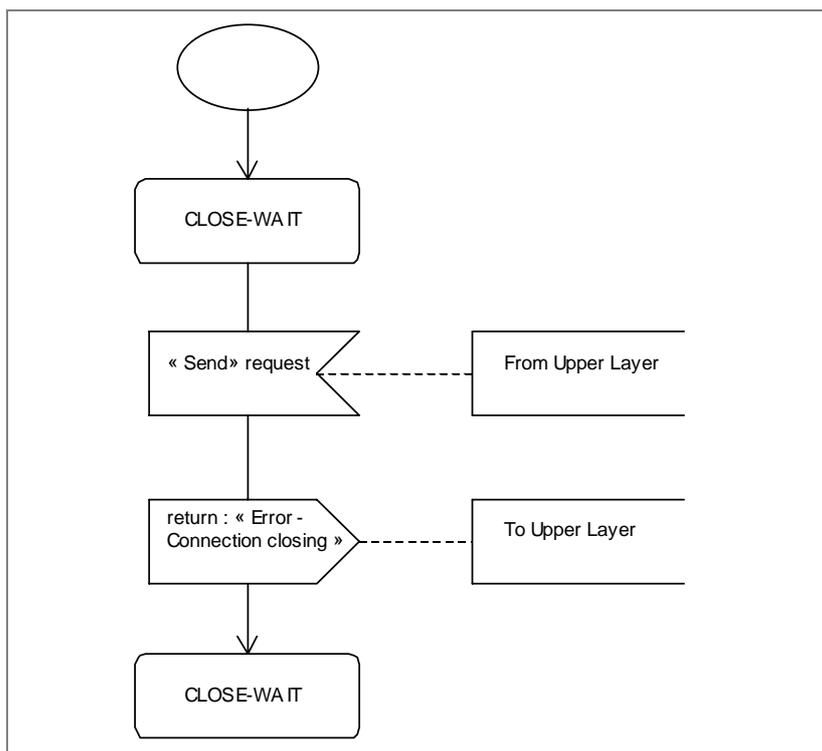
Figure 23: Send request in CLOSE-WAIT state

5.4.2   PDU arrival events

Figures 24 to 35 specify the behaviour of a CAT_TP entity upon PDUs reception (i.e. actions to perform and resulting state).The assumption is made that the PDU was addressed to the local port associated with the connection record.

5.4.2.1 Initial state: CLOSE
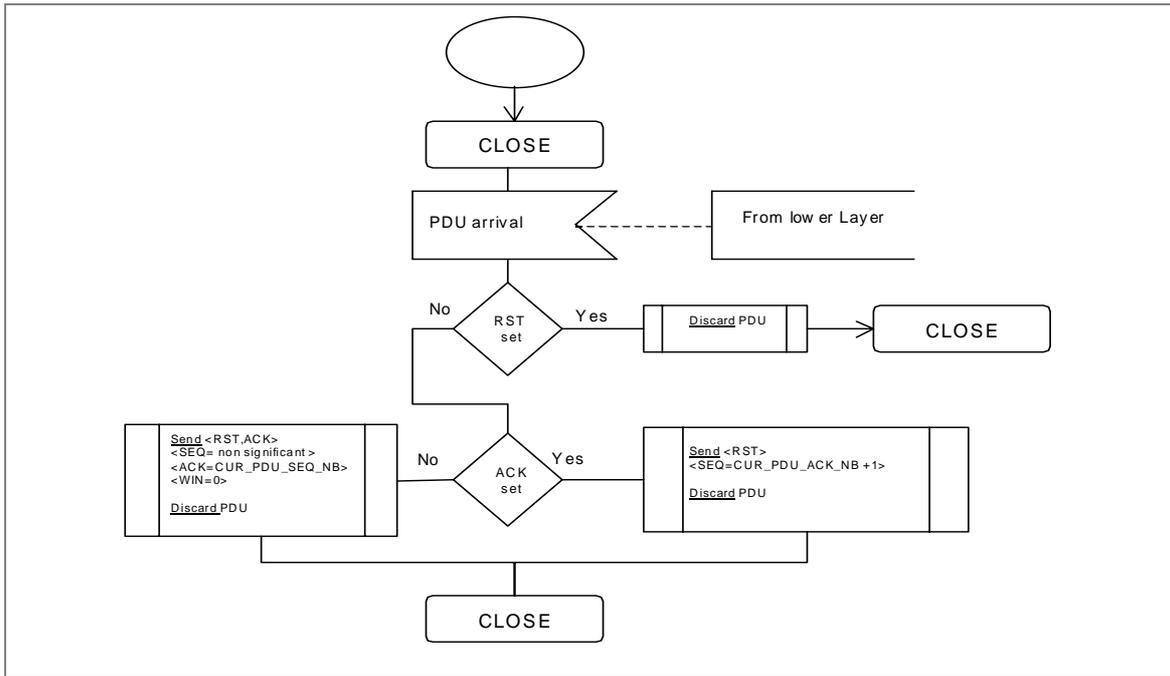
Figure 24: CLOSE state PDU arrival event processing

5.4.2.2 Initial state: OPEN
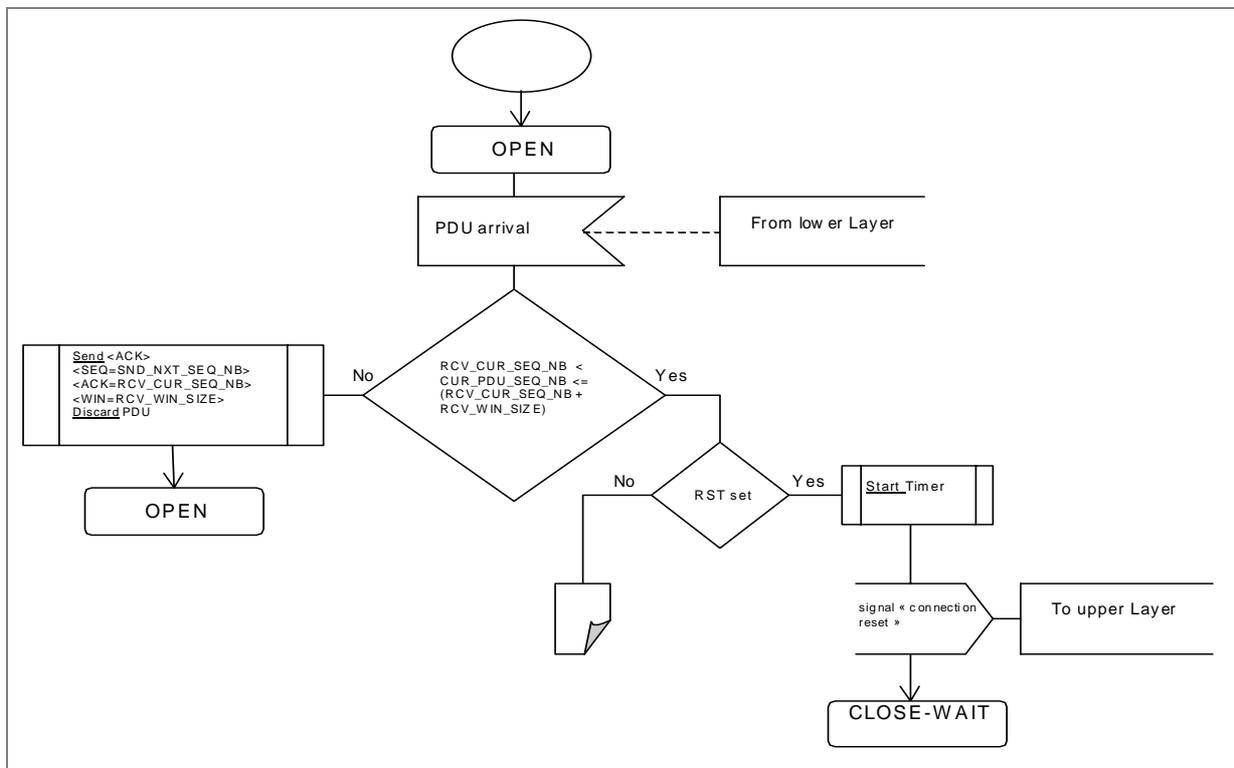
Figure 25: OPEN state PDU arrival event processing 1/4
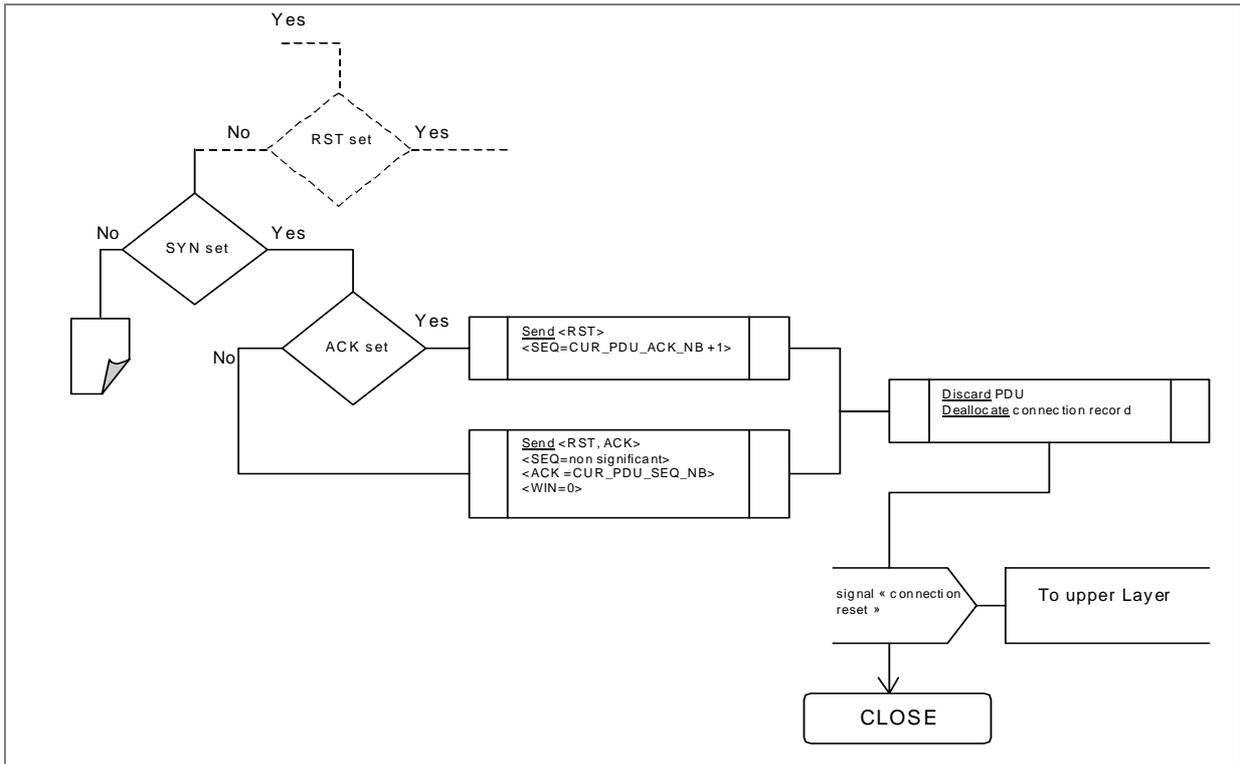
Figure 26: OPEN state PDU arrival event processing 2/4
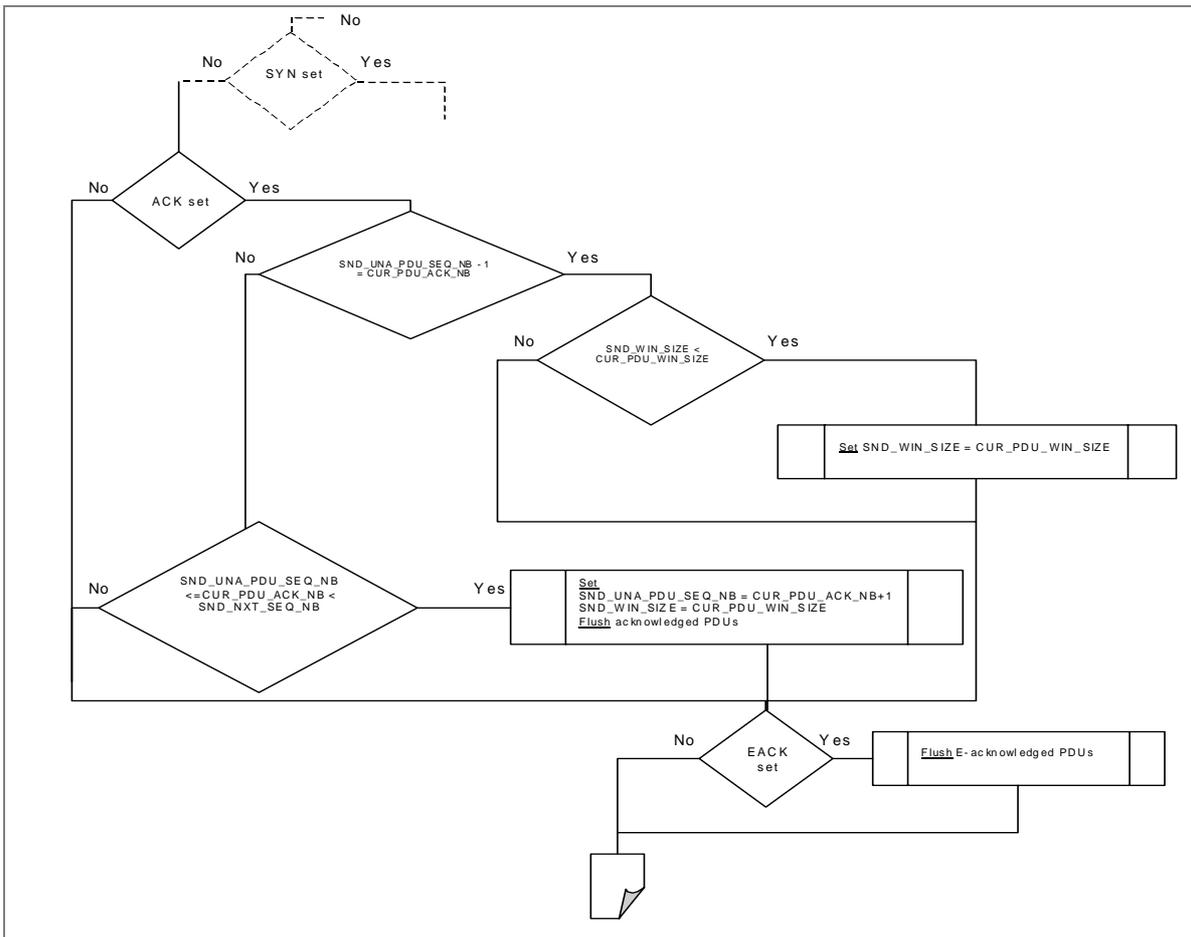
Figure 27: OPEN state PDU arrival event processing 3/4
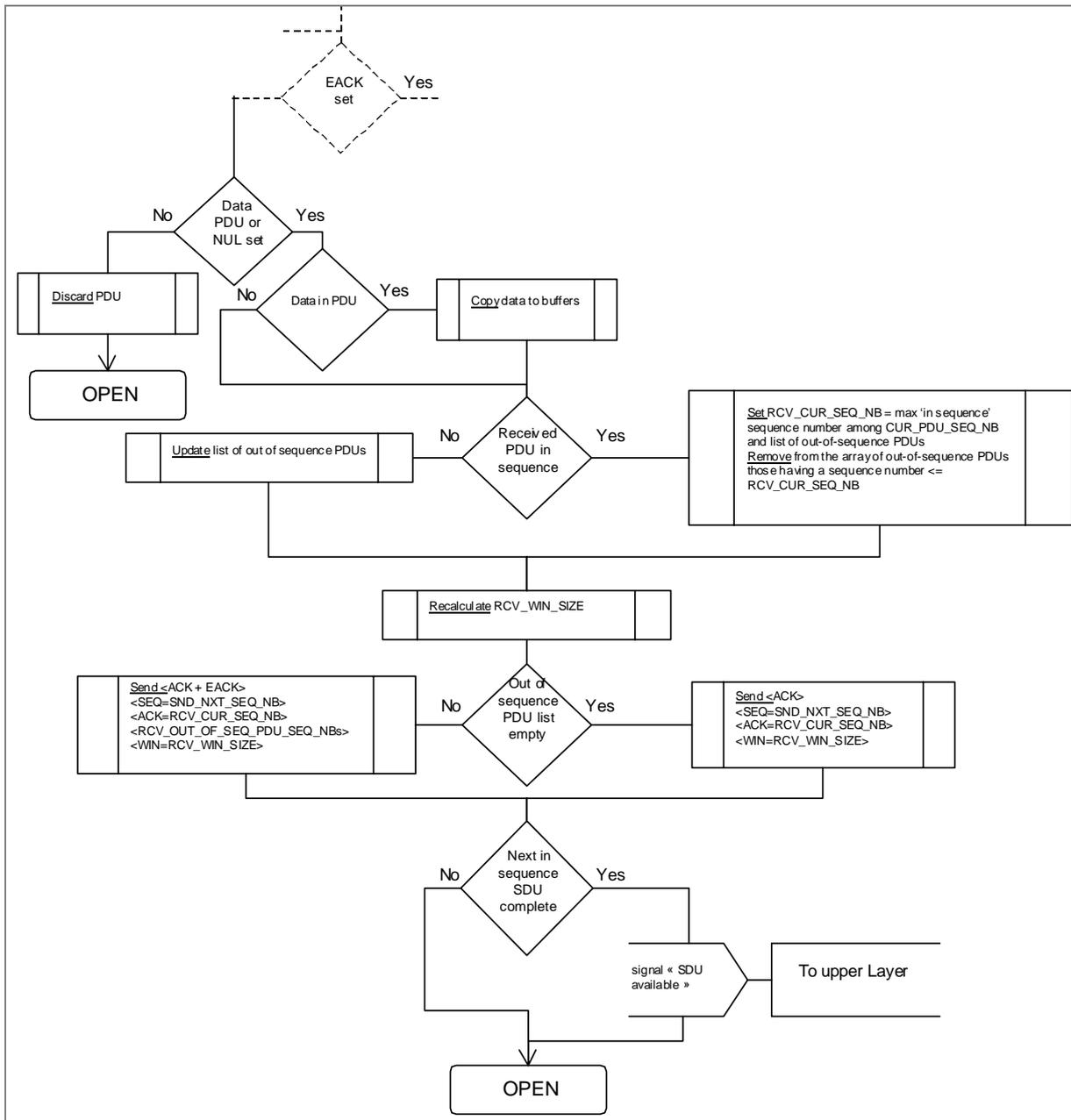
Figure 28: OPEN state PDU arrival event processing 4/4

5.4.2.3 Initial state: LISTEN
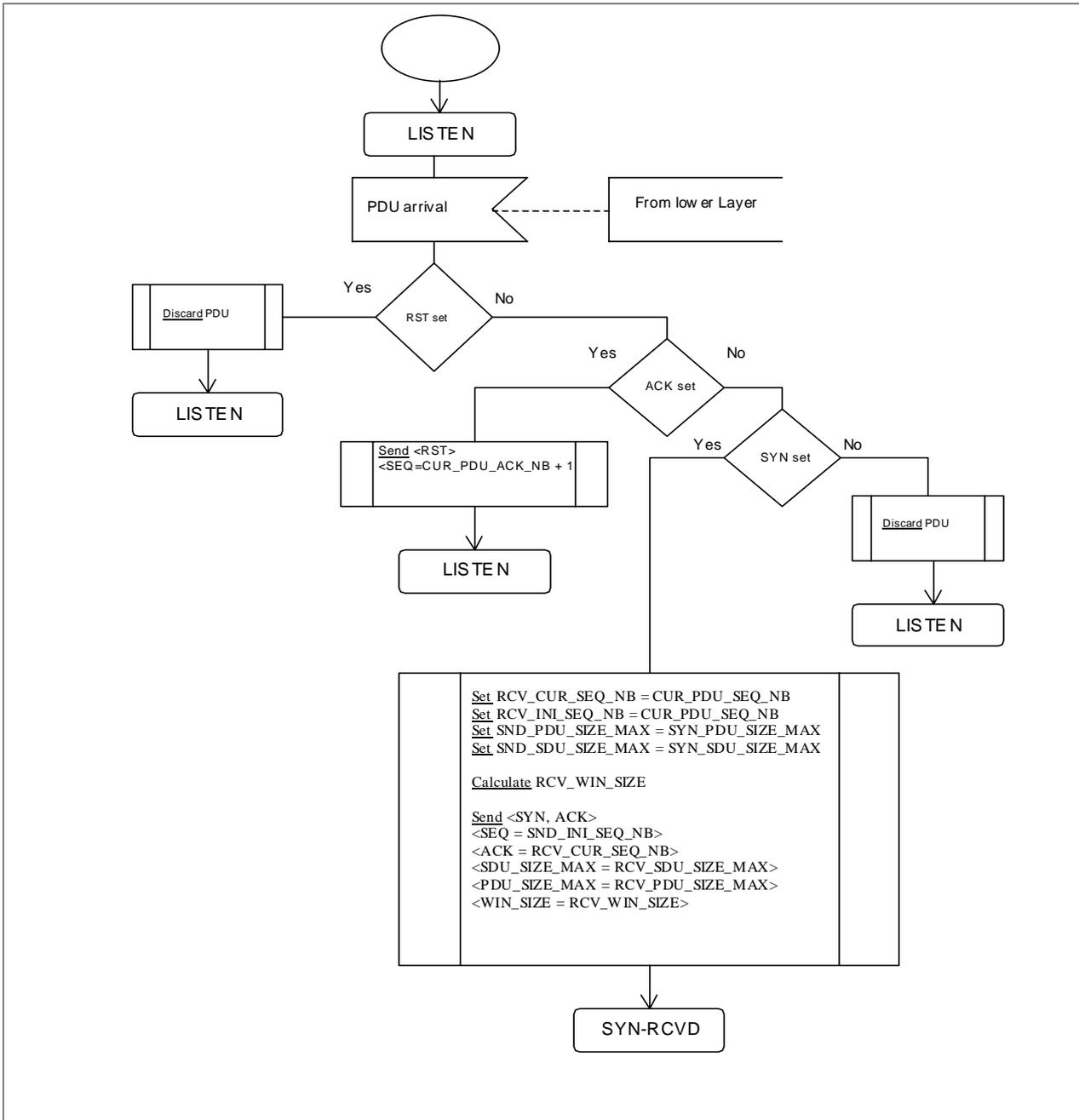
Figure 29: LISTEN state PDU arrival event processing

5.4.2.4 Initial state: SYN-SENT
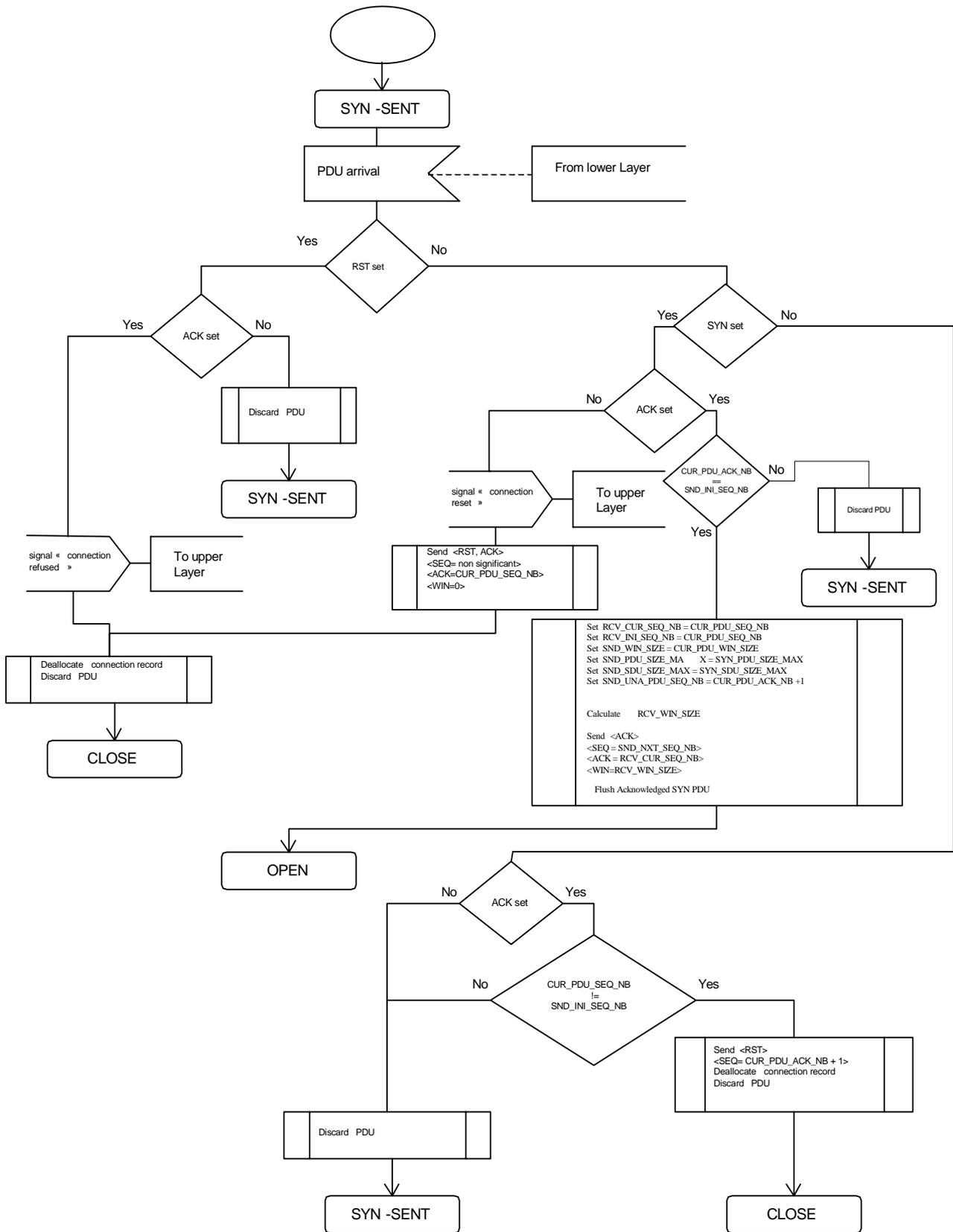
Figure 30: SYN-SENT state PDU arrival event processing

5.4.2.5 Initial state: SYN-RCVD
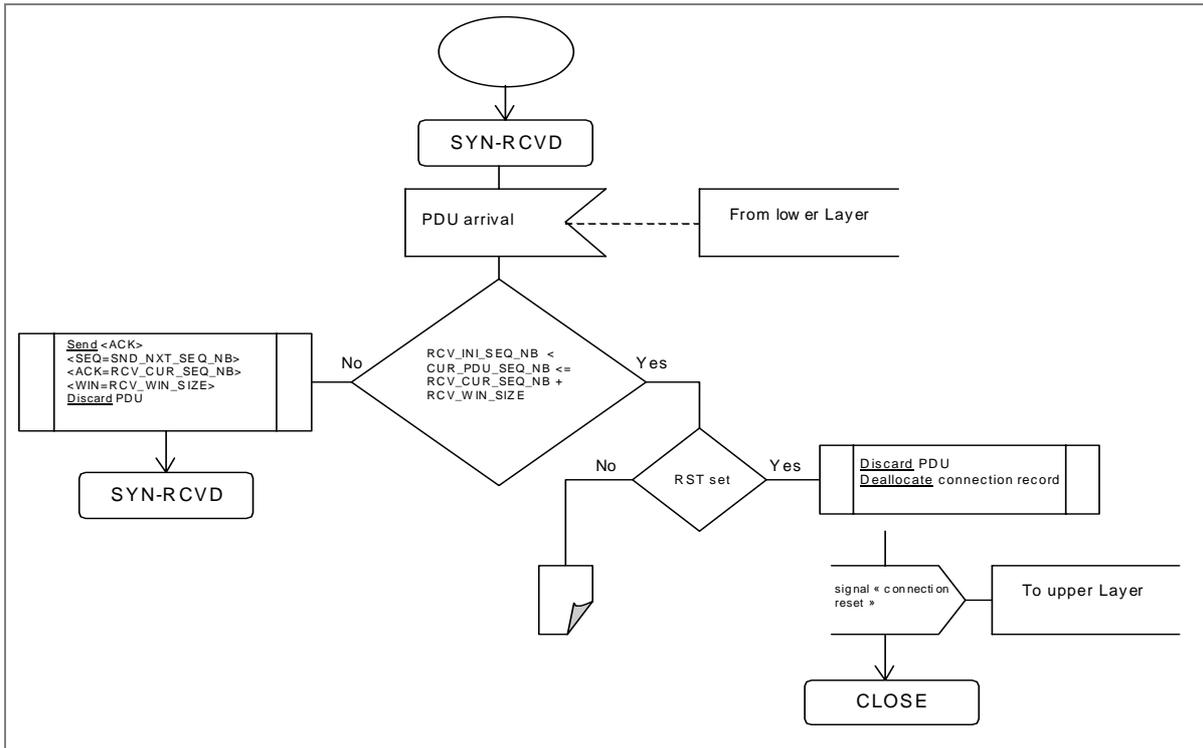
Figure 31: SYN-RCVD state PDU arrival event processing 1/4
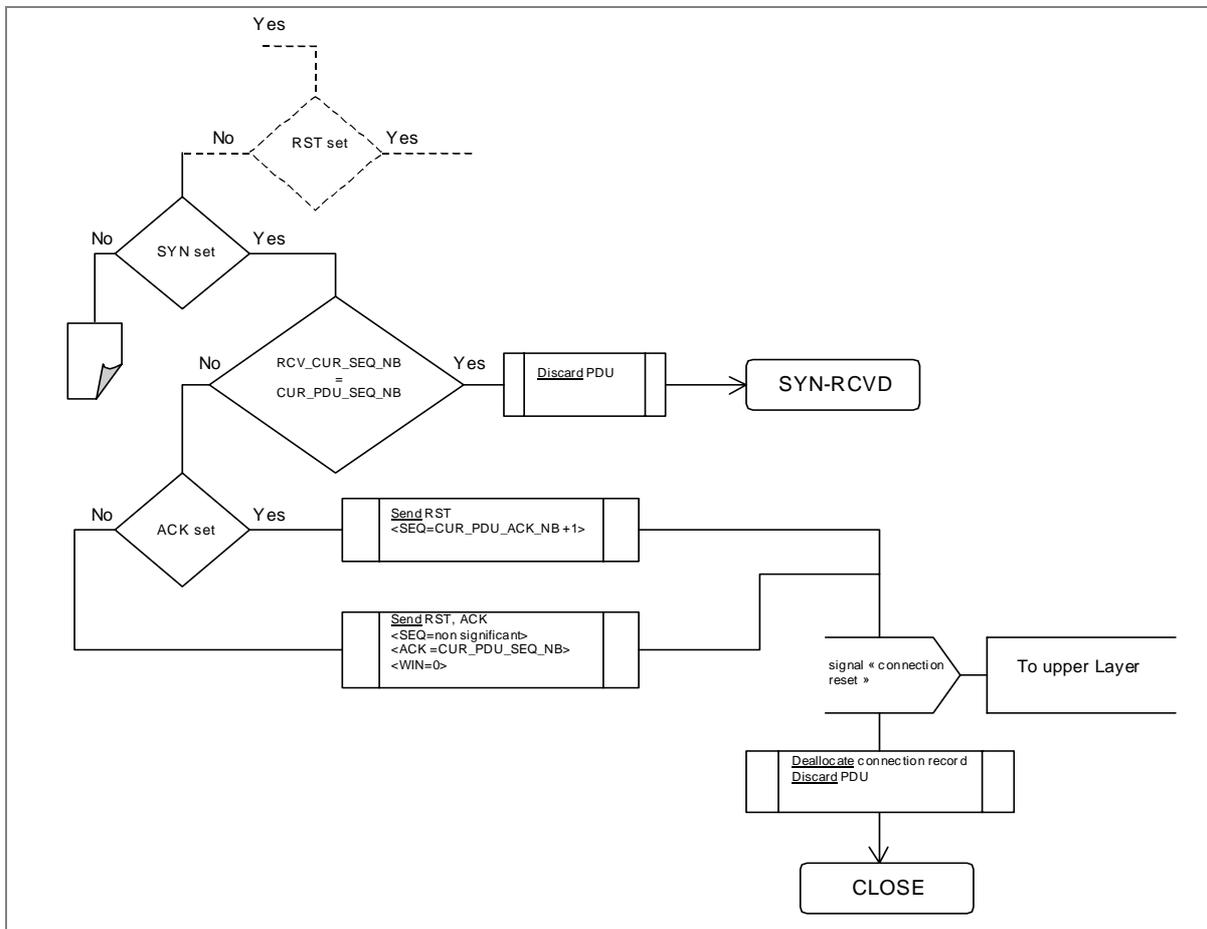
Figure 32: SYN-RCVD state PDU arrival event processing 2/4
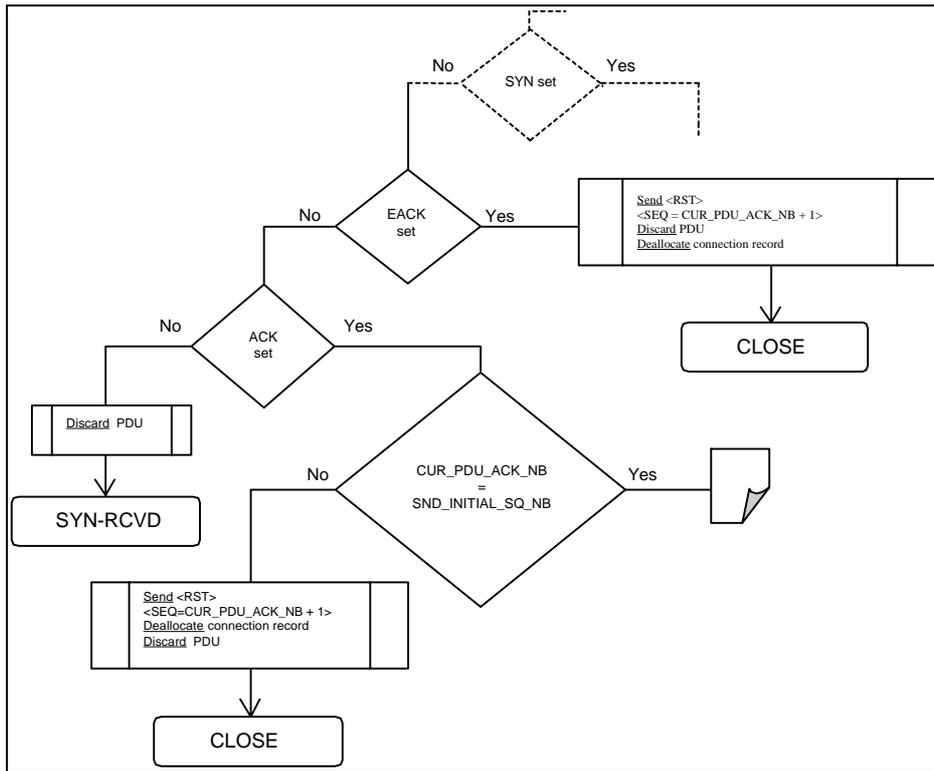
Figure 33: SYN-RCVD state PDU arrival event processing 3/4
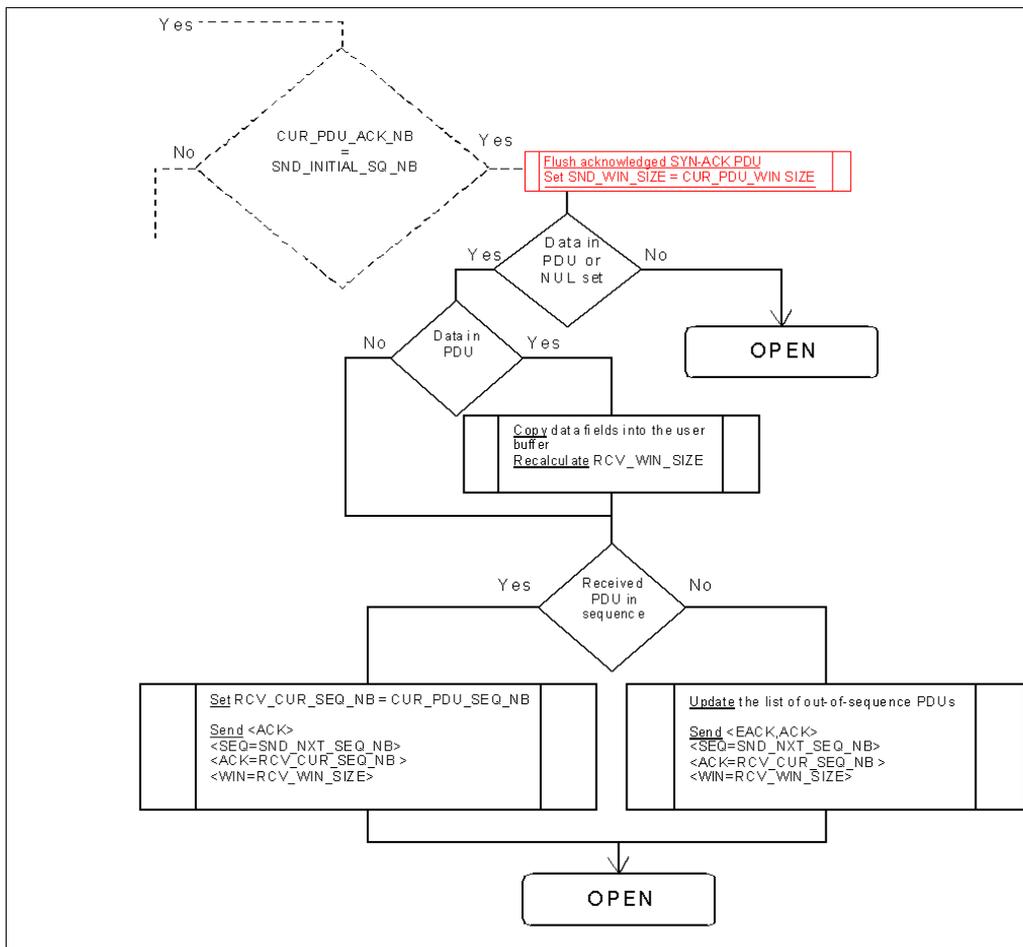
Figure 34: SYN-RCVD state PDU arrival event processing 4/4

5.4.2.6 Initial state: CLOSE-WAIT
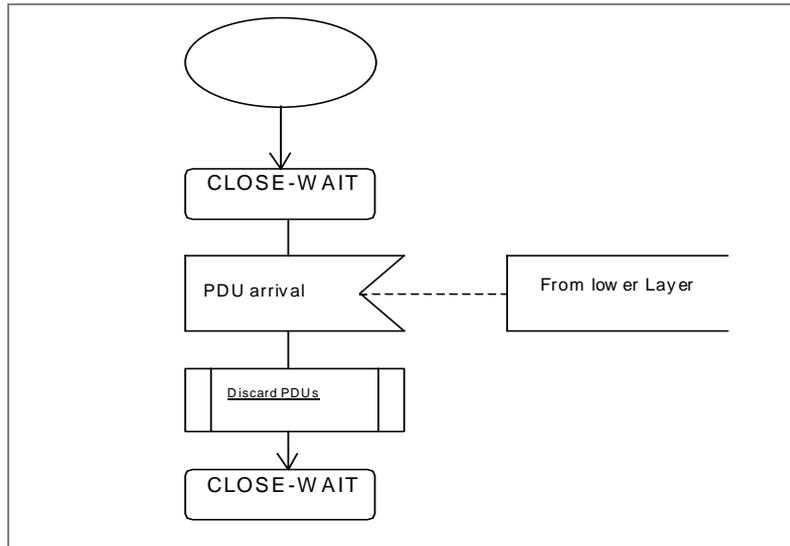
Figure 35: CLOSE-WAIT state PDU arrival event processing

5.4.3 Timeout events

Timeout events occur when a timer expires and signals the CAT_TP. Two types of timeout events can occur, as described below.

5.4.3.1 Retransmission timeout
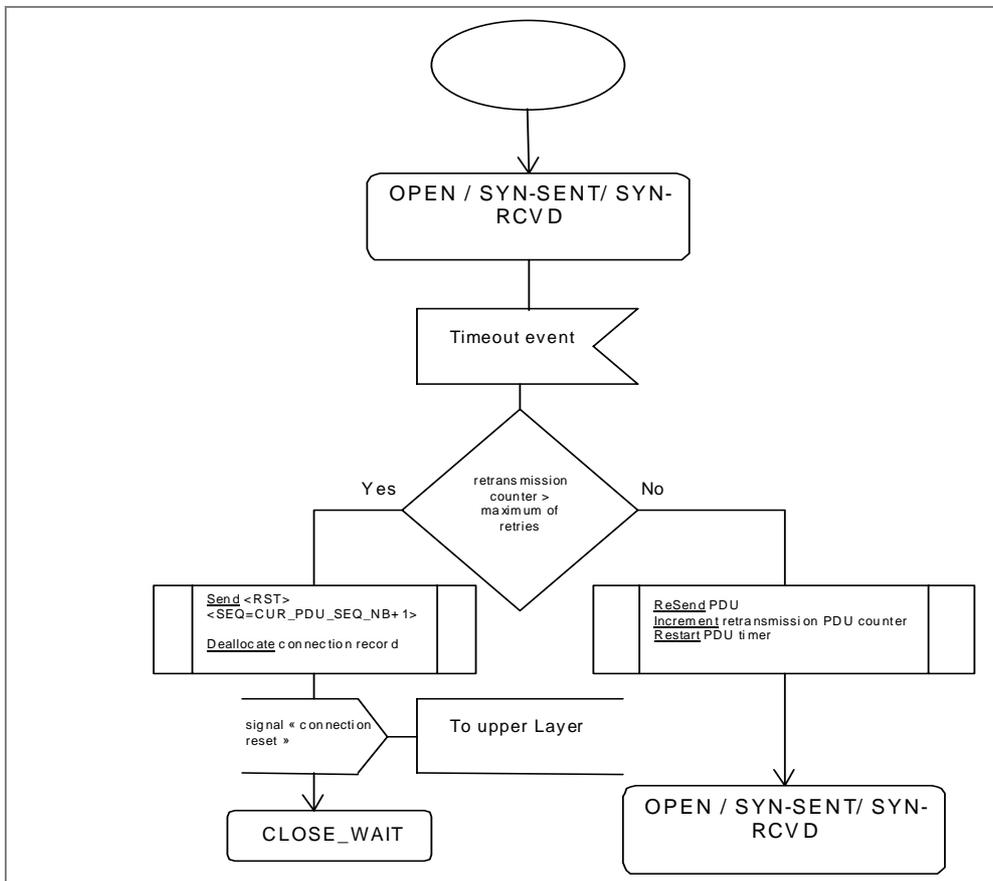
Figure 36: PDU retransmission timeout

5.4.3.2 Close-wait timeout
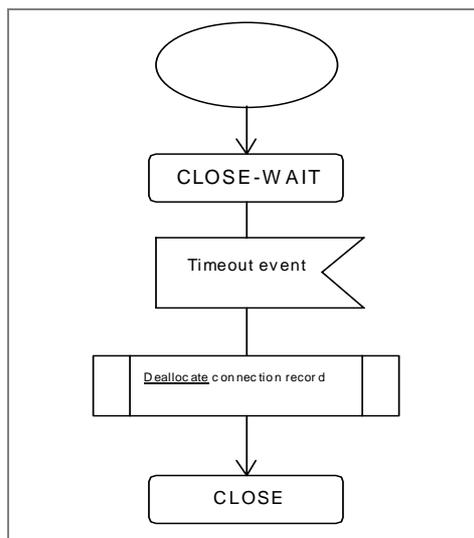
Figure 37: CLOSE-WAIT timeout

5.5 Identification

A LV structure is defined in the CAT_TP variable header to be used for identification of the CAT_TP entity during connection startup. It is up to the application to specify the syntax and semantics of the identification data. If the length is set to zero then no identification data is provided.

The coding is defined as follows:

Byte(s)	Description	Length
1	Length (x)	1
2 - (x + 1)	Identification data	x

5.6 CAT_TP header format

CAT_TP PDUs are based on this header.
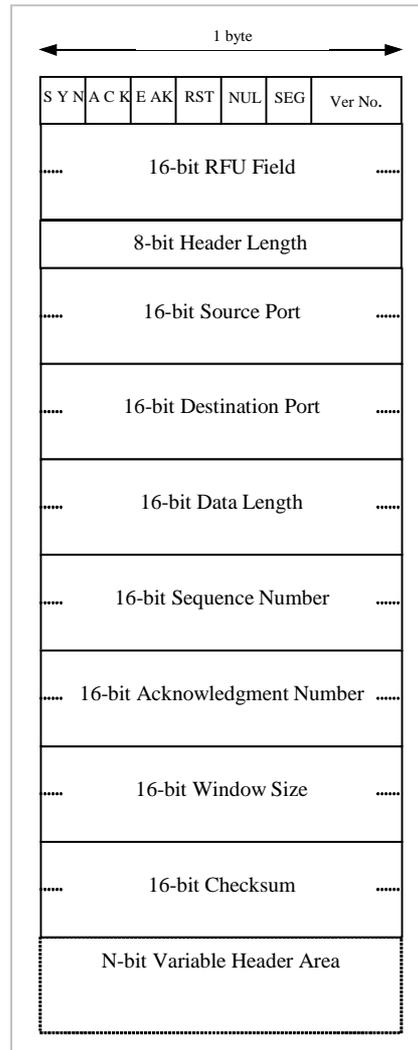
Figure 38: CAT_TP header format

5.6.1 First octet

This 8-bit field occupies the first octet of word one in the header. It is bit encoded with the following bits currently defined.

```
B8  b7  b6  b5  b4  B3  b2  b1
```

Version Number of CAT_TP
00: contains the version number of the protocol
    described by the present document. Current value
    is 00.

Segmentation sub layer bit
0: No segmentation or last Segmentation PDU (see
   clause 7)
1: Segmentation PDU

NULL PDU indicator
0: The current PDU is not a NULL PDU
1: The current PDU is a NULL PDU

RST PDU indicator
0: The current PDU is not a RST PDU
1: The current PDU is a RST PDU

EACK PDU indicator
0: The current PDU does not contain EACK information
1: The current PDU contains EACK information

ACK PDU indicator
0: The current PDU does not contain ACK information
1: The current PDU contains ACK information

SYN PDU indicator
0: The current PDU is not a SYN PDU
1: The current PDU is a SYN PDU

Bit Name	Description
SYN	Establish connection and synchronize sequence numbers.
ACK	Acknowledge and window fields significant.
EACK	Non-cumulative (Extended) acknowledgement.
RST	Reset the connection.
NUL	This is a null (zero data length) PDU.
SEG	Segmentation sub-layer header bit.
Version Nb	CAT_TP version number.

Figure 39: First octet bits description

The SYN, RST shall be sent as separate PDUs and shall not contain any data. The ACK may accompany any message. The NUL PDU shall have a zero data length, but may be accompanied by ACK and EACK information. The SEG bit may only be set for PDUs containing data.

5.6.2 Header length

The length of the CAT_TP header in units of octet, including this field. This field allows CAT_TP to find the start of the Data field. This field is 8 bits length. For a PDU with no variable header section, the header length field has the value '12'.

5.6.3 Source and destination ports

The source and destination ports are used to identify CAT_TP entities that are communicating with each other. The combination of the port identifiers with the source and destination identifiers of the lower layer serves to fully qualify the connection and constitutes the connection identifier. This permits CAT_TP to distinguish multiple connections between two entities running CAT_TP. Each field shall be 16-bits in length, allowing port numbers from '0000' to 'FFFF'.

5.6.4   Data length

The length in octets of the data in a PDU. The data length shall not include the CAT_TP header. This field shall be 16-bits in length.

5.6.5   Sequence number

This field indicates the sequence number of a PDU. This field shall be 16-bits in length. The sequence number is a cyclic parameter and may be incremented (only by one) from '0000' to 'FFFF'.

5.6.6   Acknowledgement number

If the ACK bit is set in the header, this field shall contain the sequence number of the last correctly and in sequence received PDU. Once the connection is established, the ACK bit and this field shall always be set. This field shall be 16-bits in length.

5.6.7   Window size

If the ACK bit is set in the header, this field shall contain the maximum numbers of PDUs that the other entity is allowed to send, relative to the Acknowledgement Number. The last allowed Sequence Number is the Acknowledgement Number plus Window Size.

This field reflects flow control and window management as described in clause 5.3.3.

5.6.8   Checksum

This field contains the result of the checksum calculation described in clause 5.3.2.2.

5.6.9   Variable header area

This area shall be used to transmit parameters for the SYN and EACK PDUs.

5.6.10   RFU field

This area is reserved for future use.

5.7 SYN PDU
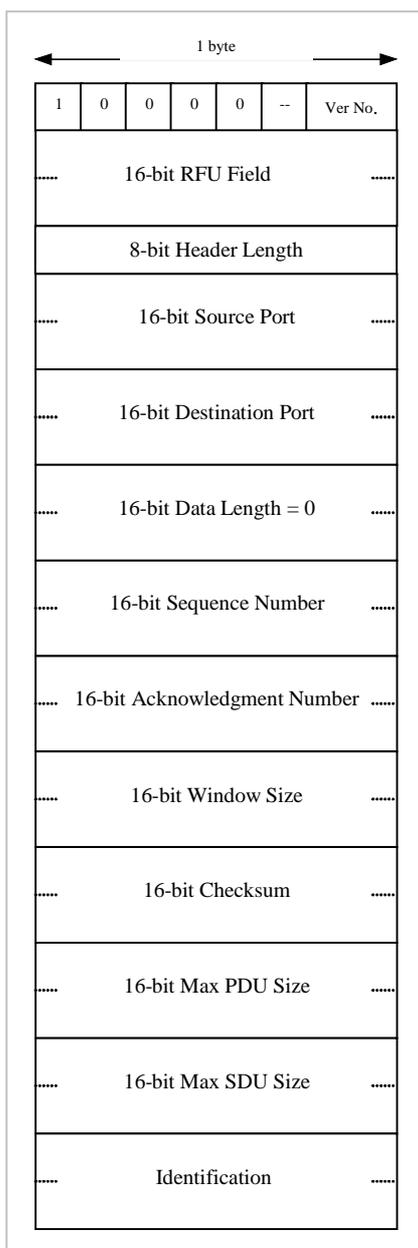
Figure 40: SYN PDU structure

5.7.1 SYN PDU fields

5.7.1.1 Data length

This SYN shall not carry any data. This length is coded to zero (0).

5.7.1.2 Sequence number

This field shall contain the initial sequence number selected for this CAT_TP connection.

NOTE: Once the SYN PDU has been sent, the sequence number of the CAT_TP entity is increased by one.

5.7.1.3 Acknowledgment number

This field is valid only if the ACK flag is set. In that case, this field shall contain the sequence number of the SYN PDU received from the other CAT_TP entity.

5.7.1.4 Maximum PDU size

This field contains the maximum PDU size in octets that the CAT_TP entity (issuing the SYN PDU) supports in reception. This information shall be used by the peer CAT_TP entity to set its maximum PDU size for its further emission within this connection. The specified size includes the CAT_TP header and data.

> NOTE: This parameter is a consequence from the possible constraint of the lower layer (e.g. UDP maximum PDU size).

5.7.1.5 Maximum SDU size

This field contains the maximum SDU size in octets that the CAT_TP entity (issuing the SYN) supports in reception. This information shall be used by the peer CAT_TP entity to set its maximum SDU size for its further emission within this connection. The specified size does not include any CAT_TP headers and shall not change during the connection lifetime.

5.7.1.6 Identification

This field contains an identification value as described in clause 5.5.

5.8 ACK PDU
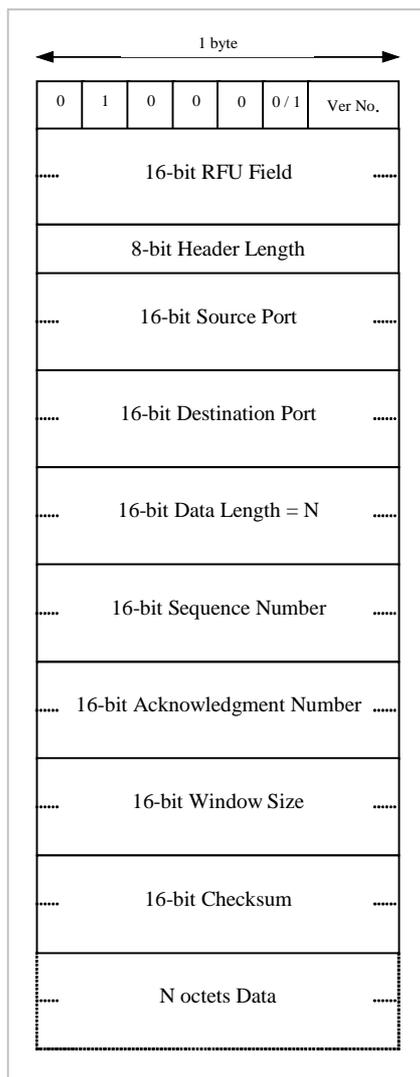
Figure 41: ACK PDU structure

The ACK PDU is used to acknowledge in-sequence CAT_TP PDUs. The ACK PDU may be sent as a separate PDU, but it should be combined with data whenever possible.

The ACK PDU plays the role of the DATA PDU, data being segmented or not. Therefore, any emission of data by a CAT_TP entity is done through a ACK PDU.

The ACK PDU acknowledging the SYN ACK PDU during a connection set-up shall not carry any data.

5.8.1 ACK PDU field

5.8.1.1 Data length

A "Data Length" field equal to zero indicates that the ACK segment does not carry any data after the header.

A non-zero "Data Length" field indicates that there is data carried after the header, in the "Data" field.

5.8.1.2 Sequence number

The "Sequence Number" field contains the sequence number of the CAT_TP entity sending the ACK PDU.

5.8.1.3 Acknowledgment number

The "Acknowledgement Number" field indicates the sequence number of the last in-sequence CAT_TP PDU previously received, and acknowledges therefore all the PDUs having a Sequence Number up to the value of the Acknowledgement Number.

5.9 EACK PDU
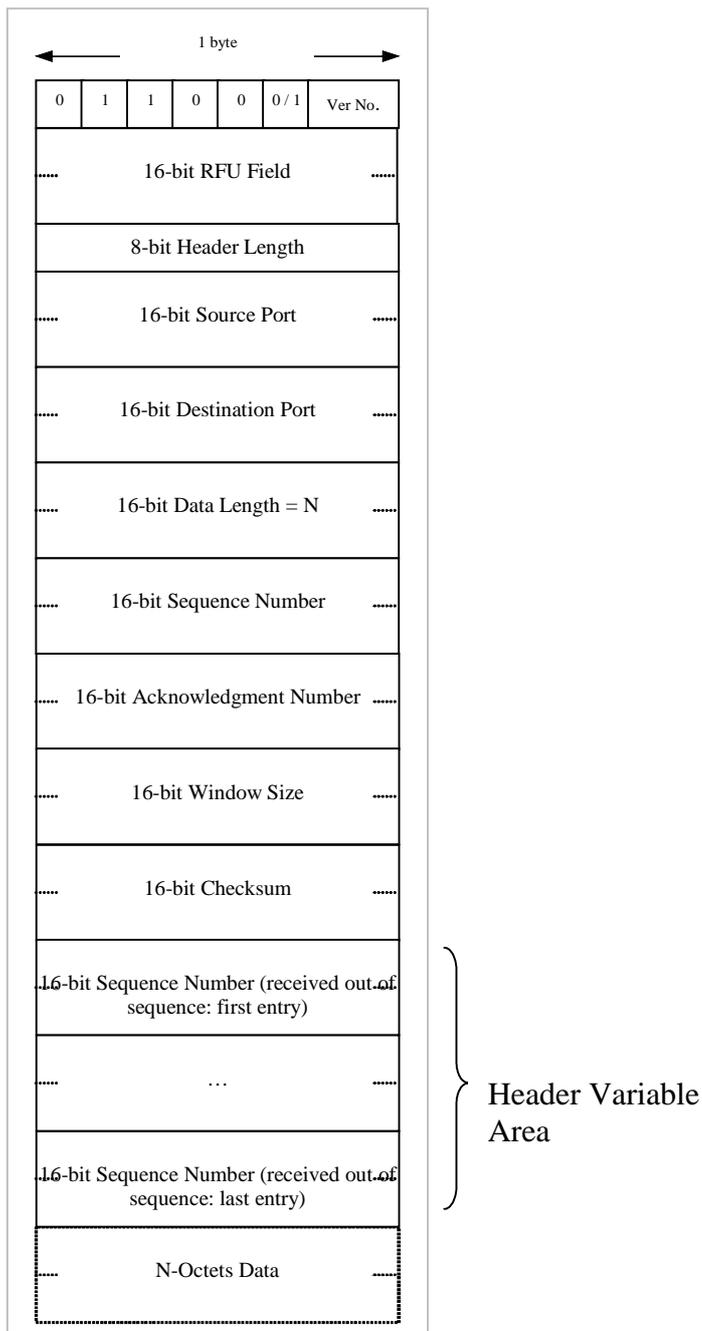
Figure 42: EACK PDU structure

The EACK PDU is used to acknowledge PDUs received out of sequence by the receiving CAT_TP entity. It contains in the header variable area the sequence numbers of one or more PDUs received with a correct checksum, but out of sequence.

The EACK shall always be combined with the ACK parameter in the PDU, thus providing in the same time the sequence number of the last PDU received in sequence.

If the EACK information plus the data to be sent exceed the size of one PDU, an EACK PDU without data shall be sent before the Data, which shall be sent after this in an ACK PDU.

5.9.1 EACK PDU Field

5.9.1.1 Data length

A "Data Length" field equal to zero indicates that the EACK segment does not carry any data after the header.

A non-zero "Data Length" field indicates that there is data carried after the header, in the "Data" field.

5.9.1.2 Sequence number

The "Sequence Number" field contains the sequence number of the CAT_TP entity sending the EACK PDU.

5.9.1.3 Acknowledgment number

The "Acknowledgement Number" field indicates the sequence number of the last in-sequence CAT_TP PDU previously received, and acknowledges therefore all the PDU having a Sequence Number up to the value of the Acknowledgement Number.

5.9.1.4 Variable header area

This area in the EACK CAT_TP PDU is dedicated to the sequence numbers of the PDUs received with a correct checksum and out of sequence. There may therefore be several sequence numbers provided in this area, each being 16-bit long. The EACK sequence numbers order is not relevant.

Obviously, the sequence numbers in this variable header area shall be greater than the acknowledgement number provided in the fixed header area (clause 5.6.5, Sequence Number description).

5.10 RST PDU
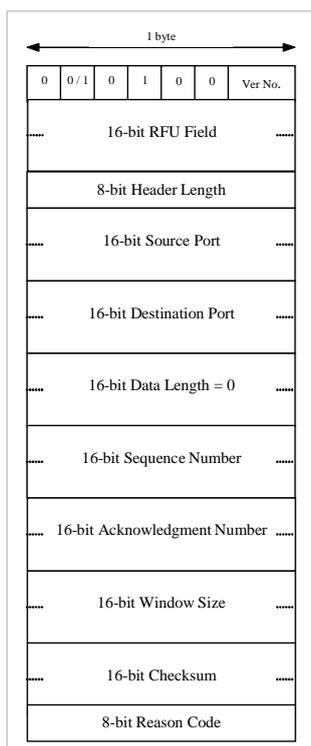
Figure 43: RST PDU structure

The RST PDU is used to close or reset a connection. Upon reception of an RST PDU, the sender shall stop the emission of any CAT_TP PDU on this connection, and shall abort the un-serviced requests.

A RST PDU shall not include any data, but may include an acknowledgement and therefore combine with an ACK PDU It shall not be combined with an EACK PDU.

5.10.1 RST PDU fields

5.10.1.1 Data length

The RST PDU shall not carry any data. The "Data Length" field is therefore set to zero.

5.10.1.2 Sequence number

The "Sequence Number" field contains the sequence number of the CAT_TP entity sending the RST PDU.

5.10.1.3 Acknowledgment number

The "Acknowledgement Number" field indicates the sequence number of the last in-sequence CAT_TP PDU previously received, and acknowledges therefore all the PDU having a Sequence Number up to the value of the Acknowledgement Number.

5.10.1.4 Reason code

The Reason code shall indicate to the other side, why the connection is terminated, especially during connection establishment.

The following values are defined:

- '00' Normal ending;

- '01'   Connection set-up failed, no details given;

- '02'   Temporarily unable to set up this connection;

- '03'   Requested Port not available;

- '04'   Unexpected PDU received;

- '05' .. 'FF'   RFU.

5.11   NUL PDU
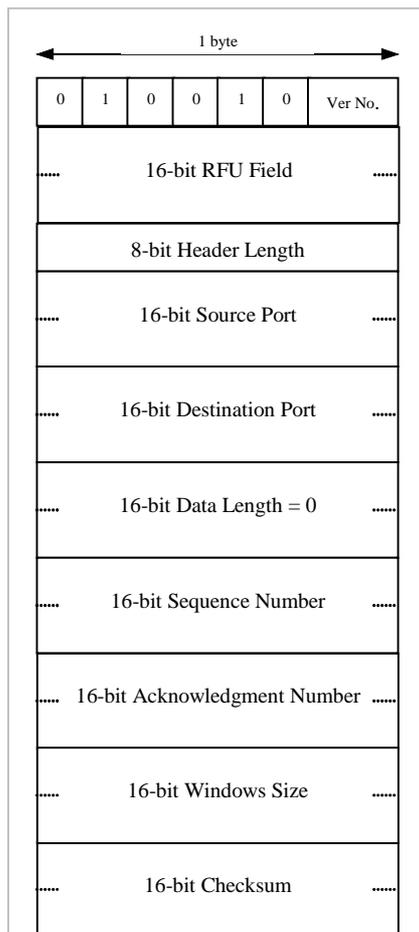
Figure 44: NUL PDU structure

The NUL PDU is used to determine if the other side of a CAT_TP connection is still active. When a NUL PDU is received, an CAT_TP entity shall acknowledge it if a valid connection exists. The NUL PDU is then discarded. The NUL PDU shall be combined with an ACK PDU but shall never carry user data. The NUL PDU shall not be combined with the EACK PDU.

5.11.1   NUL PDU fields

5.11.1.1   Data length

The NUL PDU shall not carry any data. The "Data Length" field is therefore set to zero (0).

5.11.1.2   Sequence number

The "Sequence Number" field contains the sequence number of the CAT_TP entity sending the NUL PDU.

5.11.1.3 Acknowledgment number

The "Acknowledgement Number" field indicates the sequence number of the last in-sequence CAT_TP PDU previously received, and acknowledges therefore all the PDU having a Sequence Number up to the value of the Acknowledgement Number.

5.12 Header flags combinations

Parameters⇨ PDU types ⇩	SYN	ACK	EACK	RST	NUL	SEG	Seq Nb	Ack Nb	Optional Header Area	Data
SYN PDU	= 1	= 0	= 0	= 0	= 0	= 0	SND_INITIAL_SEQ_NB	Insignificant	Max PDU Size Max SDU Size Identification	No Data
SYN/ACK PDU	= 1	= 1	= 0	= 0	= 0	= 0	SND_INITIAL_SEQ_NB	RCV_INITIAL_SEQ_NB	Max PDU Size Max SDU Size Identification	No Data
ACK PDU	= 0	= 1	= 0	= 0	= 0	= 0	insignificant	CUR_PDU_SEQ_NB	None	No data
ACK PDU + un-segmented data	= 0	= 1	= 0	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	SDU
ACK PDU + segmented data	= 0	= 1	= 0	= 0	= 0	= 1 = 0 if last segmented PDU	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	SDU Segment
ACK/EACK PDU	= 0	= 1	= 1	= 0	= 0	= 0	insignificant	CUR_PDU_SEQ_NB	RCV_OUT_OF_SEQ_PDU_SEQ_NBs	No data
ACK/EACK PDU + un-segmented data	= 0	= 1	= 1	= 0	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	RCV_OUT_OF_SEQ_PDU_SEQ_NBs	SDU
RST PDU	= 0	= 0	= 0	= 1	= 0	= 0	SND_NEXT_SEQ_NB	Insignificant	None	No data
RST/ACK PDU	= 0	= 1	= 0	= 1	= 0	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	No data
NUL/ACK PDU	= 0	= 1	= 0	= 0	= 1	= 0	SND_NEXT_SEQ_NB	CUR_PDU_SEQ_NB	None	No data

Figure 45: PDU Flags possible combinations

6 Implementation on BIP

6.1 Sending and receiving data

One CAT_TP PDU shall be transmitted as exactly one packet SDU on the BIP channel without adding or deleting any data. This may require one or more SEND DATA/RECEIVE DATA command to transfer the PDU to/from the Terminal Equipment.

6.2 Timers

Accuracy of timers shall be considered. Timers shall be considered for Acknowledgment and retransmission features.

Annex A (informative):
Scenarios examples

A.1 Connection establishment

This is an example of a connection being established between Host A and Host B. Host B has done a passive OPEN and is in LISTEN state. Host A does an active Open to establish the connection.

```
               Host A                          Host B

    Time  State                                    State

    1.    CLOSED                                   LISTEN

    2.    SYN-SENT     <SEQ=100><SYN> --->

    3.                            <--- <SEQ=200><ACK=100><SYN,ACK>
                                                  SYN-RCVD

    4.    OPEN    <SEQ=101><ACK=200> --->          OPEN

    5.      <SEQ=101><ACK=200><Data> --->

    6.                            <--- <SEQ=201><ACK=101>
```

A.2 Lost PDUs

This is an example of what happens when a PDU is lost. It shows how PDUs can be acknowledged out of sequence and that only the missing PDU need be retransmitted. Note that in this and the following examples "EA" stands for "Out of Sequence Acknowledgement".

```
    Time   Host A                      Host B

    1.     <SEQ=100><ACK=200><Data>  --->

    2.                          <--- <SEQ=201><ACK=100>

    3.     <SEQ=101><ACK=200><Data> (PDU lost)

    4.

    5.     <SEQ=102><ACK=200><Data>  --->

    6.                          <--- <SEQ=201><ACK=100><EA=102>

    7.     <SEQ=103><ACK=200><Data>  --->

    8.                          <--- <SEQ=201><ACK=100><EA=102,103>

    9.     <SEQ=101><ACK=200><Data>  --->

    10.                         <--- <SEQ=201><ACK=103>

    11.    <SEQ=104><ACK=200><Data>  --->

    12.                         <--- <SEQ=201><ACK=104>
```

A.3 PDUs received out of order

This is an example of PDUs received out of order. It further illustrates the use of acknowledging PDUs out of order to prevent needless retransmissions.

```
Time     Host A                           Host B

1.   <SEQ=100><ACK=200><Data>  --->

2.                                <--- <SEQ=201><ACK=100>

3.   <SEQ=101><ACK=200><Data> (delayed)

4.

5.   <SEQ=102><ACK=200><Data>  --->

6.                                <--- <SEQ=201><ACK=100><EA=102>

7.   <SEQ=103><ACK=200><Data>  --->
                                 ---> (delayed PDU 101 arrives)

8.                                <--- <SEQ=201><ACK=103>

9.   <SEQ=104><ACK=200><Data>  --->

10.                               <--- <SEQ=201><ACK=104>
```

A.4 Communication over long delay path

This is an example of a data transfer over a long delay path. In this example, Host A is permitted to have as many as five unacknowledged PDUs. The example shows that it is not necessary to wait for an acknowledgement in order to send additional data.

```
Time       Host A                     Host B

1.   <SEQ=100><ACK=200><Data> -1->
2.   <SEQ=101><ACK=200><Data> -2->
3.   <SEQ=102><ACK=200><Data> -3->
                               -1-> (received)
4.                        <-4- <SEQ=201><ACK=100>
5.   <SEQ=103><ACK=200><Data> -5->
                               -2-> (received)
6.                        <-6- <SEQ=201><ACK=101>
7.   <SEQ=104><ACK=200><Data> -7->
                               -3-> (received)
8.                        <-8- <SEQ=201><ACK=102>
               (received) <-4-
9.   <SEQ=105><ACK=200><Data> -9->
                               -5-> (received)
10.                      <-10- <SEQ=201><ACK=103>
               (received) <-6-
11.  <SEQ=106><ACK=200><Data> -11->
                               -7-> (received)
12.                      <-12- <SEQ=201><ACK=104>
               (received) <-8-
13.                            -9-> (received)
14.                      <-13- <SEQ=201><ACK=105>
               (received) <-10-
15.                            -11-> (received)
16.                      <-14- <SEQ=201><ACK=106>
               (received) <-12-
17.            (received) <-13-
18.            (received) <-14-
```

A.5 Communication over long delay path with lost PDUs

This is an example of communication over a long delay path with a lost PDU. It shows that by acknowledging PDUs out of sequence, only the lost PDU need be retransmitted.

```
    Time        Host A                    Host B

    1. <SEQ=100><ACK=200><Data>  -1->
    2. <SEQ=101><ACK=200><Data>  -2->
    3. <SEQ=102><ACK=200><Data>  -3->
                                 -1-> (received)
    4.                           <-4-  <ACK=100>
    5. <SEQ=103><ACK=200><Data> (PDU lost)
                                 -2-> (received)
    6.                           <-5- <SEQ=201><ACK=101>
    7. <SEQ=104><ACK=200><Data>  -6->
                                 -3-> (received)
    8.                           <-7- <SEQ=201><ACK=102>
                    (received) <-4-
    9. <SEQ=105><ACK=200><Data>  -8->
    10.
                    (received) <-5-
    11. <SEQ=106><ACK=200><Data> -10->
                                 -6-> (received)
    12.                          <-11- <SEQ=201><ACK=102><EA=104>
                    (received) <-7-
                                 -8-> (received)
    13.                          <-12- <SEQ=201><ACK=102><EA=104,105>
                                 -10-> (received)
    14.                          <-13- <SEQ=201><ACK=102><EA=104-106>
                    (received) <-11-
    15. <SEQ=103><ACK=200><Data> -14->
                    (received) <-12-
    16.                 (received) <-13-
                                 -14-> (received)
    17.                          <-15- <SEQ=201><ACK=106>
    18.
    19.             (received) <-15-
```

A.6 Detecting a half open connection on crash recovery

This is an example of a host detecting a half-open connection due to the crash and subsequent restart of the host. In this example, Host A crashes during a communication session, then recovers and tries to reopen the connection. During the reopen attempt, it discovers that a half-open connection still exists and it then resets the other side. Both sides were in the OPEN state prior to the crash.

```
    Host A                                Host B

    Time

    1.  OPEN                                OPEN
        (crash!)              <--- <SEQ=200><ACK=100><ACK>

    2.  CLOSED                               OPEN
        (recover)

    3.  SYN-SENT                             OPEN
            <SEQ=400><SYN> --->        (?)

    4.  SYN-SENT                             OPEN
         (!)                 <--- <SEQ=200><ACK=100><ACK>

    5.  SYN-SENT                             OPEN
            <SEQ=101><RST> --->        (abort)

    6.  SYN-SENT                             CLOSED

    7.  SYN-SENT <SEQ=400><SYN> --->
```

A.7 Detecting a half open connection from the active side

This is another example of detecting a half-open connection due to the crash and restart of a host involved in a connection. In this example, host A again crashes and restarts. Host B is still active and tries to send data to host A. Since host A has no knowledge of the connection, it rejects the data with an RST PDU, causing host B to reset the connection.

```
         Host A                          Host B

   Time

   1.  (crash!)                                     OPEN

   2.  CLOSED             <--- <SEQ=200><ACK=100><Data> OPEN

   3.  CLOSED  <SEQ=101><RST> --->                 (abort)

   4.  CLOSED                                      CLOSED
```

A.8 Dynamic window management

In this example, a card with just one global buffer for 5 segments is assumed. 2 applet loads happen just after each other, the first for a 5 segment applet. The current window is given after the "/" in the ACK. After reception of the first 5 segments, the card closes the window and keeps it closed until the application finishes processing and releases the buffer again. One segment loss is also included in the example sequence.

```
            Server                          Card

   Time  State                              State

   1.    LISTEN                             LISTEN

   2.    SYN-SENT    <SEQ=100><SYN> --->

   3.                          <--- <SEQ=200><ACK=100/5><SYN,ACK>
                                    (card announcing its window)
                                                SYN-RCVD

   4.    OPEN   <SEQ=201><ACK=200/9> --->           OPEN

   5.     <SEQ=101><ACK=200/9><Data> --->

   6.                          <--- <SEQ=201><ACK=101/4>
                                    (card reducing window)

   7.    <SEQ=102><ACK=200/9><Data> packet lost

   7.    <SEQ=103><ACK=200/9><Data> --->

   7.                          <--- <SEQ=201><ACK=101/4><EA=103>

   10.   <SEQ=104><ACK=200/9><Data> --->

   11.                         <--- <ACK=101/4><EA=103,104>

   12.   <SEQ=102><ACK=200/9><Data> --->
         (packet resent)

   13.                         <--- <SEQ=201><ACK=104/1>

   14.   <SEQ=105><ACK=200/9><Data> --->

   15.                         <--- <SEQ=201><ACK=105/0>
                                    (window now closed)

   16.    <SEQ=106><ACK=200/9><NUL> --->
         (server probes the card)

   17.                         <--- <SEQ=201><ACK=106/0>
                                    (window still closed)
```

```
18.                              <--- <SEQ=201><ACK=106/5>
                                      (processing finished,
                                       window opened again)

19.     <SEQ=107><ACK=200/9><Data> --->
        (data transfer resuming)

20.                              <--- <SEQ=201><ACK=107/4>
```

Annex B (informative):
CAT_TP-Upper layer interface definition

This annex specifies some minimum behaviour between CAT_TP and an application and lists exchanged parameters. The underneath technology or bearer is out of the scope of this annex.

This annex is language independent.

B.1 OPEN

ACTIVE OPEN

While performing an ACTIVE OPEN, an application shall give:

- The remote CAT_TP port to connect with.

may give:

- The maximum SDU size it will need in emission and in reception.

- Its own local CAT_TP port.

The CAT_TP layer shall respond:

- with a CAT_TP link identifier (if the CAT_TP link has been established successfully); or

- with an error (if the CAT_TP link has not been established successfully);

- with the minimum value between the maximum SDU size that the CAT_TP is able to send and the maximum SDU size that the remote CAT_TP entity is able to receive (this value corresponds to the max data size that the application can send later on);

- with the Maximum SDU size in reception (provide in the first SYN);

may respond:

- with the local port of the CAT_TP connection.

PASSIVE OPEN

While performing a PASSIVE OPEN, an application shall give:

- Its own local port. (may be a well-known one).

may give:

- The maximum SDU size it will be able to receive and to send.

- The authorized "client" CAT_TP port.

The CAT_TP layer shall respond:

- With a CAT_TP link identifier (if the CAT_TP instance has reached successfully the LISTEN state).

The CAT_TP layer shall:

- Signal the application once a connection was established by a remote entity.

- Give the maximum SDU size in emission and reception.

B.2   Send

Sends a SDU.

While performing a SEND, an application shall give:

- A SDU (equal or smaller than the maximum SDU size returned by ACTIVE OPEN).

- A CAT_TP link identifier on which the SDU has to be sent.

The CAT_TP layer shall respond:

- with request accepted; or

- with request error;

may later notify:

- that the SDU has been successfully received (PDUs have been successfully acknowledged) by the peer CAT_TP entity.

B.3   Receive

Receives data. A notification shall be performed by the CAT_TP layer indicating the size of the available data.

The application then takes the data by performing a RECEIVE.

While performing a RECEIVE, an application shall give:

- A buffer where the data has to be stored.

The CAT_TP layer shall respond:

- with the received data and its size; or

- with an error in case of receiving issue.

may respond:

- with the remaining available data size (if any).

B.4   CLOSE

Close a CAT_TP link.

While performing a CLOSE, an application shall give:

- The CAT_TP link identifier that has to be closed.

The CAT_TP layer shall respond:

- with OK if the CAT_TP layer has entered the CLOSE-WAIT state; or

- with an error in case of closing problem.

B.5 Status

Return the status of a CAT_TP link. This will send a NUL frame to check if the connection is established on both ends.

While performing a STATUS, an application shall give:

- The CAT_TP link identifier that has to be checked.

The CAT_TP layer shall respond:

- the status (ok or nok); or

- an error in case of problem when performing the STATUS.

Annex C (informative):
Change history

The table below indicates all changes that have been incorporated into the present document since it was placed under change control.

Change history								
Date	**Meeting**	**Plenary Doc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**Old**	**New**
2004	SCP-16	SCP-040084	1		Rel-6	Correction of 2 errors in flow chart.	6.0.0	6.1.0
2004	SCP-17	SCP-040238	2		Rel-6	Reintroduce ACK Sequence Numbers in annex A examples	6.1.0	6.2.0
2004	SCP-17	SCP-040238	3		Rel-6	Corrections to annex B	6.1.0	6.2.0
2004	SCP-17	SCP-040238	4		Rel-6	Corrections to Retransmission	6.1.0	6.2.0
2004	SCP-19	SCPt040285	5		Rel-6	Clarification for non-specific references.	6.2.0	6.3.0
2005	SCP-20	SCPt040481	6		Rel-6	Corrections regarding the term OSI	6.3.0	6.4.0
2005	SCP-23	SCP-050483	007		Rel-6	Clarification on SYN-ACK PDU reception checks	6.4.0	6.5.0
2006	SCP-25	SCP-060147	008		F	Correction of an error in a flow chart	6.5.0	6.6.0

List of figures

History

Document history		
V6.0.0	January 2004	Publication
V6.1.0	March 2004	Publication
V6.2.0	June 2004	Publication
V6.3.0	December 2004	Publication
V6.4.0	May 2005	Publication
V6.5.0	January 2006	Publication
V6.6.0	April 2006	Publication