ETSI TS 125 322 V3.4.0 (2000-09)

Technical Specification

Universal Mobile Telecommunications System (UMTS); RLC protocol specification (3GPP TS 25.322 version 3.4.0 Release 1999)

Reference

RTS/TSGR-0225322UR3

Keywords

UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

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://www.etsi.org/ipr).

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 the ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under www.etsi.org/key .

Contents

Foreword

This Technical Specification (TS) has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG 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 specifies the RLC protocol.

Release '99 features:

- Transparent mode.
- Unacknowledged mode.
- Acknowledged mode.

Features for future Releases:

- Hybrid ARQ.

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, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

[1] 3GPP TS 25.401: "UTRAN Overall Description".

[2] 3GPP TR 25.990: "Vocabulary for the UTRAN".

[3] 3GPP TS 25.301: "Radio Interface Protocol Architecture".

[4] 3GPP TS 25.302: "Services Provided by the Physical Layer".

[5] 3GPP TS 25.303: "Interlayer Procedures in Connected Mode".

[6] 3GPP TS 25.304: "UE Procedures in Idle Mode and Procedures for Cell Reselection in Connected Mode".

[7] 3GPP TS 25.321: "MAC Protocol Specification".

[8] 3GPP TS 25.331: "RRC Protocol Specification".

3 Abbreviations

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

ARQ Automatic Repeat Request
BCCH Broadcast Control Channel
BCH Broadcast Channel
C- Control-
CCCH Common Control Channel
CCH Control Channel
CCTrCH Coded Composite Transport Channel
CRC Cyclic Redundancy Check
DCCH Dedicated Control Channel
DCH Dedicated Channel
DL Downlink

DSCH	Downlink Shared Channel
DTCH	Dedicated Traffic Channel
FACH	Forward Link Access Channel
FDD	Frequency Division Duplex
L1	Layer 1 (physical layer)
L2	Layer 2 (data link layer)
L3	Layer 3 (network layer)
LSB	Least Significant Bit
MAC	Medium Access Control
MRW	Move Receive Window
MSB	Most Significant Bit
PCCH	Paging Control Channel
PCH	Paging Channel
PDU	Protocol Data Unit
PU	Payload Unit.
PHY	Physical layer
PhyCH	Physical Channels
RACH	Random Access Channel
RLC	Radio Link Control
RRC	Radio Resource Control
SAP	Service Access Point
SDU	Service Data Unit
SHCCH	Shared Channel Control Channel
TCH	Traffic Channel
TDD	Time Division Duplex
TFI	Transport Format Indicator
U-	User-
UE	User Equipment
UL	Uplink
UMTS	Universal Mobile Telecommunications System
UTRA	UMTS Terrestrial Radio Access
UTRAN	UMTS Terrestrial Radio Access Network

4 General

4.2 Overview on sublayer architecture

The model presented in this section is not for implementation purposes.

4.2.1 Model of RLC

Figure 4.1 gives an overview model of the RLC layer. The figure illustrates the different RLC peer entities. There is one transmitting and one receiving entity for the transparent mode service and the unacknowledged mode service and one combined transmitting and receiving entity for the acknowledged mode service. The dashed lines between the AM-Entities illustrate the possibility to send the RLC PDUs on separate logical channels, e.g. control PDUs on one and data PDUs on the other. More detailed descriptions of the different entities are given in subclauses 4.2.1.1, 4.2.1.2 and 4.2.1.3.

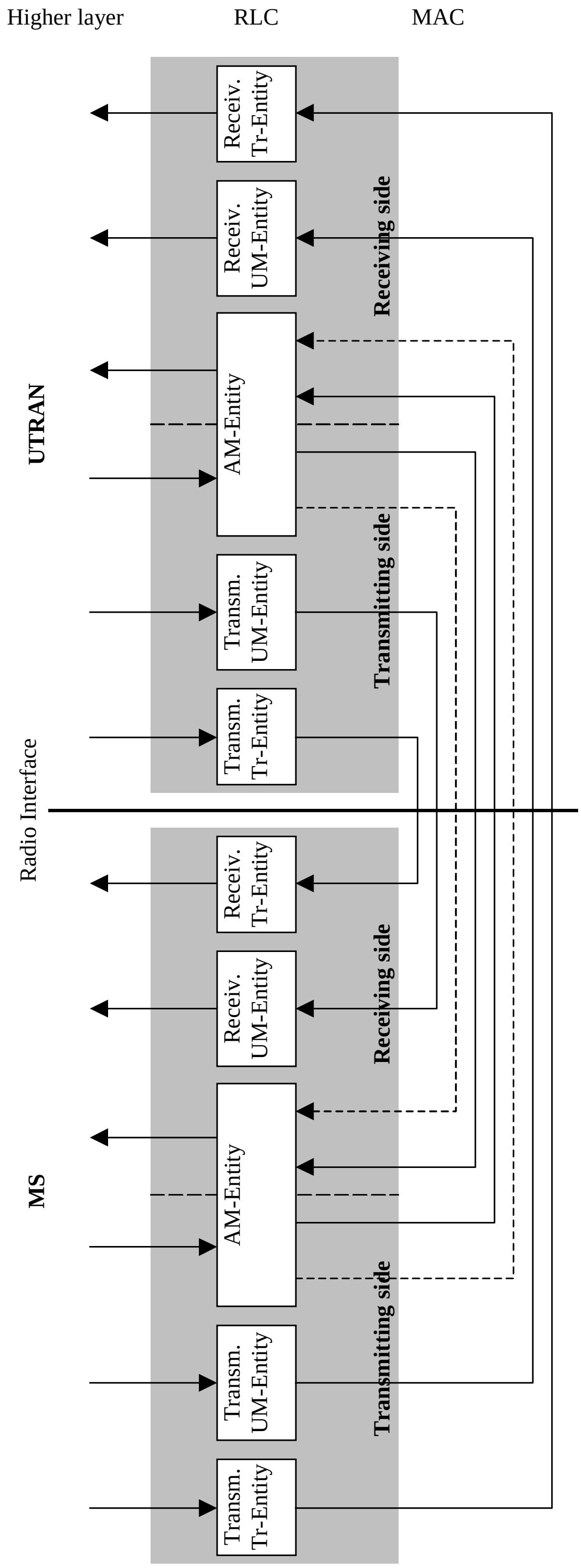


Figure 4.1: Overview model of RLC

4.2.1.1 Transparent mode entities

Figure 4.2 below shows the model of two transparent mode peer entities.

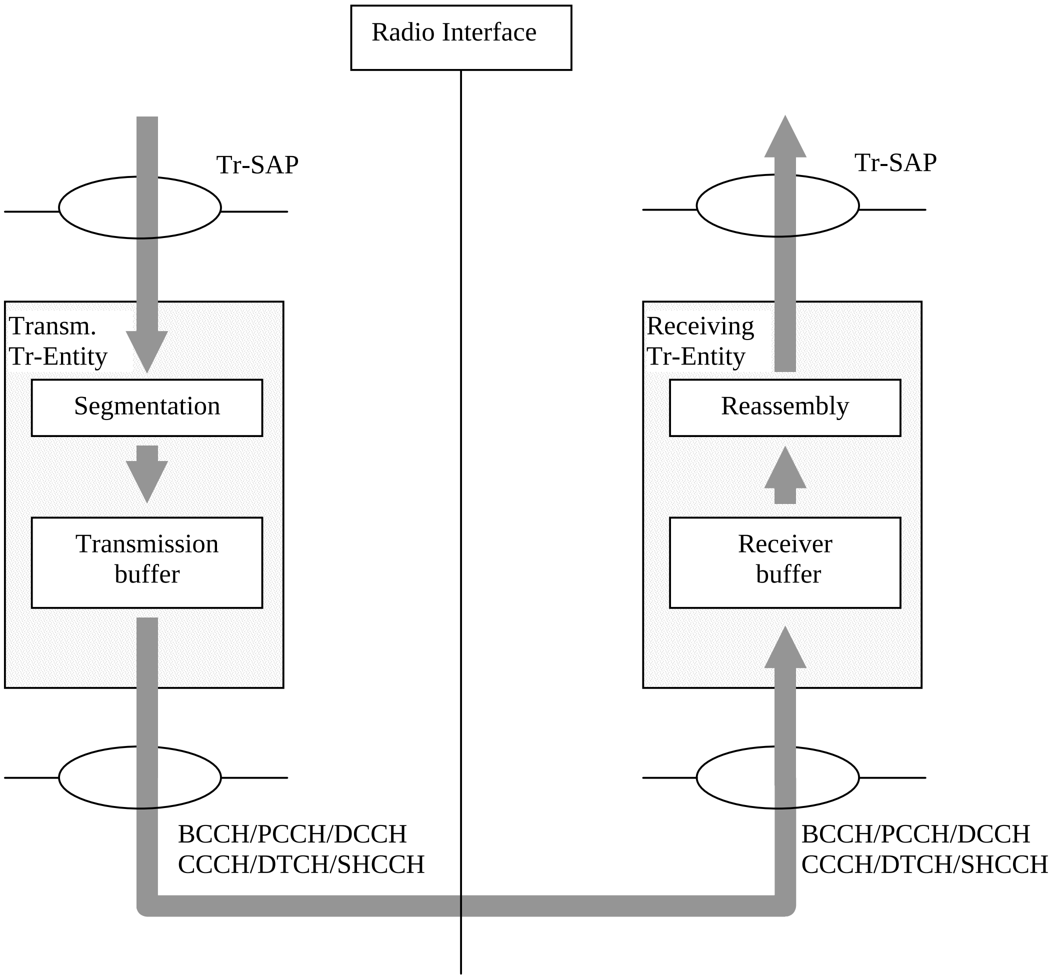


Figure 4.2: Model of two transparent mode peer entities

The transmitting Tr-entity receives SDUs from the higher layers through the Tr-SAP. RLC might segment the SDUs into appropriate RLC PDUs without adding any overhead. How to perform the segmentation is decided upon when the service is established. RLC delivers the RLC PDUs to MAC through either a BCCH, DCCH, PCCH, SHCCH or a DTCH. The CCCH and SHCCH also uses transparent mode, but only for the uplink. Which type of logical channel depends on if the higher layer is located in the control plane (BCCH, DCCH, PCCH, CCCH, SHCCH) or user plane (DTCH).

The Tr-entity receives PDUs through one of the logical channels from the MAC sublayer. RLC reassembles (if segmentation has been performed) the PDUs into RLC SDUs. How to perform the reassembling is decided upon when the service is established. RLC delivers the RLC SDUs to the higher layer through the Tr-SAP.

4.2.1.2 Unacknowledged mode entities

Figure 4.3 below shows the model of two unacknowledged mode peer entities.

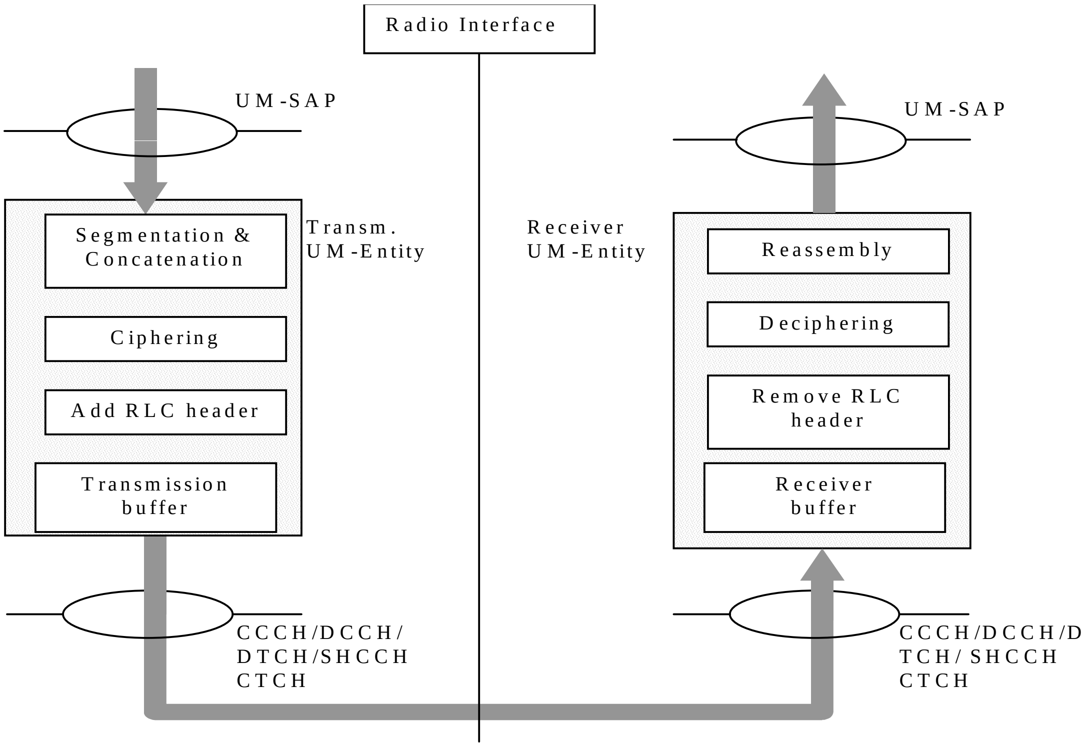


Figure 4.3: Model of two unacknowledged mode peer entities

The transmitting UM-entity receives SDUs from the higher layers. RLC might segment the SDUs into RLC PDUs of appropriate size. The SDU might also be concatenated with other SDUs. RLC adds a header and the PDU is placed in the transmission buffer. RLC delivers the RLC PDUs to MAC through either a DCCH, CTCH or a DTCH. The CCCH and SHCCH also uses unacknowledged mode, but only for the downlink. Which type of logical channel depends on if the higher layer is located in the control plane (CCCH, DCCH, SHCCH) or user plane (CTCH, DTCH).

The receiving UM-entity receives PDUs through one of the logical channels from the MAC sublayer. RLC removes header from the PDUs and reassembles the PDUs (if segmentation has been performed) into RLC SDUs. The RLC SDUs are delivered to the higher layer.

4.2.1.3 Acknowledged mode entity

Figure 4.4 below shows the model of an acknowledged mode entity, when one logical channel (shown as a solid line) and when two logical channels (shown as dashed lines) are used.

In case two logical channels are used in the uplink the UTRAN can indicate that the first logical channel shall be used for data PDUs and the second logical channel shall be used for control PDUs. If the indication is not given from the UTRAN, data and control PDUs can be sent on either of the two logical channels. The indication of the logical channel mapping is signalled by RRC.

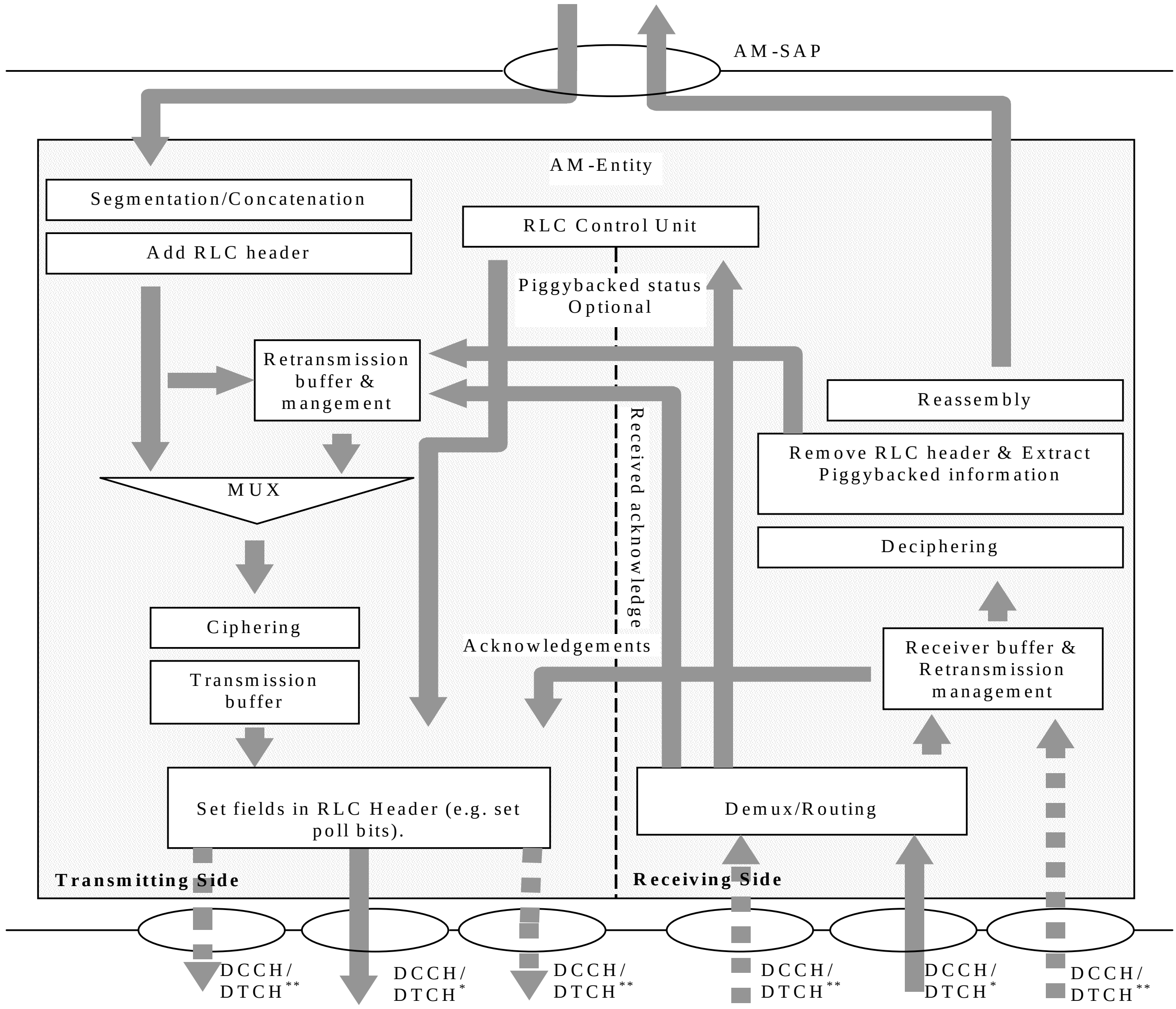


Figure 4.4: Model of a acknowledged mode entity

The transmitting side of the AM-entity receives SDUs from the higher layers. The SDUs are segmented and/or concatenated to PUs of fixed length. PU length is a semi-static value that is decided in bearer setup and can only be changed through bearer reconfiguration by RRC.

For purposes of RLC buffering and retransmission handling, the operation is the same as if there would be one PU per PDU. For concatenation or padding purposes, bits of information on the length and extension are inserted into the beginning of the last PU where data from an SDU is included. Padding can be replaced by piggybacked status information. This includes setting the poll bit.

If several SDUs fit into one PU, they are concatenated and the appropriate length indicators are inserted into the beginning of the PU. After that the PUs are placed in the retransmission buffer and the transmission buffer. One PU is included in one RLC PDU.

The MUX then decides which PDUs and when the PDUs are delivered to MAC. The PDUs are delivered via a function that completes the RLC-PDU header. The fixed 2 octet AMD PDU header is not ciphered.

When Piggybacking mechanism is applied the padding is replaced by control information, in order to increase the transmission efficiency and making possible a faster message exchange between the peer to peer RLC entities. The piggybacked control information is not saved in any retransmission buffer. The piggybacked control information is contained in the piggybacked STATUS PDU, which is in turn included into the AMD-PDU. The piggybacked STATUS PDUs will be of variable size in order to match with the amount of free space in the AMD PDU.

The retransmission buffer also receives acknowledgements from the receiving side, which are used to indicate retransmissions of PUs and when to delete a PU from the retransmission buffer.

The Receiving Side of the AM-entity receives PDUs through one of the logical channels from the MAC sublayer. The RLC-PDUs are expanded into separate PUs and potential piggybacked status information are extracted. The PUs are placed in the receiver buffer until a complete SDU has been received. The receiver buffer requests retransmissions of PUs by sending negative acknowledgements to the peer entity. After that the headers are removed from the PDUs and the PDUs are reassembled into a SDU. Finally the SDU is delivered to the higher layer. The receiving side also receives acknowledgements from the peer entity. The acknowledgements are passed to the retransmission buffer on the transmitting side.

5 Functions

The following functions are supported by RLC. For a detailed description of the following functions see [3]:

- Segmentation and reassembly.
- Concatenation.
- Padding.
- Transfer of user data.
- Error correction.
- In-sequence delivery of higher layer PDUs.
- Duplicate Detection.
- Flow control.
- Sequence number check (Unacknowledged data transfer mode).
- Protocol error detection and recovery.
- Ciphering.
- Suspend/resume function.

6 Services provided to upper layers

This clause describes the different services provided by RLC to higher layers. It also includes mapping of functions to different services. For a detailed description of the following functions see [3].

- **Transparent data transfer Service.**

  The following functions are needed to support transparent data transfer:

  - Segmentation and reassembly.
  - Transfer of user data.

- **Unacknowledged data transfer Service:**

  The following functions are needed to support unacknowledged data transfer:

  - Segmentation and reassembly.
  - Concatenation.
  - Padding.
  - Transfer of user data.

  - Ciphering.
  - Sequence number check.

- **Acknowledged data transfer Service:**

  The following functions are needed to support acknowledged data transfer:

  - Segmentation and reassembly.
  - Concatenation.
  - Padding.
  - Transfer of user data.
  - Error correction.
  - In-sequence delivery of higher layer PDUs.
  - Duplicate detection.
  - Flow Control.
  - Protocol error detection and recovery.
  - Ciphering.

- **QoS setting:**
- **Notification of unrecoverable errors.**

6.1 Mapping of services/functions onto logical channels

The following tables show the applicability of services and functions to the logical channels in UL/DL and UE/UTRAN. A '+' in a column denotes that the service/function is applicable for the logical channel in question whereas a '-' denotes that the service/function is not applicable.

Table 6.1: RLC modes and functions in UE uplink side

Service	Functions	CCCH	SHCCH	DCCH	DTCH
Transparent Service	Applicability	+	+	+	+
	Segmentation	-	-	+	+
	Transfer of user data	+	+	+	+
Unacknowledged Service	Applicability	-	-	+	+
	Segmentation	-	-	+	+
	Concatenation	-	-	+	+
	Padding	-	-	+	+
	Transfer of user data	-	-	+	+
	Ciphering	-	-	+	+
Acknowledged Service	Applicability	-	-	+	+
	Segmentation	-	-	+	+
	Concatenation	-	-	+	+
	Padding	-	-	+	+
	Transfer of user data	-	-	+	+
	Flow Control	-	-	+	+
	Error Correction	-	-	+	+
	Protocol error correction & recovery	-	-	+	+
	Ciphering	-	-	+	+

Table 6.2: RLC modes and functions in UE downlink side

Service	Functions	BCCH	PCCH	SHCCH	CCCH	DCCH	DTCH	CTCH
Transparent Service	Applicability	+	+	-	-	+	+	-
	Reassembly	-	+	-	-	+	+	-
Unacknowledged Service	Applicability	-	-	+	+	+	+	+
	Reassembly	-	-	+	+	+	+	+
	Deciphering	-	-	-	-	+	+	-
	Sequence number check	-	-	+	+	+	+	+
Acknowledged Service	Applicability	-	-	-	-	+	+	-
	Reassembly	-	-	-	-	+	+	-
	Error correction	-	-	-	-	+	+	-
	Flow Control	-	-	-	-	+	+	-
	In sequence delivery	-	-	-	-	+	+	-
	Duplicate detection	-	-	-	-	+	+	-
	Protocol error correction & recovery	-	-	-	-	+	+	-
	Deciphering	-	-	-	-	+	+	-

Table 6.3: RLC modes and functions in UTRAN downlink side

Service	Functions	BCCH	PCCH	CCCH	SHCCH	DCCH	DTCH	CTCH
Transparent Service	Applicability	+	+	-	-	+	+	-
	Segmentation	-	+	-	-	+	+	-
	Transfer of user data	+	+	-	-	+	+	-
Unacknowledged Service	Applicability	-	-	+	+	+	+	+
	Segmentation	-	-	+	+	+	+	+
	Concatenation	-	-	+	+	+	+	+
	Padding	-	-	+	+	+	+	+
	Ciphering	-	-	-	-	+	+	-
	Transfer of user data	-	-	+	+	+	+	+
Acknowledged Service	Applicability	-	-	-	-	+	+	-
	Segmentation	-	-	-	-	+	+	-
	Concatenation	-	-	-	-	+	+	-
	Padding	-	-	-	-	+	+	-
	Transfer of user data	-	-	-	-	+	+	-
	Flow Control	-	-	-	-	+	+	-
	Error Correction	-	-	-	-	+	+	-
	Protocol error correction & recovery	-	-	-	-	+	+	-
	Ciphering	-	-	-	-	+	+	-

Table 6.4: RLC modes and functions in UTRAN uplink side

Service	Functions	CCCH	SHCCH	DCCH	DTCH
Transparent Service	Applicability	+	+	+	+
	Reassembly	-	-	+	+
Unacknowledged Service	Applicability	-	-	+	+
	Reassembly	-	-	+	+
	Deciphering	-	-	+	+
	Sequence number check	-	-	+	+
Acknowledged Service	Applicability	-	-	+	+
	Reassembly	-	-	+	+
	Error correction	-	-	+	+
	Flow Control	-	-	+	+
	In sequence delivery	-	-	+	+
	Duplicate detection	-	-	+	+
	Protocol error correction & recovery	-	-	+	+
	Deciphering	-	-	+	+

7 Services expected from MAC

For a detailed description of the following functions see [3].

- Data transfer.

8 Elements for layer-to-layer communication

The interaction between the RLC layer and other layers are described in terms of primitives where the primitives represent the logical exchange of information and control between the RLC layer and other layers. The primitives shall not specify or constrain implementations.

8.1 Primitives between RLC and higher layers

The primitives between RLC and upper layers are shown in Table 8.1.

Table 8.1: Primitives between RLC and upper layers

Generic Name	Parameter			
	Req.	Ind.	Resp.	Conf.
RLC-AM-DATA	Data, CNF, MUI	Data, DiscardInfo	Not Defined	MUI
RLC-UM-DATA	Data, Use special LI	Data	Not Defined	Not Defined
RLC-TR-DATA	Data	Data	Not Defined	Not Defined
CRLC-CONFIG	E/R, Stop, Ciphering Elements (UM/AM only), AM_parameters (AM only)	Not Defined	Not Defined	Not Defined
CRLC-SUSPEND (UM/AM only)	N	Not Defined	Not Defined	VT(US) (UM only), VT(S) (AM only)
CRLC-RESUME (UM/AM only)	No Parameter	Not Defined	Not Defined	Not Defined
CRLC-STATUS	Not Defined	EVC	Not Defined	Not Defined

Each Primitive is defined as follows:

RLC-AM-DATA-Req/Ind/Conf

- RLC-AM-DATA-Req is used by higher layers to request transmission of a higher layer PDU in acknowledged mode.
- RLC-AM-DATA-Ind is used by RLC to deliver to higher layers RLC SDUs, that have been transmitted in acknowledged mode and to indicate higher layers of the discarded RLC SDU in the receiving RLC.
- RLC-AM-DATA-Conf is used by RLC to confirm to higher layers the transmission of a RLC SDU.

RLC-UM-DATA-Req/Ind

- RLC-UM-DATA-Req is used by higher layers to request transmission of a higher layer PDU in unacknowledged mode.
- RLC-UM-DATA-Ind is used by RLC to deliver to higher layers RLC SDUs, that have been transmitted in unacknowledged mode.

RLC-TR-DATA-Req/Ind

- RLC-TR-DATA-Req is used by higher layers to request transmission of a higher layer PDU in transparent mode.
- RLC-TR-DATA-Ind is used by RLC to deliver to higher layers RLC SDUs, that have been transmitted in transparent mode.

CRLC-CONFIG-Req

This primitive is used by RRC to establish, re-establish, release or reconfigure the RLC. Ciphering elements are included for UM and AM operation.

CRLC-SUSPEND-Req/Conf

This primitive is used by RRC to suspend the RLC. The N parameter indicates that RLC shall not send a PDU with SN>=VT(S)+N, where N is an integer. RLC informs RRC of the VT(S) value in the confirm primitive.

CRLC-RESUME-Req

This primitive is used by RRC to resume RLC when RLC has been suspended.

CRLC-STATUS-Ind

It is used by the RLC to send status information to RRC.

8.2 Primitive parameters

Following parameters are used in the primitives:

1) The parameter Data is the RLC SDU that is mapped onto the Data field in RLC PDUs. The Data parameter may be divided over several RLC PDUs. In case of a RLC-AM-DATA or a RLC-UM-DATA primitive the length of the Data parameter shall be octet-aligned.

2) The parameter Confirmation request (CNF) indicates whether the RLC needs to confirm the correct transmission of the RLC SDU.

3) The parameter Message Unit Identifier (MUI) is an identity of the RLC SDU, which is used to indicate which RLC SDU that is confirmed with the RLC-AM-DATA conf. primitive.

4) The parameter E/R indicates (re)establishment, release or modification of RLC If it indicates establishment, the state variables in 9.4 shall be set to their initial value, the configurable parameters shall be set to their configured value and RLC shall enter the data transfer ready state. If it indicates re-establishment, the state variables in 9.4 shall be set to their initial value, the configurable parameters shall be set to their configured value the RLC buffers shall be emptied and RLC shall enter the data transfer ready state. If it indicates release, all protocol parameters, variables and timers shall be released and RLC shall exit the data transfer ready state. If it indicates modification, the protocol parameters indicated by RRC (e.g. ciphering parameters) shall only be modified with keeping the other protocol parameters, the protocol variables, the protocol timers and the protocol state. RLC shall always be re-established if the PU size is changed.

5) The parameter Event Code (EVC) indicates the reason for the CRLC-STATUS-ind (i.e., unrecoverable errors such as data link layer loss or recoverable status events such as reset, etc.).

6) The parameter ciphering elements are only applicable for UM and AM operation. These parameters are Ciphering Mode, Ciphering Key, Activation Time (SN to activate a new ciphering configuration) and HFN (Hyper Frame Number).

7) The AM_parameters are only applicable for AM operation. It contains PU size, Timer values (see subclause 9.5), Protocol parameter values (see subclause 9.6), Polling triggers (see subclause 9.7.1), Status triggers (see subclause 9.7.2), SDU discard mode (see subclause 9.7.3) and Minimum WSN (see subclause 9.2.2.11.3). The Minimum WSN shall always be greater than or equal to the number of transport blocks in the smallest transport block set.

8) The parameter DiscardInfo indicates the upper layer of each of the discarded RLC SDU. It is applicable only when in-sequence delivery is active and it is purposed to be used when the upper layer requires the reliable data transfer and especially the information of the discarded RLC SDU.

9) The Stop parameter indicates that the RLC entity shall not transmit or receive RLC PDUs.

10) The parameter Use special LI indicates that the LI indicating that a RLC SDU begins in the beginning of a RLC PDU (the first data octet of the PDU is the first octet of an SDU) shall be used. If the RLC SDU does not begin in the beginning of the RLC PDU, or if the LI indicating that a SDU ended exactly in the end of the previous RLC PDU is present, the LI shall not be used.

9 Elements for peer-to-peer communication

9.1 Protocol data units

9.1.1 Data PDUs

a) TrD PDU (Transparent Mode Data PDU).

The TrD PDU is used to convey RLC SDU data without adding any RLC overhead. The TrD PDU is used by RLC when it is in transparent mode.

b) UMD PDU (Unacknowledged Mode Data PDU).

The UMD PDU is used to convey sequentially numbered PDUs containing RLC SDU data. It is used by RLC when using unacknowledged data transfer.

c) AMD PDU (Acknowledged Mode Data PDU).

The AMD PDU is used to convey sequentially numbered PUs containing RLC SDU data. The AMD PDU is used by RLC when it is in acknowledged mode.

9.1.2 Control PDUs

a) STATUS PDU and Piggybacked STATUS PDU

The STATUS PDU and the Piggybacked STATUS PDU are used:

- by the receiving entity to inform the transmitting entity about missing PUs at the receiving entity;
- by the receiving entity to inform the transmitting entity about the size of the allowed transmission window;
- and by the transmitting entity to request the receiving entity to move the receiving window.

b) RESET PDU

The RESET PDU is used in acknowledged mode to reset all protocol states, protocol variables and protocol timers of the peer RLC entity in order to synchronise the two peer entities.

c) RESET ACK PDU

The RESET ACK PDU is an acknowledgement to the RESET PDU.

Table 9.1: RLC PDU names and descriptions

Data Transfer Mode	PDU name	Description
Transparent	TrD	Transparent mode data
Unacknowledged	UMD	Sequenced unacknowledged mode data
Acknowledged	AMD	Sequenced acknowledged mode data
	STATUS	Solicited or Unsolicited Status Report
	Piggybacked STATUS	Piggybacked Solicited or Unsolicited Status Report
	RESET	Reset Command
	RESET ACK	Reset Acknowledgement

9.2 Formats and parameters

9.2.1 Formats

This subclause specifies the format of the RLC PDUs. The parameters of each PDU are explained in subclause 9.2.2.

9.2.1.1 General

An RLC PDU is a bit string, with a length not necessarily a multiple of 8 bits. In the drawings in clause 9.2, bit strings are represented by tables in which the first bit is the leftmost one on the first line of the table, the last bit is the rightmost on the last line of the table, and more generally the bit string is to be read from left to right and then in the reading order of the lines.

Depending on the provided service, RLC SDUs are bit strings, with any nonnull length, or bit strings with an integer number of octets in length. An SDU is included into an RLC PDU from first bit onward.

9.2.1.2 TrD PDU

The TrD PDU transfers user data when RLC is operating in transparent mode. No overhead is added to the SDU by RLC. The data length is not constrained to be an integer number of octets.

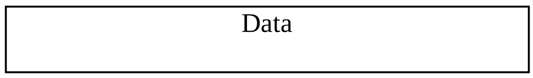


Figure 9.1: TrD PDU

9.2.1.3 UMD PDU

The UMD PDU transfers user data when RLC is operating in unacknowledged mode. The length of the data part shall be an integer number of octets.

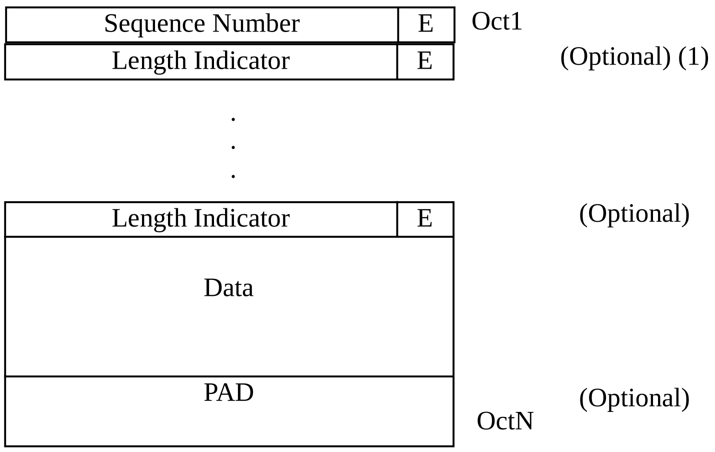


Figure 9.2: UMD PDU

NOTE (1): The Length Indicator may be 15 bits.

9.2.1.4 AMD PDU

The AMD PDU transfers user data and piggybacked status information and requests status report by setting Poll bit when RLC is operating in acknowledged mode. The length of the data part shall be an integer number of octets.

D/C	Sequence Number		Oct1
Sequence Number	P	HE	Oct2
Length Indicator		E	Oct3 (Optional) (1)

.
.
.

Length Indicator	E	
Data		
PAD or a piggybacked STATUS PDU		OctN

NOTE (1): The Length Indicator may be 15 bits.

Figure 9.3: AMD PDU

9.2.1.5 STATUS PDU

The STATUS PDU is used to report the status between two RLC AM entities. Both receiver and transmitter status information may be included in the same STATUS PDU.

The format of the STATUS PDU is given in Figure 9.4 below.

D/C	PDU type	SUFI 1	Oct 1
$SUFI_1$			Oct2
…			
$SUFI_K$			
PAD			OctN

Figure 9.4: Status Information Control PDU (STATUS PDU)

Up to K different super-fields ($SUFI_1$-$SUFI_K$) can be included into one STATUS PDU. The size of a STATUS PDU is variable and upper bounded by the maximum RLC PDU size used by an RLC entity. Padding shall be included to exactly fit one of the PDU sizes used by the entity. The length of the STATUS PDU shall be an integer number of octets.

9.2.1.6 Piggybacked STATUS PDU

The format of the piggybacked STATUS PDU is the same as the ordinary Control PDU except that the D/C field is replaced by a reserved bit (R). This PDU can be used to piggyback STATUS PDU in an AMD PDU if the data does not fill the complete AMD PDU. The PDU Type field is set to zero and all other values are invalid for this version of the protocol and the PDU is discarded.

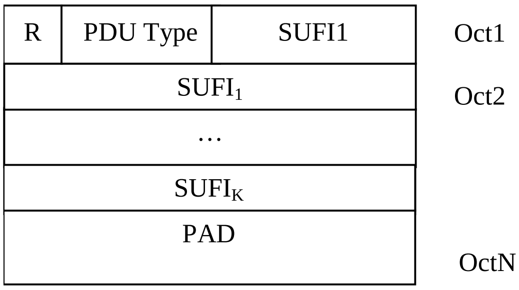


Figure 9.5: Piggybacked STATUS PDU

9.2.1.7 RESET, RESET ACK PDU

The RESET PDU has a one-bit sequence number field (RSN). With the aid of this field the Receiver can define whether the received RESET PDU is transmitted by the Sender for the first time or whether it is a retransmission of a previous RESET PDU.

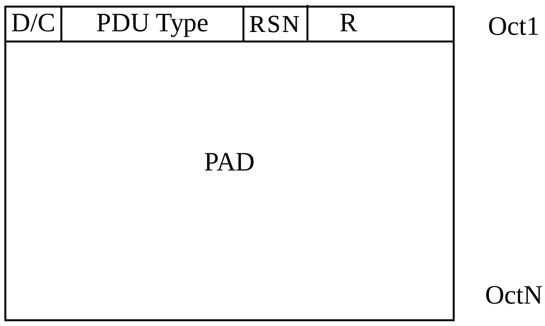


Figure 9.6: RESET, RESET ACK PDU

9.2.2 Parameters

If not otherwise mentioned in the definition of each field then the bits in the parameters shall be interpreted as follows: the left-most bit string is the first and most significant and the right most bit is the last and least significant bit.

Unless otherwise mentioned, integers are encoded in standard binary encoding for unsigned integers. In all cases, including when a value extends over more than one octet as shown in the tables, the bits appear ordered from MSB to LSB when read in the PDU.

9.2.2.1 D/C field

Length: 1bit.

The D/C field indicates the type of an acknowledged mode PDU. It can be either data or control PDU.

Bit	Description
0	Control PDU
1	Acknowledged mode data PDU

9.2.2.2 PDU Type

Length: 3 bit.

The PDU type field indicates the Control PDU type.

Bit	PDU Type
000	STATUS
001	RESET
010	RESET ACK
011-111	Reserved (PDUs with this coding will be discarded by this version of the protocol).

9.2.2.3 Sequence Number (SN)

This field indicates the sequence number of the payload unit, encoded in binary.

PDU type	Length	Notes
AMD PDU	12 bits	Used for retransmission and reassembly
UMD PDU	7 bits	Used for reassembly

9.2.2.4 Polling bit (P)

Length: 1bit.

This field is used to request a status report (one or several STATUS PDUs) from the receiver RLC.

Bit	Description
0	Status report not requested
1	Request a status report

9.2.2.5 Extension bit (E)

Length: 1bit.

This bit indicates if the next octet will be a length indicator and E bit.

Bit	Description
0	The next field is data
1	The next field is Length Indicator and E bit

9.2.2.6 Reserved (R)

Length: 3 bits.

This field in the RESET PDU and RESET ACK PDU is used to achieve octet alignment and for this purpose it is coded as 000. Other functions of it are left for future releases.

9.2.2.7 Header Extension Type (HE)

Length: 2 bits.

This two-bit field indicates if the next octet will be data or a length indicator and E bit.

Value	Description
00	The succeeding octet contains data
01	The succeeding octet contains a length indicator and E bit
10-11	Reserved (PDUs with this coding will be discarded by this version of the protocol).

9.2.2.8 Length Indicator (LI)

The Length Indicator is used to indicate, each time, the end of an SDU occurs in the PU. The Length Indicator points out the number of octets between the end of the last Length Indicator field and up to and including the octet at the end of an SDU segment. Length Indicators are included in the PUs that they refer to. The size of the Length Indicator may be either 7bits or 15bits. The maximum value of a Length Indicator in AM will be no greater than the RLC PDU size – AMD PDU Header – PADDING. The maximum value of a Length Indicator in UM will be no greater than the RLC PDU size – UMD PDU Header – PADDING.

A Length Indicator group is a set of Length Indicators that refer to a PU. Length Indicators that are part of a Length Indicator group must never be reordered within the Length Indicator group or removed from the Length Indicator group.

If there can be more than one Length Indicator, each specifying the end of an SDU in a PU, the order of these Length Indicators must be in the same order as the SDUs that they refer to.

In the case where the end of last segment of an SDU exactly ends at the end of a PDU and there is no LI that indicates the end of the SDU, the next Length Indicator, shall be placed as the first Length Indicator in the following PU and have value LI=0.

In the case where the last segment of an RLC SDU is one octet short of exactly filling the previous RLC PU, and 15-bit Length Indicators are used, the next Length Indicator shall be placed as the first Length Indicator in the following PU and have value LI=111 1111 1111 1011. The remaining one octet in the previous RLC PU shall be ignored.

A PU that has unused space, to be referred to as padding, must use a Length Indicator to indicate that this space is used as padding. A padding Length Indicator must be placed after any Length Indicators for a PU.

All unused space in a PU must be located at the end of the PDU, be a homogeneous space and is referred to as padding. Predefined values of the Length Indicator are used to indicate this. The values that are reserved for special purposes are listed in the tables below depending on the size of the Length Indicator. Only predefined Length Indicator values can refer to the padding space.

STATUS PDUs can be piggybacked on the AMD PDU by using part or all of the padding space. A Length Indicator must be used to indicate the piggybacked STATUS PDU. This Length Indicator takes space from the padding space or piggybacked STATUS PDU and not the PDU data and will always be the last Length Indicator. Where only part of the padding space is used by a piggybacked STATUS PDU then the end of the piggybacked STATUS PDU is determined by one of the SUFI fields NO_MORE or ACK, thus no additional Length Indicator is required to show that there is still padding in the PDU. The padding/piggybacked STATUS PDU predefined Length Indicators shall be added after the very last (i.e. there could be more than one SDU that end within a PDU) Length Indicator that indicates the end of the last SDU segment in the PU.

If SDU discard with explicit signalling is used an AMD PDU can contain a maximum number of 15 LIs indicating the end of an SDU and the rest of the AMD PDU space shall be used as padding/piggybacked STATUS PDU.

For AM, 7bit indicators shall be used if the AMD PDU size is ≤ l26 octets. Otherwise 15bit indicators shall be used. For UM, 7bit indicators shall be used if the UMD PDU size is ≤ l25 octets. Otherwise 15bit indicators shall be used.

The length of the Length Indicator only depends on the size of the largest RLC PDU. The length of the Length Indicator is always the same for all PUs, for one RLC entity.

For Release 99, there is one PU in an AMD PDU.

Length: 7bit

Bit	Description
0000000	The previous RLC PDU was exactly filled with the last segment of a RLC SDU and there is no LI that indicates the end of the SDU in the previous RLC PDU.
1111100	UMD PDU: The first data octet in this RLC PDU is the first octet of a RLC SDU. AMD PDU: Reserved (PDUs with this coding will be discarded by this version of the protocol).
1111101	Reserved (PDUs with this coding will be discarded by this version of the protocol).
1111110	AMD PDU: The rest of the RLC PDU includes a piggybacked STATUS PDU. UMD PDU: Reserved (PDUs with this coding will be discarded by this version of the protocol).
1111111	The rest of the RLC PDU is padding. The padding length can be zero.

Length: 15bit

Bit	Description
000000000000000	The previous RLC PDU was exactly filled with the last segment of an RLC SDU and there is no LI that indicates the end of the SDU in the previous RLC PDU.
111111111111011	The last segment of an RLC SDU was one octet short of exactly filling the previous RLC PDU. The remaining one octet in the previous RLC PDU is ignored.
111111111111100	UMD PDU: The first data octet in this RLC PDU is the first octet of a RLC SDU. AMD PDU: Reserved (PDUs with this coding will be discarded by this version of the protocol).
111111111111101	Reserved (PDUs with this coding will be discarded by this version of the protocol).
111111111111110	AMD PDU: The rest of the RLC PDU includes a piggybacked STATUS PDU. UMD PDU: Reserved (PDUs with this coding will be discarded by this version of the protocol).
111111111111111	The rest of the RLC PDU is padding. The padding length can be zero.

9.2.2.9 Data

RLC SDUs or segments of RLC SDUs are mapped to this field in transparent, unacknowledged and acknowledged mode.

Transparent mode data:

The length of RLC SDUs is not constrained to a multiple of 8 bits.

The RLC SDUs might be segmented. The allowed size for the segments shall be determined from the transport formats of the transport channel [4, 8]. All the RLC PDUs carrying one RLC SDU shall be sent in one transmission time interval. Only segments from one RLC SDU shall be sent in one transmission time interval.

NOTE: If segmentation is not used for the transparent mode RLC entity then more than one RLC SDU can be sent in one transmission time interval using one RLC PDU per RLC SDU. The RLC PDUs need, however, to be of the same size due to L1 limitations.

Unacknowledged mode data and Acknowledged mode data:

The length of RLC SDUs is constrained to a multiple of 8 bits.

RLC SDUs might be segmented. If possible, the last segment of a SDU shall be concatenated with the first segment of the next SDU in order to fill the data field completely and avoid unnecessary padding. The length indicator field is used to point the borders between SDUs.

9.2.2.10 Padding (PAD)

Padding has a length such that the PDU has the required predefined total length.

Padding may have any value and the receiving entity shall disregard it.

9.2.2.11 SUFI

Which SUFI fields to use is implementation dependent, but when a STATUS PDU includes information about which PUs have been received and which are detected as missing, information shall not be included about PUs with SN≥VR(H) i.e. PUs that have not yet reached the receiver. Information about PUs with SN<VR(R) shall not be given except when this is necessary in order to use the BITMAP SUFI, see 9.2.2.11.5.

Length: variable number of bits.

The SUFI (Super-Field) includes three sub-fields: type information (type of super-field, e.g. list, bitmap, acknowledgement, etc), length information (providing the length of a variable length field within the following value field) and a value.

Figure 9.7 shows the structure of the super-field. The size of the type sub-field is non-zero but the size of the other sub-fields may be zero.

Type
Length
Value

Figure 9.7: The Structure of a Super-Field

The length of the type field is 4 bits and it may have any of following values.

Bit	Description
0000	No More Data (**NO_MORE**)
0001	Window Size (**WINDOW**)
0010	Acknowledgement (**ACK**)
0011	List (**LIST**)
0100	Bitmap (**BITMAP**)
0101	Relative list (**Rlist)**
0110	Move Receiving Window **(MRW)**
0111	Move Receiving Window Acknowledgement **(MRW_ACK)**
1000-1111	Reserved (PDUs with this encoding are invalid for this version of the protocol)

The length sub-field gives the length of the variable size part of the following value sub-field and the length of it depends on the super-field type. The value sub-field includes the value of the super-field, e.g. the bitmap in case of a BITMAP super-field, and the length is given by the length of the type sub-field.

9.2.2.11.1 The No More Data super-field

The 'No More Data' super-field indicates the end of the data part of a STATUS PDU and is shown in Figure 9.8 below. It shall always be placed as the last SUFI if it is included in a STATUS PDU. All data after this SUFI shall be regarded as padding and shall be neglected.

Type=**NO_MORE**

Figure 9.8: NO_MORE field in a STATUS PDU

9.2.2.11.2 The Acknowledgement super-field

The 'Acknowledgement' super-field consists of a type identifier field (ACK) and a sequence number (LSN) as shown in Figure 9.9 below. The acknowledgement super-field is also indicating the end of the data part of a STATUS PDU. Thus, no 'NO_MORE' super-field is needed in the STATUS PDU when the 'ACK' super-field is present. The ACK SUFI shall always be placed as the last SUFI if it is included in a STATUS PDU. All data after this SUFI shall be regarded as padding and shall be neglected.

Type = **ACK**
LSN

Figure 9.9: The ACK fields in a STATUS PDU

LSN

Length: 12 bits

Acknowledges the reception of all PUs with sequence numbers < LSN (Last Sequence Number) that are *not* indicated to be erroneous in earlier parts of the STATUS PDU. This means that if the LSN is set to a different value than VR(R) all erroneous PUs must be included in the same STATUS PDU and if the LSN is set to VR(R) the erroneous PUs can be split into several STATUS PDUs. At the transmitter, if the value of the LSN =< the value of the first error indicated in the STATUS PDU VT(A) will be updated according to the LSN, otherwise VT(A) will be updated according to the first error indicated in the STATUS PDU. The LSN should not be set to a value > VR(H).

9.2.2.11.3 The Window Size super-field

The 'Window Size' super-field consists of a type identifier (WINDOW) and a window size number (WSN) as shown in Figure 9.10 below. The receiver is always allowed to change the Tx window size of the peer entity during a connection, but the minimum and the maximum allowed value is given by RRC configuration. The Rx window of the receiver is not changed.

Type = **WINDOW**
WSN

Figure 9.10: The WINDOW fields in a STATUS PDU

WSN

Length: 12 bits

The allowed Tx window size to be used by the transmitter. The range of the WSN is $[0, 2^{12}-1]$. The minimum value of the window size is 1, if WSN is zero the SUFI shall be discarded by this version of the protocol. The Tx_Window_Size parameter is set equal to WSN upon reception of this SUFI.

9.2.2.11.4 The List super-field

The List Super-Field consists of a type identifier field (LIST), a list length field (LENGTH) and a list of LENGTH number of pairs as shown in Figure 9.11 below:

Type = **LIST**
LENGTH
SN_1
L_1
SN_2
L_2
…
SN_{LENGTH}
L_{LENGTH}

Figure 9.11: The List fields in a STATUS PDU for a list

LENGTH

Length: 4 bits

The number of (SN_i , L_i)-pairs in the super-field of type LIST. The value "0000" is invalid and the list is discarded.

SN_i

Length: 12 bits

Sequence number of PU, which was not correctly received.

L_i

Length: 4 bits

Number of consecutive PUs not correctly received following PU with sequence number SN_i.

9.2.2.11.5 The Bitmap super-field

The Bitmap Super-Field consists of a type identifier field (BITMAP), a bitmap length field (LENGTH), a first sequence number (FSN) and a bitmap as shown in Figure 9.12 below:

Type = **BITMAP**
LENGTH
FSN
Bitmap

Figure 9.12: The Bitmap fields in a STATUS PDU

LENGTH

Length: 4 bits

The size of the bitmap in octets equals LENGTH+1, i.e. LENGTH="0000" means that the size of the bitmap is one octet and LENGTH="1111" gives the maximum bitmap size of 16 octets.

FSN

Length: 12 bits

The sequence number for the first bit in the bitmap. FSN shall not be set to a value lower than VR(R)-7 when the Rx window size is less then half the maximum RLC AM sequence number. If the Rx window size is larger, FSN shall not be set to a value lower than VR(R).

Bitmap

Length: Variable number of octets given by the LENGTH field.

Status of the SNs in the interval [FSN, FSN + (LENGTH+1)*8 - 1] indicated in the bitmap where each position (from left to right) can have two different values (0 and 1) with the following meaning (bit_position∈[0,(LENGTH+1)*8 - 1]):

1: SN = (FSN + bit_position) has been correctly received.

0: SN = (FSN + bit_position) has not been correctly received.

9.2.2.11.6 The Relative List super-field

The Relative List super-field consists of a type identifier field (RLIST), a list length field (LENGTH), the first sequence number (FSN) and a list of LENGTH number of codewords (CW) as shown in Figure 9.134 below.

Type = **RLIST**
LENGTH
FSN
CW_1
CW_2
…
CW_{LENGTH}

Figure 9.13: The RList fields in a STATUS PDU

LENGTH

Length: 4 bits

The number of codewords (CW) in the super-field of type RLIST.

FSN

Length: 12 bits

The sequence number for the first erroneous PU in the RLIST, i.e. LENGTH="0000" means that only FSN is present in the SUFI.

CW

Length: 4 bits

The CW consists of 4 bits where the three first bits are part of a number and the last bit is a status indicator and it shall be interpreted as follows:

Code Word	Description
$X_1X_2X_3$ 0	Next 3 bits of the number are $X_1X_2X_3$ and the number continues in the next CW. The most significant bit within this CW is X_1.
$X_1X_2X_3$ 1	Next 3 bits of the number are $X_1X_2X_3$ and the number is terminated. The most significant bit within this CW is X_1. This is the most significant CW within the number.

By default, the number given by the CWs represents a distance between the previous indicated erroneous PU up to and including the next erroneous PU.

One special value of CW is defined:

000 1 'Error burst indicator'.

The error burst indicator means that the next CWs will represent the number of subsequent erroneous PUs (not counting the already indicated error position). After the number of errors in a burst is terminated with XXX 1, the next codeword will again by default be the least significant bits (LSB) of the distance to the next error.

9.2.2.11.7 The Move Receiving Window Acknowledgement super-field

The 'Move Receiving Window Acknowledgement' super-field acknowledges the reception of a MRW SUFI. The format is given in the figure below.

Type = **MRW_ACK**
N
SN_ACK

Figure 9.14: The MRW-ACK fields in a STATUS PDU

N

Length: 4 bits

The N field shall be set equal to the N_{LENGTH} field in the received MRW SUFI if the SN_ACK field is equal to the SN_MRW_{LENGTH} field. Otherwise N shall be set to 0.

With the aid of this field in combination with the SN_ACK field, it can be determined if the MRW_ACK corresponds to a previously transmitted MRW SUFI.

SN_ACK

Length: 12 bits

The SN_ACK field indicates the updated value of VR(R) after the reception of the MRW SUFI. With the aid of this field in combination with the N field, it can be determined if the MRW_ACK corresponds to a previously transmitted MRW SUFI.

9.2.2.11.8 The Move Receiving Window (MRW) super-field

The 'Move Receiving Window' super-field is used to request the RLC receiver to move its receiving window and to indicate the amount of discarded SDUs, as a result of a SDU discard in the RLC transmitter. The format is given in the figure below.

Type = **MRW**
LENGTH
SN_MRW_1
...
SN_MRW_{LENGTH}
N_{LENGTH}

Figure 9.15: The MRW fields in a STATUS PDU

LENGTH

Length: 4 bits

The number of SN_MRW_i fields in the super-field of type MRW**.** The values "0001" through "1111" indicate 1 through 15 SN_MRW_i respectively. The value "0000" indicates that one SN_MRW_i field is present and that the discarded SDU extends above the Tx window in the transmitter.

SN_MRW_i

Length: 12 bits

SN_MRW_i is used to indicate the end of each discarded SDU. SN_MRW_i is the sequence number of the PU that contains the LI of the i:th discarded SDU (except when $N_{LENGTH} = 0$, see definition of N_{LENGTH}).

Additionally SN_MRW_{LENGTH} requests the RLC receiver to discard all PUs with sequence number < SN_MRW_{LENGTH}, and to move the receiving window accordingly. In addition, the receiver has to discard the first N_{LENGTH} LIs and the corresponding data bytes in the PU with sequence number SN_MRW_{LENGTH}.

N_{LENGTH}

Length: 4 bits

N_{LENGTH} is used together with SN_MRW_{LENGTH} to indicate the end of the last discarded SDU.

N_{LENGTH} indicates which LI in the PU with sequence number SN_MRW_{LENGTH} corresponds to the last discarded SDU. $N_{LENGTH} = 0$ indicates that the last SDU ended in the PU with sequence number $SN\_MRW_{LENGTH} -1$ and that the first data byte in the PU with sequence number SN_MRW_{LENGTH} is the first data byte to be reassembled next.

9.2.2.12 Reserved (R)

Length: 1 bit

This bit in the Piggybacked STATUS PDU is used to achieve octet alignment and for this purpose it is coded as 0. Otherwise the PDU is treated as invalid and hence shall be discarded by this version of the protocol.

9.2.2.13 Reset Sequence Number (RSN)

Length: 1 bit

This field is used to indicate the sequence number of the transmitted RESET PDU. If this RESET PDU is a retransmission of the original RESET PDU then the retransmitted RESET PDU would have the same sequence number value as the original RESET PDU. Otherwise it will have the next reset sequence number. The initial value of this field is zero. The value of this field shall be reinitialized when the RLC is re-established. It shall not be reinitialized when the RLC is reset.

9.3 Protocol states

9.3.1 State model for transparent mode entities

Figure 9.16 illustrates the state model for transparent mode RLC entities (both transmitting and receiving). A transparent mode entity can be in one of following states.

9.3.1.1 Null State

In the null state the RLC entity does not exist and therefore it is not possible to transfer any data through it.

Upon reception of an CRLC-CONFIG-Req from higher layer the RLC entity is created and transparent data transfer ready state is entered.

9.3.1.2 Transparent Data Transfer Ready State

In the transparent data transfer ready, transparent mode data can be exchanged between the entities. Upon reception of an CRLC-CONFIG-Req from higher layer the RLC entity is terminated and the null state is entered.

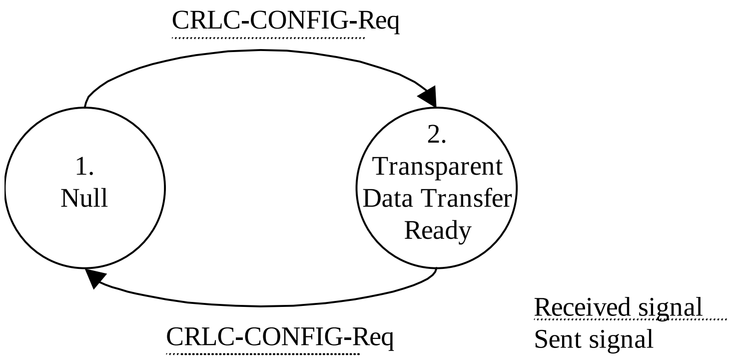


Figure 9.16: The state model for transparent mode entities

9.3.2 State model for unacknowledged mode entities

Figure 9.17 illustrates the state model for unacknowledged mode RLC entities (both transmitting and receiving). An unacknowledged mode entity can be in one of following states.

9.3.2.1 Null State

In the null state the RLC entity does not exist and therefore it is not possible to transfer any data through it.

Upon reception of an CRLC-CONFIG-Req from higher layer the RLC entity is created and unacknowledged data transfer ready state is entered.

9.3.2.2 Unacknowledged Data Transfer Ready State

In the unacknowledged data transfer ready, unacknowledged mode data can be exchanged between the entities. Upon reception of an CRLC-CONFIG-Req from higher layer the RLC entity is terminated and the null state is entered.

9.3.2.3 Local Suspend State

Upon reception of CRLC-SUSPEND-Req from higher layer (RRC) the RLC entity is suspended and the Local Suspend state is entered. In the Local Suspend state RLC shall not send RLC-PDUs with SN>=VT(US)+N. Upon reception of CRLC-RESUME-Req from higher layer (RRC) the RLC entity is resumed and the Data Transfer Ready state is entered.

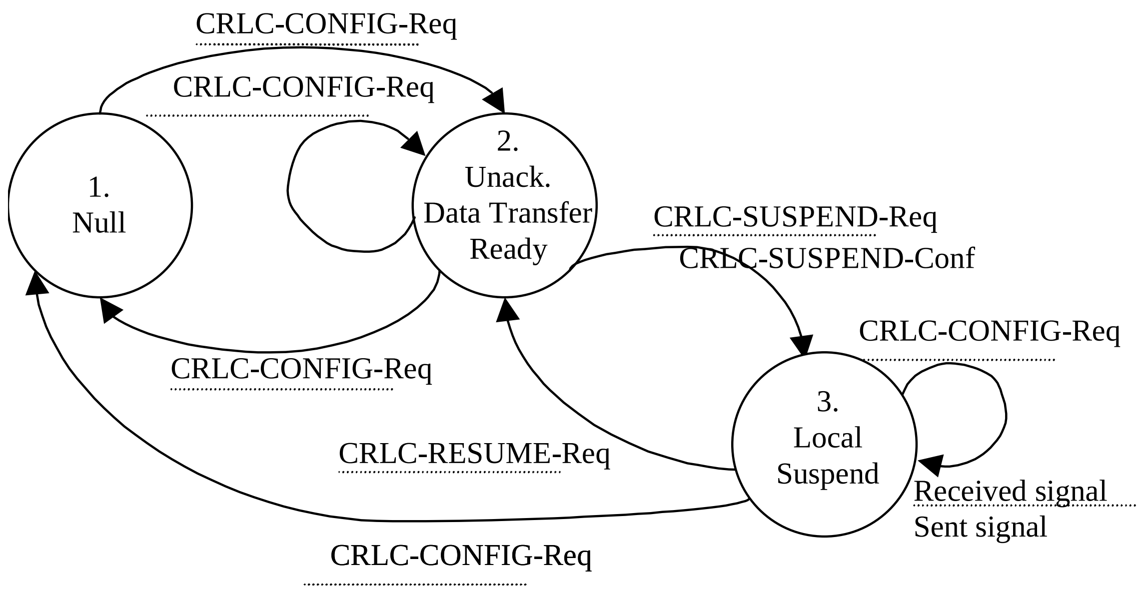


Figure 9.17: The state model for unacknowledged mode entities

9.3.3 State model for acknowledged mode entities

Figure 9.18 illustrates the state model for the acknowledged mode RLC entity (both transmitting and receiving). An acknowledged mode entity can be in one of following states.

9.3.3.1 Null State

In the null state the RLC entity does not exist and therefore it is not possible to transfer any data through it.

Upon reception of an CRLC-CONFIG-Req from higher layer the RLC entity is created and acknowledged data transfer ready state is entered.

9.3.3.2 Acknowledged Data Transfer Ready State

In the acknowledged data transfer ready state, acknowledged mode data can be exchanged between the entities. Upon reception of a CRLC-CONFIG-Req from higher layer the RLC entity is terminated and the null state is entered.

Upon errors in the protocol, the RLC entity sends a RESET PDU to its peer and enters the reset pending state.

Upon reception of a RESET PDU, the RLC entity resets the protocol (resets the state variables in 9.4 to their initial value and resets configurable parameters to their configured value, increments the hyper frame number if the RSN field indicates that the RESET PDU is not a retransmitted RESET PDU) and responds to the peer entity with a RESET ACK PDU.

Upon reception of a RESET ACK PDU, the RLC takes no action.

9.3.3.3 *Reset* Pending State

In the reset pending state the entity waits for a response from its peer entity and no data can be exchanged between the entities. Upon reception of CRLC-CONFIG-Req from higher layer the RLC entity is terminated and the null state is entered.

Upon reception of a RESET ACK PDU with the same RSN value as in the corresponding RESET PDU, the RLC entity resets the protocol (resets the state variables in 9.4 to their initial value, resets configurable parameters to their configured value, increments the hyper frame number) and one of the following state transitions take place.

The RLC entity enters the acknowledged data transfer ready state if Reset Pending State was entered from Acknowledged Data Transfer Ready State or if Reset Pending State was entered from Local Suspend State and a CRLC-RESUME–Req was received in Reset Pending State.

The RLC entity enters into Local Suspend State if Reset Pending State was entered from Local Suspend State or if Reset Pending State was entered from Acknowledged Data Transfer Ready State and a CRLC-SUSPEND-Req was received in Reset Pending State.

Upon reception of a RESET ACK PDU with a different RSN value as in the corresponding RESET PDU the RESET ACK PDU is discarded..

Upon reception of a RESET PDU, the RLC entity resets the protocol (resets the state variables in 9.4 to their initial value, resets configurable parameters to their configured value, increments the hyper frame number if the RSN field indicates that the RESET PDU is not a retransmitted RESET PDU), sends a RESET ACK PDU and stays in the reset pending state.

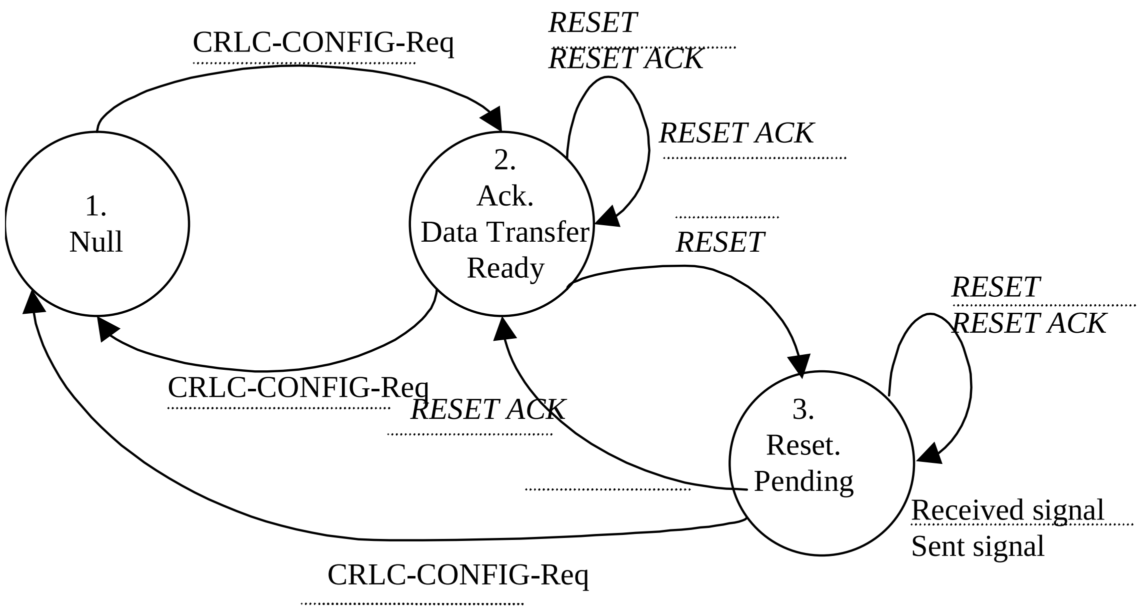


Figure 9.18: The state model for the acknowledged mode entities when reset is performed

9.3.3.4 Local Suspend State

Upon reception of CRLC-SUSPEND-Req from higher layer (RRC) in Acknowledge Data Transfer Ready State the RLC entity is suspended and the Local Suspend state is entered. In the Local Suspend state RLC shall not send a RLC-PDUs with a SN>=VT(S)+N. Upon reception of CRLC-RESUME-Req from higher layer (RRC) in this state, the RLC entity is resumed and the Data Transfer Ready state is entered.

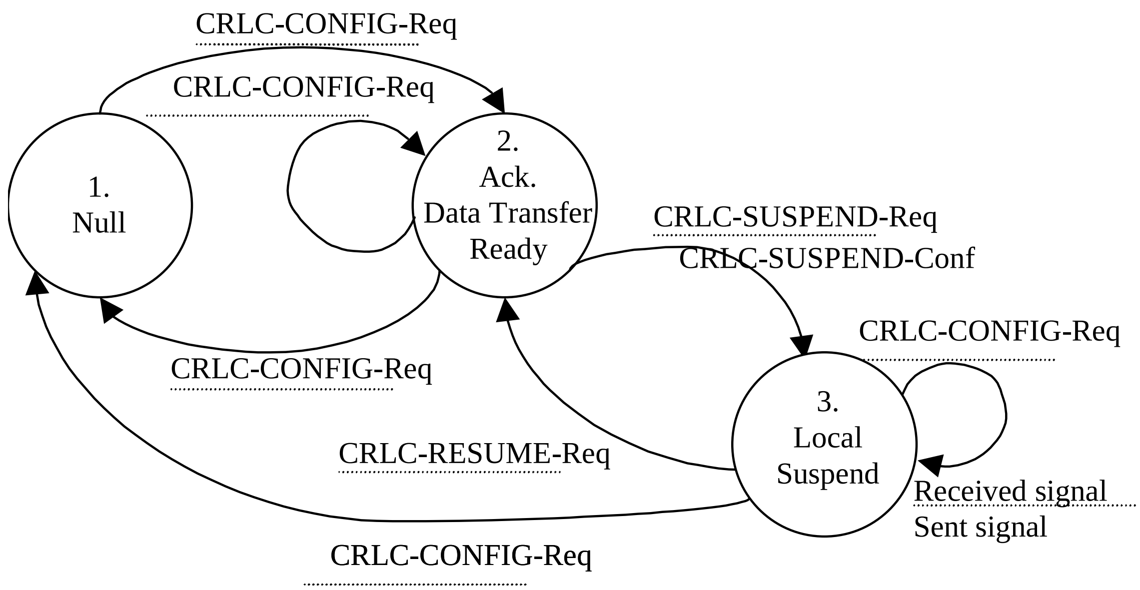


Figure 9.19: The state model for the acknowledged mode entities when local suspend is performed

9.4 State variables

This sub-clause describes the state variables used in the specification of the peer-to-peer protocol. PUs are sequentially and independently numbered and may have the value 0 through n minus 1 (where n is the modulus of the sequence numbers). The modulus equals 2^{12} for AM and 2^7 for UM; the sequence numbers cycle through the entire range: 0 through $2^{12} - 1$ for AM and 0 through $2^7 - 1$ for UM. All arithmetic operations on the following state variables and sequence numbers contained in this specification are affected by the modulus: VT(S), VT(A), VT(MS), VR(R), VR(H), VR(MR), VT(US) and VR(US). When performing arithmetic comparisons of transmitter variables, VT(A) is assumed to be the base. When performing arithmetic comparisons of receiver variables, VR(R) is assumed to be the base.

The RLC maintains the following state variables at the transmitter.

a) VT(S) - Send state variable.

The sequence number of the next PU to be transmitted for the first time (i.e. excluding retransmission). It is updated after transmission of a PDU, which includes not earlier transmitted PUs and after transmission of a MRW SUFI. The initial value of this variable is 0.

b) VT(A) - Acknowledge state variable.

The sequence number of the next in-sequence PU expected to be acknowledged, which forms the lower edge of the window of acceptable acknowledgements. VT(A) is updated based on receipt of a STATUS PDU including an ACK and/or MRW_ACK super-field. The initial value of this variable is 0.

c) VT(DAT).

This state variable counts the number of times a PU has been transmitted. There is one VT(DAT) for each PU and it is incremented each time the PU is transmitted. The initial value of this variable is 0.

d) VT(MS) - Maximum Send state variable.

The sequence number of the first PU not allowed by the peer receiver [i.e. the receiver will allow up to VT(MS) – 1], VT(MS) = VT(A) + Tx_Window_Size. This value represents the upper edge of the transmit window. The transmitter shall not transmit a new PU if VT(S) $\geq$ VT(MS). VT(MS) is updated based on receipt of a STATUS PDU including an ACK and/or MRW_ACK and/or a WINDOW super-field. The PU with SN VT(S)-1 can be transmitted also when VT(S)>VT(MS).

e) VT(US) – UM data state variable.

This state variable gives the sequence number of the next UMD PDU to be transmitted. It is updated each time a UMD PDU is transmitted. The initial value of this variable is 0.

f) VT(PU).

This state variable is used when the poll every Poll_PU PU function is used. It is incremented with 1 for each PU that is transmitted. It should be incremented for both new and retransmitted PUs. When it reaches Poll_PU a new poll is transmitted and the state variable is set to zero. The initial value of this variable is 0.

g) VT(SDU).

This state variable is used when the poll every Poll_SDU SDU function is used. It is incremented with 1 for each SDU that is transmitted. When it reaches Poll_SDU a new poll is transmitted and the state variable is set to zero. The poll bit should be set in the PU that contains the last segment of the SDU. The initial value of this variable is 0.

h) VT(RST) - Reset state variable.

It is used to count the number of times a RESET PDU is transmitted. VT(RST) is incremented with 1 each time a RESET PDU is transmitted. VT(RST) is reset only upon the reception of a RESET ACK PDU, i.e. VT(RST) is not reset when a RLC reset occurs which was initiated from the peer RLC entity. The initial value of this variable is 0.

i) VT(MRW) – MRW command send state variable.

It is used to count the number of times a MRW command is transmitted. VT(MRW) is incremented with 1 each time an MRW command is transmitted. VT(MRW) is reset when the discard procedure is terminated. The initial value of this variable is 0.

The RLC maintains the following state variables at the receiver:

a) VR(R) - Receive state variable.

The sequence number of the next in-sequence PU expected to be received. It is set equal to SNmax+1 upon receipt of the next in-sequence PU, where SNmax is the sequence number of the highest received in-sequence PU. The initial value of this variable is 0.

b) VR(H) - Highest expected state variable.

The sequence number of the highest expected PU. This state variable is set equal to SN+1 onlywhen a new PU is received with $VR(MR)>SN\geq VR(H)$. The initial value of this variable is 0.

c) VR(MR) - Maximum acceptable Receive state variable.

The sequence number of the first PU not allowed by the receiver [i.e. the receiver will allow up to VR(MR) – 1], VR(MR) = VR(R) + Rx_Window_Size. The receiver shall discard PUs with $SN \geq VR(MR)$.

d) VR(US) - Receiver Send Sequence state variable.

The sequence number of the next PDU to be received. It shall set equal to SN + 1 upon reception of a PDU. The initial value of this variable is 0.

e) VR(EP) – Estimated PDU Counter state variable.

The number of PUs that should be received yet as a consequence of the transmission of the latest status report. In acknowledged mode, this state variable is updated at the end of each transmission time interval. It is decremented by the number of PUs that should have been received during the transmission time interval. If VR(EP) is equal to zero, then check if all PUs requested for retransmission in the latest status report have been received.

9.5 Timers

a) Timer_Poll.

This timer is only used when the poll timer trigger is used. It is started when the transmitting side sends a poll to the peer entity. The timer is stopped when receiving a STATUS PDU that contains an acknowledgement of all AMD PDUs with SN up to and including VT(S)-1 at the time the poll was transmitted (or a negative acknowledgement of the same PU). The value of the timer is signalled by RRC.

If the timer expires and no STATUS PDU fulfilling the criteria above has been received, the receiver is polled once more (either by the transmission of a PDU which was not yet sent, or by a retransmission) and the timer is restarted with a new value of VT(S)-1. If there is no PU to be transmitted and all PUs have already been acknowledged, the receiver shall not be polled.

If a new poll is sent when the timer is running it is restarted, with a new value of VT(S)-1.

b) Timer_Poll_Prohibit.

This timer is only used when the poll prohibit function is used. It is used to prohibit transmission of polls within a certain period. A poll shall be delayed until the timer expires if a poll is triggered when the timer is active. Only one poll shall be transmitted when the timer expires even if several polls were triggered when the timer was active. If there is no PU to be transmitted and all PUs have already been acknowledged, a poll shall not be transmitted. This timer will not be stopped by a STATUS PDU. The value of the timer is signalled by RRC.

c) Timer_EPC.

This timer is only used when the EPC function is used and it accounts for the roundtrip delay, i.e. the time when the first retransmitted PU should be received after a status report has been sent. The timer is started when the first STATUS PDU of a status report is transmitted and when it expires EPC can start decrease (see subclause 9.7.3). The value of the timer is signalled by RRC.

d) Timer_Discard.

This timer is used for the SDU discard function. In the transmitter, the timer is activated upon reception of a SDU from higher layer. One timer is used for each SDU that is received from higher layer. If the SDU has not been acknowledged and/or transmitted when the timer expires, the SDU is discarded. Following which, if the SDU discard function uses explicit signalling, a Move Receiving Window request is sent to the receiver. The value of the timer is signalled by RRC.

e) Timer_Poll_Periodic.

This timer is only used when the timer based polling is used. The timer is started when the RLC entity is created. Each time the timer expires a poll is transmitted (either by the transmission of a PDU which was not yet sent, or by a retransmission) and the timer is restarted. If there is no PU to be transmitted and all PUs have already been acknowledged, a poll shall not be transmitted and the timer shall only be restarted. The value of the timer is signalled by RRC.

f) Timer_Status_Prohibit.

This timer is only used when the STATUS prohibit function is used. It prohibits the receiving side from sending status reports containing any of the SUFIs LIST, BITMAP, RLIST or ACK. The timer is started when the last STATUS PDU in a status report is transmitted and no new status report containing the mentioned SUFIs can be transmitted before the timer has expired. The timer does not prohibit transmission of the SUFIs MRW, MRW_ACK, WINDOW or NO_MORE. The value of the timer is signalled by RRC.

g) Timer_Status_Periodic.

This timer is only used when timer based status report sending is used. The timer is started when the RLC entity is created. Each time the timer expires a status report is transmitted and the timer is restarted. The value of the timer is signalled by RRC.

h) Timer_RST.

This timer is used to detect the loss of RESET ACK PDU from the peer RLC entity. This timer is set when the RESET PDU is transmitted. It will only be stopped upon reception of RESET ACK PDU, i.e. this timer is not stopped when an RLC reset occurs which was initiated from the peer RLC entity. If it expires, RESET PDU will be retransmitted. The value of the timer is signalled by RRC.

i) Timer_MRW.

This timer is used as part of the Move Receiving Window protocol. It is used to trigger the retransmission of a status report containing an MRW SUFI field. The timer is started when the STATUS PDU containing the MRW SUFI is first transmitted. Each time the timer expires the MRW SUFI is retransmitted and the timer is restarted (when the STATUS PDU containing the MRW SUFI is retransmitted). It shall be stopped when one of the termination criteria for the SDU discard is fulfilled. The value of the timer is signalled by RRC.

9.6 Protocol Parameters

The values of the protocol parameters in this section are signalled by RRC.

a) MaxDAT.

It is the maximum value for the number of retransmissions of a PU. This parameter is an upper limit of counter VT(DAT). When the value of VT(DAT) comes to MaxDAT, error recovery procedure will be performed.

b) Poll_PU.

This parameter indicates how often the transmitter should poll the receiver in case of polling every Poll_PU PU. This is an upper limit for the VT(PU) state variable, when VT(PU) reaches Poll_PU a poll is transmitted to the peer entity.

c) Poll_SDU.

This parameter indicates how often the transmitter should poll the receiver in case of polling every Poll_SDU SDU. This is an upper limit for the VT(SDU) state variable, when VT(SDU) reaches Poll_SDU a poll is transmitted to the peer entity.

d) Poll_Window.

This parameter indicates when the transmitter should poll the receiver in case of performing window-based polling. A poll is triggered for each PUwhen:

1) VT(S)<VT(MS), Tx_Window_Size>0, and

$$\left[1 - \frac{(Tx\_Window\_Size + \mathrm{VT(MS)} - \mathrm{VT(S)} - 1) \bmod Tx\_Window\_Size}{Tx\_Window\_Size} \right] * 100 > Poll\_Window$$

2) VT(S)≥VT(MS), and Tx_Window_Size>0

When Tx_Window_Size=0, the transmitter does not perform window-based polling.

e) MaxRST.

It is the maximum value for the number of retransmission of RESET PDU. This parameter is an upper limit of counter VT(RST). When the value of VT(RST) comes to MaxRST, the higher layer (RRC) is notified.

f) Tx_Window_Size.

The maximum allowed transmitter window size.

g) Rx_Window_Size.

The maximum allowed receiver window size.

h) MaxMRW.

It is the maximum value for the number of retransmissions of a MRW command. This parameter is an upper limit of counter VT(MRW). When the value of VT(MRW) comes to MaxMRW, error recovery procedure will be performed.

9.7 Specific functions

9.7.1 Polling function for acknowledged mode transfer

The transmitter of AMD PDUs may poll the receiver for a status report (consisting of one or several STATUS PDUs). The Polling bit in the AMD PDU indicates the poll request. There are several triggers for setting the polling bit. The network (RRC) controls, which triggers should be used for each RLC entity. Following triggers are possible:

1) Last PU in buffer.

   The sender transmits a poll when the last PU available for transmission is transmitted.

2) Last PU in retransmission buffer.

   The sender transmits a poll when the last PU to be retransmitted is transmitted.

3) Poll timer.

   The timer Timer_Poll is started when a poll is transmitted to the receiver and if the criterion for stopping the timer has not occurred before the timer Timer_Poll expires a new poll is transmitted to the receiver.

4) Every Poll_PU PU.

   The sender polls the receiver every Poll_PU PU. Both retransmitted and new Pus shall be counted.

5) Every Poll_SDU SDU.

   The sender polls the receiver every Poll_SDU SDU.

6) Poll_Window% of transmission window.

   The sender polls the receiver when it has reached Poll_Window% of the transmission window.

7) Timer based.

   The sender polls the receiver periodically.

Either the trigger "Last PU in buffer" and "Last PU in retransmission buffer" or "Timer based" can be chosen to avoid deadlock for every RLC entity. The network also controls if the poll prohibit function shall be used. The poll bit shall be set to 0 if the poll prohibit function is used and the timer Timer_Poll_Prohibit is active. This function has higher priority than any of the above mentioned triggers.

9.7.2 STATUS transmission for acknowledged mode

The receiver of AMD PDUs transmits status reports (each status report consists of one or several STATUS PDUs) to the sender in order to inform about which PUs that have been received and not received. There are several triggers for sending a status report. The network (RRC) controls which triggers should be used for each RLC entity, except for one, which is always present. The receiver shall always send a status report when receiving a poll request. Except for that trigger following triggers are configurable:

1) Detection of missing PU(s).

   If the receiver detects one or several missing PUs it shall send a status report to the sender.

2) Timer based STATUS transfer.

   The receiver transmits a status report periodically to the sender. The timer Timer_Status_Periodic controls the time period.

3) The EPC mechanism.

   The Timer EPC is started when the first STATUS PDU of a status report is transmitted to the peer entity. If not all PUs requested for retransmission have been received before the Timer EPC has expired a new status report is transmitted to the peer entity. A more detailed description of the EPC mechanism is given in subclause 9.7.4.

There are two functions that can prohibit the receiver from sending a status report. The network (RRC) controls which functions should be used for each RLC entity. If any of the following functions is used the sending of the status report shall be delayed, even if any of the conditions above are fulfilled:

1) STATUS prohibit.

   The Timer_Status_Prohibit is started when the last STATUS PDU of a status report is transmitted to the peer entity. As long as the timer is running the receiving side is not allowed to send a status report to the peer entity. If a status report was triggered while the timer was running, the status report is transmitted after the timer has expired. The receiver shall only send one status report, even if there are several triggers when the timer running. This timer only prohibits the transmission of status reports containing any of the SUFIs LIST, BITMAP, RLIST or ACK. Status reports containing other SUFIs are not prohibited.

2) The EPC mechanism.

   If the EPC mechanism is active and the sending of a status report is triggered it shall be delayed until the EPC mechanism has ended. The receiver shall only send one status report, even if there are several triggers when the timer is active or the counter is counting down. This mechanism only prohibits the transmission of status reports containing any of the SUFIs LIST, BITMAP, RLIST or ACK. Status reports containing other SUFIs are not prohibited.

9.7.3 SDU discard function

The SDU discard function allows to discharge RLC PDU from the buffer on the transmitter side, when the transmission of the RLC PDU does not success for a long time. The SDU discard function allows to avoid buffer overflow. There will be several alternative operation modes of the RLC SDU discard function, and which discard function to use will be given by the QoS requirements of the Radio Access Bearer.

The following is a list of operation modes for the RLC SDU discard function.

Table 9.2: List of criteria's that control when to perform SDU discard

Operation mode	Presence
Timer based discard, with explicit signalling	Network controlled
Timer based discard, without explicit signalling	Network controlled
SDU discard after MaxDAT number of retransmissions	Network controlled

9.7.3.1 Timer based discard, with explicit signalling

This alternative uses a timer based triggering of SDU discard (Timer_Discard). This makes the SDU discard function insensitive to variations in the channel rate and provides means for exact definition of maximum delay. However, the SDU loss rate of the connection is increased as SDUs are discarded.

For every SDU received from a higher layer, timer monitoring of the transmission time of the SDU is started. If the transmission time exceeds a predefined value for a SDU in acknowledged mode RLC, this SDU is discarded in the transmitter and a Move Receiving Window (MRW) command is sent to the receiver so that AMD PDUs carrying that SDU are discarded in the receiver and the receiver window is updated accordingly. Note that when the concatenation function is active, PDUs carrying segments of other SDUs that have not timed out shall not be discarded.

The MRW command is defined as a super-field in the RLC STATUS PDU (see subclause 9.2), and piggy backed to status information of transmissions in the opposite direction. If the MRW command has not been acknowledged by receiver, it will be retransmitted. Therefore, SDU discard variants requiring peer-to-peer signalling are only possible for full duplex connections.

9.7.3.2 Timer based discard, without explicit signalling

This alternative uses the same timer based trigger for SDU discard (Timer_Discard) as the one described in the subclause 9.7.3.1. The difference is that this discard method does not use any peer-to-peer signalling. This function is applied only for unacknowledged and transparent mode RLC and peer-to-peer signalling is never needed. The SDUs are simply discarded in the transmitter, once the transmission time is exceeded.

9.7.3.3 SDU discard after MaxDAT number of retransmissions

This alternative uses the number of retransmissions as a trigger for SDU discard, and is therefore only applicable for acknowledged mode RLC. This makes the SDU discard function dependent of the channel rate. Also, this variant of the SDU discard function strives to keep the SDU loss rate constant for the connection, on the cost of a variable delay. SDU discard is triggered at the transmitter, and a MRW command is necessary to convey the discard information to the receiver, like in the timer based discard with explicit signalling.

9.7.4 The Estimated PDU Counter

The Estimated PDU Counter is a mechanism used for scheduling the retransmission of status reports in the receiver side. With this mechanism, the receiver will send a new status report in which it requests for PUs not yet received. The time between two subsequent status report retransmissions is not fixed, but it is controlled by the Estimated PDU Counter (EPC), which adapt this time to the current bit rate, indicated in the TFI, in order to minimise the delay of the status report retransmission.

The EPC is a counter, which is decremented every transmission time interval with the estimated number of PUs that should have been transmitted during that transmission time interval. When the receiver detects that PDUs are missing it generates and sends a status report to the transmitter and sets the EPC equal to the number of requested PUs.

A special timer, called EPC timer, controls the maximum time that the EPC needs to wait before it will start counting down. This timer starts immediately after a transmission of a retransmission request from the receiver (when the first STATUS PDU of the status report is transmitted). The EPC timer typically depends on the roundtrip delay, which consists of the propagation delay, processing time in the transmitter and receiver and the frame structure. This timer can also be implemented as a counter, which counts the number of 10 ms radio frames that could be expected to elapse before the first requested AMD PDU is received.

When the EPC is equal to zero and not all of these requested PUs have been received correctly, a new status report will be transmitted and the EPC will be reset accordingly. The EPC timer will be started once more.

9.7.5 Multiple payload units in an RLC PDU

The possibility to include multiple payload units (PU) into one RLC AMD PDU is part of the service capabilities of a UE in acknowledged mode. For Release 99, there shall be only one PU per AMD PDU.

A payload unit is the smallest unit that can be separately addressed for retransmission and is of fixed size, containing data and optionally, length indicators and/or padding. The padding space of a PU can be used to piggyback STATUS PDUs.

The size of the PU is set by the RRC.

9.7.6 Local Suspend function for acknowledged mode transfer

The higher layer (RRC) may suspend the RLC entity. The CRLC-SUSPEND-Req indicates this request. The RLC entity shall, when receiving this request, not send RLC PDUs with SN>=VT(S)+N (N is given by the CRLC_SUSPEND-Req primitive). The RLC entity shall acknowledge the CRLC-SUSPEND-Req ordering a suspend with a CRLC-SUSPEND-Conf with the current value of VT(S). The suspend state is left when a CRLC-RESUME-Req primitive indicating resume is received.

10 Handling of unknown, unforeseen and erroneous protocol data

The list of error cases is reported below:

a) Inconsistent state variables.

If the RLC entity receives a PDU including "erroneous Sequence Number", state variables between peer entities may be inconsistent. Following shows "erroneous Sequence Number" examples:

- Each Sequence Number of missing PU informed by SUFI LIST, BITMAP or RLIST is not within the value between "Acknowledge state variable(VT(A))" and "Send state variable(VT(S))", and

-- LSN of SUFI ACK is not within the value between "Acknowledge state variable(VT(A))" and "Send state variable(VT(S))".

In case of error situations the following actions are foreseen:

1) RLC entity should use RESET procedure in case of an unrecoverable error.

2) RLC entity should discard invalid PDU.

3) RLC entity should notify upper layer of unrecoverable error occurrence in case of failed retransmission.

b) Inconsistent status indication of a PU

If a received STATUS PDU indicates different status for the same PU, then the transmitter shall discard the STATUS PDU.

11 Elementary procedures

11.1 Transparent mode data transfer procedure

11.1.1 Purpose

The transparent mode data transfer procedure is used for transferring of data between two RLC peer entities, which are operating in transparent mode. Figure 11.1 below illustrates the elementary procedure for transparent mode data transfer. The sender can be either the UE or the network and the receiver is either the network or the UE.

Figure 11.1: Transparent mode data transfer procedure

11.1.2 Initiation

The sender initiates this procedure upon a request of transparent mode data transfer from higher layer. When the sender is in data transfer ready state it shall put the data received from the higher layer into TrD PDUs. If required RLC shall perform segmentation.

Channels that can be used are DTCH, CCCH (uplink only), SHCCH (uplink only), BCCH and PCCH. The type of logical channel depends on if the RLC entity is located in the user plane (DTCH) or in the control plane (CCCH/BCCH/SHCCH/PCCH). One or several PDUs may be transmitted in each transmission time interval (TTI) and MAC decides how many PDUs shall be transmitted in each TTI. In the UE, the PDUs that can not be transmitted in a TTI (i.e. MAC has indicated that some of the available PDUs can not be transmitted) shall be buffered according to the discard configuration set by RRC.

11.1.2.1 TrD PDU contents to set

The TrD PDU includes a complete SDU or a segment of an SDU. How to perform the segmentation is decided upon when the service is established. No overhead or header is added, instead segmentation is done based on which of the transport formats of the transport channel that will be used. A particular transport format informs the receiver how the segmentation was performed.

11.1.3 Reception of TrD PDU

Upon reception of a TrD PDU, the receiving entity reassembles (if segmentation was performed) the PDUs into RLC SDUs. RLC delivers the RLC SDUs to the higher layer through the Tr-SAP.

11.1.4 Abnormal cases

11.1.4.1 Undefined SDU size at receiver

If the TrD PDUs are reassembled to a SDU which have a size that is not allowed the SDU shall be discarded.

11.1.4.2 SDU discard without explicit signalling

Upon expiry of the Timer_Discard on the sender side the sender shall discard all PDUs that contain segments of the associated SDU.

11.2 Unacknowledged mode data transfer procedure

11.2.1 Purpose

The unacknowledged mode data transfer procedure is used for transferring data between two RLC peer entities, which are operating in unacknowledged mode. Figure 11.2 below illustrates the elementary procedure for unacknowledged mode data transfer. The sender can be either the UE or the network and the receiver is either the network or the UE.

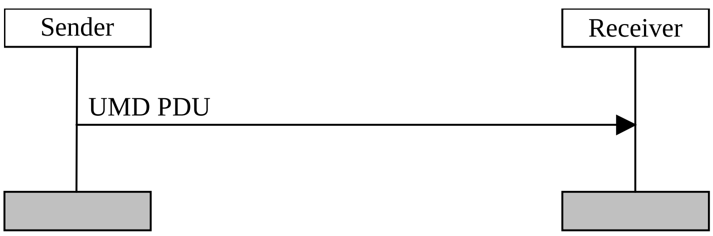


Figure 11.2: Unacknowledged mode data transfer procedure

11.2.2 Initiation

The sender initiates this procedure upon a request of unacknowledged mode data transfer from higher layer.

When the sender is in data transfer ready state it shall segment the data received from the higher layer into PDUs.

Channels that can be used are DTCH, DCCH, CCCH (downlink only), CTCH, SHCCH (downlink only). The type of logical channel depends on if the RLC entity is located in the user plane (DTCH, CTCH) or in the control plane (DCCH/CCCH(downlink only)/SHCCH(downlink only)). One or several PDUs may be transmitted in each transmission time interval (TTI) and MAC decides how many PDUs shall be transmitted in each TTI. In the UE, the PDUs that can not be transmitted in a TTI (i.e. MAC has indicated that some of the available PDUs can not be transmitted) shall be buffered according to the discard configuration set by RRC.

The VT(US) state variable shall be updated for each UMD PDU that is transmitted.

11.2.2.1 UMD PDU contents to set

The Sequence Number field shall be set equal to VT(US).

The Extension bit shall be set to 1 if the next field is a length indicator field, otherwise it shall be set to zero.

One length indicator field shall be included for each end of a SDU that the PDU includes. The length indicator shall be set equal to the number octets between the end of the header fields and the end of the segment. If padding is needed another length indicator shall be added. If the PDU is exactly filled with the last segment of a SDU and there is no room for a length indicator field a length indicator field set to only 0's shall be included in the next PDU.

11.2.3 Reception of UMD PDU

Upon reception of a UMD PDU the receiver shall update VR(US) state variable according to the received PDU(s).

The PDUs are reassembled into RLC SDUs. If a PDU with sequence number < VR(US) is missing then all SDUs that have segments in this PDU shall be discarded. RLC delivers the RLC SDUs to the higher layer through the UM-SAP.

11.2.4 Abnormal cases

11.2.4.1 Length Indicator value 1111110

Upon reception of an UMD PDU that contains Length Indicator value 1111110 or 111111111111110 ("piggybacked STATUS PDU", in case 7bit or 15 bit Length Indicator field is used, respectively) the receiver shall discard that UMD PDU. This Length Indicator value is not used in unacknowledged mode data transfer.

11.2.4.2 Invalid length indicator value

If the length indicator of a PDU has a value that is larger than the PDU size, the PDU shall be discarded and treated as a missing PDU.

11.2.4.3 SDU discard without explicit signalling

Upon expiry of the Timer_Discard on the sender side the sender shall discard all PDUs that contain segments of the associated SDU. If the concatenation function is active, PDUs carrying segments of other SDUs that have not timed out shall not be discarded. The state variable VT(US) shall be updated.

11.3 Acknowledged mode data transfer procedure

11.3.1 Purpose

The acknowledged mode data transfer procedure is used for transferring of data between two RLC peer entities, which are operating in acknowledged mode. Figure 11.3 below illustrates the elementary procedure for acknowledged mode data transfer. The sender can be either the UE or the network and the receiver is either the network or the UE.

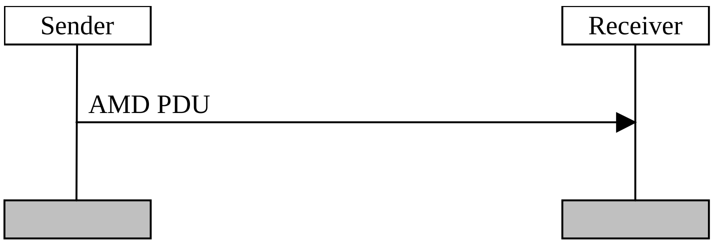


Figure 11.3: Acknowledged mode data transfer procedure

11.3.2 Initiation

The sender initiates this procedure upon a request of acknowledged mode data transfer from higher layer or upon retransmission of PUs. Retransmitted PUs have higher priority than PUs transmitted for the first time.

The sender is only allowed to retransmit PUs that have been indicated missing by the receiver. An exception is the PU with SN VT(S)-1 can be retransmitted. In addition, a PU that has not yet been acknowledged, may be retransmitted if the configured transmitter window size is less than 2048.

RLC shall segment the data received from the higher layer into PUs. When the sender is in data transfer ready state one or several PUs are included in one AMD PDU, which is sent to the receiver. The PDUs shall be transmitted on the DCCH logical channel if the sender is located in the control plane and on the DTCH if it is located in the user plane. One or several PDUs may be transmitted in each transmission time interval (TTI) and MAC decides how many PDUs shall be transmitted in each TTI. In the UE, the PDUs that can not be transmitted in a TTI (i.e. MAC has indicated that some of the available PDUs can not be transmitted) shall be buffered according to the discard configuration set by RRC.

The VT(DAT) state variables shall be updated for each AMD PDU that is transmitted. The PDU shall not include any PU with Sequence Number ≥ VT(MS) , except the PU with sequence number VT(S)-1 which may be included also when VT(S) > VT(MS).

If the poll bit is set in any of the AMD PDUs and the timer Timer_Poll shall be used the sender shall start the timer Timer_Poll when the PDU with the set poll bit is delivered to MAC.

If timer based SDU discard is used the timer Timer_Discard shall be started when the RLC entity receives an SDU from higher layer. One timer is used for each SDU that is received from higher layer.

If the trigger for polling, "Every Poll_PU PU", is used the VT(PU) shall be increased by 1 for each PU that is transmitted.

If the trigger for polling, "Every Poll_SDU SDU", is used the VT(SDU) shall be increased by 1 for each SDU that is transmitted.

11.3.2.1 AMD PDU contents to set

If the PDU is transmitted for the first time, the Sequence Number field shall be set equal to VT(S) and VT(S) shall be updated.

The setting of the Polling bit is specified in subclause 11.3.2.1.1.

One length indicator field shall be included for each end of a SDU that the PDU includes. The length indicator shall be set equal to the number of octets between the end of the header fields and the end of the segment. If the PDU is exactly filled with the last segment of a SDU and there is no room for a length indicator field a length indicator field set to only 0's shall be included in the next PDU. How to perform the segmentation of a SDU is specified in subclause 11.3.2.1.2.

11.3.2.1.1 Setting of the Polling bit

The Polling bit shall be set to 1 if any of following conditions are fulfilled except when the poll prohibit function is used and the timer Timer_Poll_Prohibit is active (the different triggers are described in 9.7.4):

1) Last PU in buffer is used and the last PU available for transmission is transmitted.

2) Last PU in retransmission buffer is used and the last PU to be retransmitted is transmitted.

3) Poll timer is used and timer Timer_Poll has expired.

4) Every Poll_PU PU is used and when VT(PU)=Poll_PU.

5) Every Poll_SDU is used and VT(SDU)=Poll_SDU and the PDU contains the last segment that SDU.

6) Poll_Window(%) of transmission window is used, Tx_Window_Size>0, VT(S)<VT(MS), and

$$\left(1 - \frac{(Tx\_Window\_Size + \mathrm{VT(MS)} - \mathrm{VT(S)} - 1) \bmod Tx\_Window\_Size}{Tx\_Window\_Size}\right) * 100 > Poll\_Window$$

.

7) Poll_Window (%) of transmission window is used, Tx_Window_Size>0 and VT(S)≥VT(MS).

8) Timer based polling is used and Timer_Poll_Periodic has expired.

9) Poll prohibit shall be used, the timer Timer_Poll_Prohibit has expired and one or several polls were prohibited during the time Timer_Poll_Prohibit was active.

11.3.2.1.2 Segmentation of a SDU

Upon reception of a SDU, RLC shall segment the SDU to fit into the fixed size of a PU. The segments are inserted in the data field of a PU. A length indicator shall be added to each PU that includes a border of a SDU, i.e. if a PU does not contain a length indicator the SDU continues in the next PU. The length indicator indicates where the border occurs in the PU. The data after the indicated border can be either a new SDU, padding or piggybacked information. If padding or piggybacking is added another length indicator shall be added, see subclause 9.2.2.8.

11.3.3 Reception of AMD PDU by the receiver

Upon reception of a AMD PDU the receiver shall update VR(R), VR(H) and VR(MR) state variables according to the received PU(s).

If any of the PUs include a Polling bit set to 1 the STATUS PDU transfer procedure shall be initiated.

If the detection of missing PU(s) shall be used and the receiver detects that a PU is missing the receiver shall initiate the STATUS PDU transfer procedure.

11.3.4 Abnormal cases

11.3.4.1 Timer_Poll timeout

Upon expiry of the Timer_Poll the sender shall retransmit the poll. The poll can be retransmitted in either a new PDU or a retransmitted PDU.

11.3.4.2 Receiving a PU outside the receiving window

Upon reception of a PU with SN<VR(R) or SN≥VR(MR) the receiver shall discard the PU. The poll bit shall be considered even if a complete PDU is discarded.

11.3.4.3 Timer_Discard timeout

11.3.4.3.1 SDU discard with explicit signalling

Upon expiry of Timer_Discard the sender shall initiate the SDU discard with explicit signalling procedure.

11.3.4.4 VT(DAT) > MaxDAT

If SDU discard after MaxDAT number of retransmission is used and VT(DAT) > MaxDAT for any PU the sender shall initiate the SDU discard with explicit signalling procedure for the SDUs to which the PU with VT(DAT)>MaxDAT belongs.

If the SDU discard is not used the sender shall initiate the RLC reset procedure when VT(DAT) > MaxDAT.

11.3.4.5 Invalid length indicator value

If the length indicator of a PU has a value that is larger than the PU size, the PU shall be discarded and treated as a missing PU.

11.4 RLC reset procedure

11.4.1 Purpose

The RLC reset procedure is used to reset two RLC peer entities, which are operating in acknowledged mode.
Figure 11.4 below illustrates the elementary procedure for a RLC reset. The sender can be either the UE or the network and the receiver is either the network or the UE.

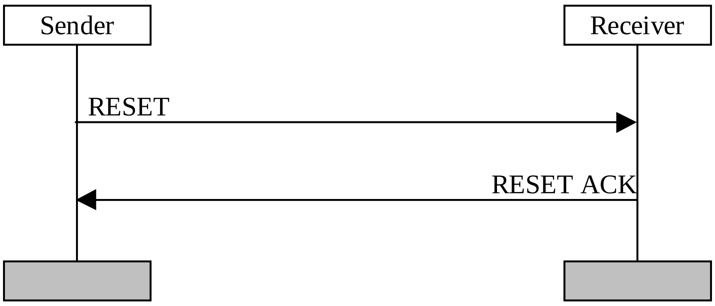


Figure 11.4: RLC reset procedure

11.4.2 Initiation

The procedure shall be initiated when a protocol error occurs.

The sender sends the RESET PDU when it is in data transfer ready state and enters reset pending state. The sender shall start the timer Timer_RST and increase VT(RST) with 1. The RESET PDU shall be transmitted on the DCCH logical channel if the sender is located in the control plane and on the DTCH if it is located in the user plane.

The RESET PDU has higher priority than data PDUs.

When a reset procedure has been initiated it can only be ended upon reception of a RESET ACK PDU with the same RSN value as in the corresponding RESET PDU, i.e., a reset procedure is not interrupted by the reception of a RESET PDU from the peer entity.

11.4.2.1 RESET PDU contents to set

The size of the RESET PDU shall be equal to one of the allowed PDU sizes. The RSN field shall indicate the sequence number of the RESET PDU. This sequence number is incremented every time a new RESET PDU is transmitted.

11.4.3 Reception of the RESET PDU by the receiver

Upon reception of a RESET PDU the receiver shall respond with a RESET ACK PDU. The receiver resets the state variables in 9.4 to their initial value and resets configurable parameters to their configured value.

In the received RESET PDU the Receiver shall check the value of RSN (Reset Sequence Number) field. If the value of the RSN field is different from the RSN value in the previously received RESET PDU the Receiver shall increase the value of the HFN by one.

If the value of the RSN is equal to the RSN value in the previously received RESET PDU, (i.e. the RESET PDU is a retransmitted RESET PDU) the value of the HFN shall not be increased and only a RESET ACK PDU shall be sent to the peer RLC entity.

The RESET ACK PDU shall be transmitted on the DCCH logical channel if the sender is located in the control plane and on the DTCH if it is located in the user plane.

The RESET ACK PDU has higher priority than data PDUs.

11.4.3.1 RESET ACK PDU contents to set

The size of the RESET ACK PDU shall be equal to one of the allowed PDU sizes. The RSN field shall always be set to the same value as in the corresponding RESET PDU.

11.4.4 Reception of the RESET ACK PDU by the sender

When the sender is in reset pending state and receives a RESET ACK PDU with the same RSN value as in the corresponding RESET PDU the Timer_RST shall be stopped and the value of the HFN shall be increased by one. The sender resets the state variables in 9.4 to their initial value and resets configurable parameters to their configured value. The sender shall enter data transfer ready state.

Upon reception of a RESET ACK PDU with a different RSN value as in the corresponding RESET PDU the RESET ACK PDU is discarded.

Upon reception of a RESET ACK PDU in data transfer ready state the RESET ACK PDU is discarded.

11.4.5 Abnormal cases

11.4.5.1 Timer_RST timeout

Upon expiry of Timer_RST the sender shall retransmit the RESET PDU and increase VT(RST) with 1. In the retransmitted RESET PDU the value of the RSN field shall not be incremented.

11.4.5.2 VT(RST) ≥ MaxRST

If VT(RST) becomes larger or equal to MaxRST the RRC layer shall be informed.

11.4.5.3 Reception of the RESET PDU by the sender

Upon reception of a RESET PDU in reset pending state the sender shall respond with a RESET ACK PDU. The sender resets the state variables in 9.4 to their initial value, resets configurable parameters to their configured value. However, VT(RST) and Timer_RST are not reset. The hyper frame number is incremented if the RSN field indicates that the RESET PDU is not a retransmitted RESET PDU. The sender shall stay in the reset pending state. The sender shall enter data transfer ready state only upon reception of a RESET ACK PDU with the same RSN value as in the corresponding RESET PDU.

11.5 STATUS report transfer procedure

11.5.1 Purpose

The status report transfer procedure is used for transferring of status information between two RLC peer entities, which are operating in acknowledged mode. Figure 11.5 below illustrates the elementary procedure for status report transfer. A status report consists of one or several STATUS PDUs. The receiver is the receiver of AMD PDUs and it is either the UE or the network and the sender is the sender of AMD PDUs and it is either the network or the UE.

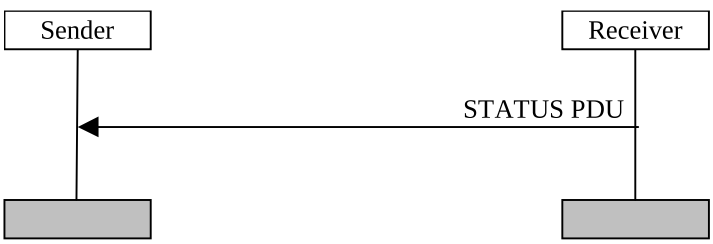


Figure 11.5: Status report transfer procedure

11.5.2 Initiation

The receiver in any of following cases initiates this procedure:

1) The poll bit in a received AMD PDU is set to 1.

2) Detection of missing PUs is used and a missing PU is detected.

3) The timer based STATUS transfer is used and the timer Timer_Status_Periodic has expired.

The receiver shall transmit a status report on the DCCH logical channel if the receiver is located in the control plane and on the DTCH if it is located in the user plane. Separate logical channels can be assigned for AMD PDU transfer and for Control PDU transfer.

The STATUS PDUs have higher priority than data PDUs.

There are two functions that can prohibit the receiver from sending a status report. If any of following conditions are fulfilled the sending of the status report shall be delayed, even if any of the conditions above are fulfilled:

1) STATUS prohibit is used and the timer Timer_Status_Prohibit is active.

   The status report shall be transmitted after the Timer_Status_Prohibit has expired. The receiver shall send only one status report, even if there are several triggers when the timer is running.

2) The EPC mechanism is used and the timer Timer_EPC is active or VR(EP) is counting down.

   The status report shall be transmitted after the VR(EP) has reached 0. The receiver send only one status report, even if there are several triggers when the timer is active or the counter is counting down.

If the timer based STATUS transfer shall be used and the Timer_Status_Periodic has expired it shall be restarted.

If the EPC mechanism shall be used the timer Timer_EPC shall be started and the VR(EP) shall be set equal to the number PUs requested to be retransmitted.

11.5.2.1 Piggybacked STATUS PDU

It is possible to piggyback a STATUS PDU on an AMD PDU. If a PDU includes padding a piggybacked STATUS PDU can be inserted instead of the padding. The sending of a piggybacked STATUS PDU follows the same rules as the sending of an ordinary STATUS PDU.

11.5.2.2 STATUS PDU contents to set

The size of the STATUS PDU shall be equal to one of the allowed PDU sizes. The information that needs to be transmitted in a status report can be split into several STATUS PDUs if one STATUS PDU does not accommodate all the information. A SUFI can not be split into several STATUS PDUs. Indication of the same PU shall not be given in more than one STATUS PDU of a STATUS report.

Which SUFI fields to use is implementation dependent, but the status report shall include information about PUs that have been received and information about all PUs detected as missing. No information shall be given for PUs with SN≥VR(H), i.e. PUs that have not yet reached the receiver.

Padding shall be inserted if the SUFI fields do not fill an entire STATUS PDU. If the PDU contains padding the last SUFI field shall be either an ACK SUFI or a NO_MORE SUFI. If there is no padding in the STATUS PDU, NO_MORE SUFI or ACK SUFI does not need to be included in the STATUS PDU.

11.5.3 Reception of the STATUS PDU by the sender

The sender shall upon reception of the STATUS PDU/piggybacked STATUS PDU update the state variables VT(A) and VT(MS) according to the received STATUS PDU/piggybacked STATUS PDU.

If the STATUS PDU includes negative acknowledged PUs the acknowledged data transfer procedure shall be initiated and the PUs shall be retransmitted. If a PU is indicated as missing more then once in a STATUS PDU, the PU shall be retransmitted only once. Retransmitted PUs have higher priority than new PUs.

11.5.4 Abnormal cases

11.5.4.1 EPC reaches zero and the requested PUs have not been received

If the EPC mechanism is used and VR(EP) has reached 0 and not all PUs requested for retransmission have been received the receiver shall:

- Retransmit the status report. The retransmitted status report may contain new or different SUFI fields in order to indicate that some PUs have been received and that some new have been lost.

11.6 SDU discard with explicit signalling procedure

11.6.1 Purpose

An SDU can be discarded with explicit signalling when MaxDAT number of retransmissions is reached or the transmission time exceeds a predefined value (Timer_Discard) for a SDU in acknowledged mode RLC. Move Receiving Window (MRW) command is sent to the receiver so that AMD PDUs carrying that SDU are discarded in the receiver and the receiver window is updated accordingly. Note that when the concatenation function is active, PDUs carrying segments of other SDUs that have not timed out shall not be discarded. If one or more segments of a SDU has been transmitted, the SDU shall not be discarded in the transmitter without notification to the receiver.

The MRW command is defined as a super-field in the RLC STATUS PDU, and can be piggybacked to status information of transmissions in the opposite direction.

Figure 11.6 below illustrates the elementary procedure for SDU discard with explicit signalling. The sender is the sender of AMD PDUs and it is either the UE or the network and the receiver is the receiver of AMD PDUs and it is either the network or the UE.

Figure 11.6: SDU discard with explicit signalling

11.6.2 Initiation

This procedure is initiated by the sender when the following conditions are fulfilled:

1) SDU discard with explicit signalling is used.

2) MaxDAT number of retransmissions is reached or Timer_Discard expires for a SDU in acknowledged mode RLC.

The sender shall discard all PUs that contain segments of the associated SDUs. If the concatenation function is active, PDUs carrying segments of other SDUs that have not timed out shall not be discarded. VT(A) shall be updated when the procedure is terminated, and VT(S) shall be updated when a new MRW SUFI is transmitted.

The sender shall transmit a status report on the DCCH logical channel if the sender is located in the control plane and on the DTCH if it is located in the user plane.

This status report is sent even if the 'STATUS prohibit' is used and the timer 'Timer_Status_Prohibit' or 'Timer_EPC'is active.

The STATUS PDUs have higher priority than data PDUs.

The sender shall start timer Timer_MRW. If a new SDU discard procedure is triggered when Timer_MRW is running, no new MRW SUFIs shall be sent before the current SDU discard procedure is terminated by one of the termination criteria.

11.6.2.1 Piggybacked STATUS PDU

It is possible to piggyback a STATUS PDU on an AMD PDU. If a PDU includes padding a piggybacked STATUS PDU can be inserted instead of the padding.

11.6.2.2 STATUS PDU contents to set

The size of the STATUS PDU shall be equal to one of the allowed PDU sizes. The discard information shall not be split into several MRW SUFIs.

The status report shall include the MRW SUFI, other SUFI fields can be used additionally. MRW SUFI shall convey information about the discarded SDU(s) to the receiver.

In order to discard a single SDU that ends in a PDU with SN≥ VT(A)+Configured_Tx_Window_Size, the LENGTH field in the MRW SUFI shall be set to "0000". If more then one SDU are discarded with the same MRW SUFI, at least the first discarded SDUs must end (i.e. the LI must be located) in a PDU with SN in the interval VT(A)≤ SN <VT(A)+Configured_Tx_Window_Size.

Padding shall be inserted if the SUFI fields do not fill the entire STATUS PDU. If the STATUS PDU contains padding the last SUFI field shall be either an ACK SUFI or a NO_MORE SUFI. If there is no padding in the STATUS PDU, NO_MORE SUFI or ACK SUFI does not need to be included in the STATUS PDU.

11.6.3 Reception of the STATUS PDU by the receiver

The receiver shall upon reception of the STATUS PDU/piggybacked STATUS PDU discard PUs and update the state variables VR(R), VR(H) and VR(MR) according to the received STATUS PDU/piggybacked STATUS PDU. Additionally the receiver should indicate the higher layers of all of the discarded SDUs.

The receiver shall initiate the transmission of a status report containing an MRW_ACK SUFI.

In the MRW_ACK SUFI, SN_ACK shall be set to the new value of VR(R), updated after reception of the MRW SUFI. The N field in the MRW_ACK SUFI shall be set to N_{LENGTH} field in the received MRW SUFI if the SN_ACK field is equal to SN_MRW_{LENGTH}. Otherwise N shall be set to 0.

The last discarded data byte is the byte indicated by the N_{LENGTH}:th LI field of the PU with sequence number SN_MRW_{LENGTH} and the succeeding data byte is the first data byte to be reassembled after the discard. When N_{LENGTH} = 0, the first data byte of the PU with sequence number SN_MRW_{LENGTH} is the first data byte to be reassembled after the discard.

If the MRW SUFI indicates an SN_MRW_i outside the interval VR(R)≤ SN_MRW_i <VR(MR) , the Rx shall consider the sequence number to be below VR(R), unless LENGTH="0000" or at least the first indicated SN_MRW_i in the MRW SUFI is within the interval VR(R)≤ SN_MRW_i <VR(MR), in which case the sequence number shall be considered to be above or equal to VR(MR).

11.6.4 Termination

The procedure is terminated in the sender in the following cases:

1. On the reception of a STATUS PDU which contains an MRW_ACK SUFI with SN_ACK > SN_MRW_{LENGTH}
2. On the reception of a STATUS PDU which contains an ACK SUFI indicating VR(R) > SN_MRW_{LENGTH}
3. On reception of a STATUS PDU which contains an MRW_ACK with SN_ACK = SN_MRW_{LENGTH} and N is equal to the N_{LENGTH} indicated in the transmitted MRW SUFI.

If one of the termination criteria above is fulfilled, Timer_MRW is stopped and the discard procedure is terminated.

When VT(MRW) reaches MaxMRW, the procedure is terminated and an RLC reset is performed.

11.6.5 Expiration of timer Timer_MRW

If Timer_MRW expires before the discard procedure is terminated,the MRW SUFI shall be retransmitted, VT(MRW) is incremented by one and Timer_MRW restarted. MRW SUFI shall be exactly the same as previously transmitted even though some new SDUs would have been discarded during the running of the Timer_MRW. If the retransmitted STATUS PDU contains other SUFIs than the MRW SUFI, the status information indicated by these SUFIs shall be updated.

11.6.6 Abnormal cases

11.6.6.1 Obsolete/corrupted MRW command

If the MRW command contains outdated information about the receiver window (receiver window already moved further than MRW command is indicating), the MRW command shall be discarded and a status report containing SUFI MRW_ACK shall be transmitted indicating the value of VR(R) and the N field shall be set to zero.

11.6.6.2 VT(MRW) equals MaxMRW

If the number of retransmission of a MRW command (i.e. VT(MRW)) reaches MaxMRW, an error indication shall be passed to RRC and RESET procedure shall be performed.

11.6.6.3 Reception of obsolete MRW_ACK

The received MRW_ACK shall be discarded in the following cases.

1. If timer Timer_MRW is not active.
2. If the SN_ACK field in the received MRW_ACK < SN_MRW_{LENGTH} in the transmitted MRW SUFI.
3. If the SN_ACK field in the received MRW_ACK is equal to the SN_MRW_{LENGTH} in the transmitted MRW SUFI and the N field in the received MRW_ACK field is not equal to the N_{LENGTH} field in the transmitted MRW SUFI.

Annex A (informative): SDL diagrams

This annex contains the SDL diagrams. For Release'99, it is meant for informative purposes only.

NOTE: All the SDL diagrams presented are [FFS].


Virtual Process Type Acknowledged_link
1_Signals(74)
;
SIGNALSET
Crlc_amconfig_req,
Crlc_Status_ind,
Rlc_AmData_req,
Rlc_AmData_ind,
Rlc_AmData_conf,
Reset_am,
Reset_am_ack,
AmdPduQueuedUp,
StatusPdu,
AmdPdu;
Am
(Am_to_AcknowledgedLink)
(AcknowledgedLink_to_Am)
DtchDcch
(DtchDcch_to_AcknowledgedLink)
(AcknowledgedLink_to_DtchDcch)
Cont
(Cont_to_AcknowledgedLink)
(AcknowledgedLink_to_Cont)

Virtual Process Type Acknowledged_link 1_Declarations(74

;
SIGNALSET

```
DCL

/*SDU, PDU, and PU declarations:________________________________*/

sdu                                     OctetType,
/*The sdu data from the upper layer protocol.*/

amd_pdu, pdu                            AmPdu,
/*A representation of data contained within an AmPdu.*/

amd_pu                                  AmPuStructType,
/*A representation of a local am_pu*/

status_pdu, tx_status_pdu               StatPdu,
/*A representation of data contained within a StatPdu.*/

rst_pdu                                 RstPdu,
/*A representation of data contained within a RstPdu.*/

/*SDU, PDU,  and PU  array declarations:____________________________*/

sdus                                    OctetArrayType,
/*An array containing SDUs.*/

pdus                                    AmPduArrayType,
/*An array containing AMD PDUs created by segmenting a SDU.*/

pus                                     AmPuArrayType,
/*An array containing PUs.*/

rem_pus                                 AmPuArrayType,
/*An array containing PDUs to be removed from queues.*/

status_pdus                             StatusPduArrayType,
/*An array containing several STATUS PDUs.*/


/*Queue declarations:____________________________________*/

receiver_queue                          Queue,
/*A queue used for storing PDUs as they arrive.*/

retransmission_queue                    Queue,
/*A queue used for PDUs that are to be retransmitted.*/

assembly_queue                          Queue,
/*A queue used for reassembly of received PDUs into an SDU.*/

transmitted_queue                       Queue,
/*A queue used for PDUs that have been transmitted.*/

amd_queue                               Queue,
/*A queue used for PDUs to be transmitted.*/

mui_queue                               Queue;
/*A queue used to store mui numbers for which confirmation
  has been requested.*/
```

Virtual Process Type Acknowledged_link 2_Declarations(74

;
SIGNALSET

```
DCL

/*Indicator declarations:______________________________________________*/

epc_active                                     IndicatorType,
/*An indicator used to store whether the Timer_EPC is active or not.*/

poll_periodic_active                           IndicatorType,
/*An indicator used to store whether the Timer_Poll_Periodic is active or not.*/

poll_prohibit_active                           IndicatorType,
/*An indicator used to store whether the Timer_Poll_Prohibit is active or not.*/

rst_active                                     IndicatorType,
/*An indicator used to store whether the Timer_RST is active or not.*/

status_periodic_active                         IndicatorType,
/*An indicator used to store whether the Timer_Status_Periodic is active or not.*/

status_prohibit_active                         IndicatorType,
/*An indicator used to store whether the Timer_Status_Prohibit is active or not.*/

empty                                          IndicatorType,
/*An Indicator used to determine whether a queue is empty or not.*/

exists                                         IndicatorType,
 /*An indicator used to determine whether a particular pdu exists
   within a queue or not.*/

complete                                       IndicatorType,
 /*An indicator used to determine whether an SDU has been
    completely reassembled.*/

cnf                                            IndicatorType,
 /*An indicator used to determine  whether an SDU requires
   confirmation.*/

possible                                       IndicatorType,
/*An indicator used to indicate whether status piggyback is possible or not.
   An indicator used to indicate whether the PUs requested by the status report
  exsists in the que or not.*/

create_status                                  IndicatorType,
 /*An indicator used to store whether a status report should be created or not.*/

poll_triggered                                 IndicatorType,
/*This variable is used to record if a poll is to be transmitted or not.*/

status_triggered                               IndicatorType,
/*This variable is used to indicate whether a status report should be transmitted
   or not.*/

suspend                                        IndicatorType,
/*This variable is used to indicate whether a local_suspend is in progress or not.*/

piggyback                              IndicatorType;
/*This variable indicates whether a piggybacked status report is included
   in the PDU or not.*/
```

Virtual Process Type Acknowledged_link 3_Declarations(74

;
SIGNALSET

```
DCL

/*Indicator declarations:_______________________________________________*/

MRW_active                                   IndicatorType,
/*An indicator used to store whether the Timer_MRW is active or not.*/

poll_active                                  IndicatorType,
/*An indicator used to keep track of whether the Poll_Timer is active or not.*/

contains, mrw_ans                            IndicatorType,
/*These indicators are used when checking the contents of a received
  status Pdu.*/

discard_n_fli                                IndicatorType,
/*This indicator is used to keep track of whether the first N length indicators of a given
  PU should be discarded or not when the receiving window is moved.*/

retrans                                      IndicatorType,
/*This indicator keeps track of whether retransmissions should occur or not.*/

missing_pu_detected                          IndicatorType;
/*This indicator is used to store whether he receive side has detected missing
  PUs.*/
```

Virtual Process Type Acknowledged_link 4_Declarations(74

```
;
SIGNALSET
```

```
DCL

/*Parameter declarations:____________________________________*/

e_r                                  ERParameterType,
/*The parameter indicating the desired end state.*/

poll_triggers                        PollTriggArrType,
/*a configuration parameter dealing with when to issue poll requests.*/

protocol_parameters                  ProtocolParametersStructType,
/*A struct variable containing the protocol parameters set.*/

status_triggers                      StatusTriggArrType,
/*A configuraion parameter dealing with when to issue Status reports.*/

timer_durations                      TimerDurationsStructType,
/*A struct containing the various timer durations.*/

discard                              DiscardArrayType,
 /*A configuration parameter identifying discard conditions.*/

ciphering_mode                       CipheringModeType,
/*The ciphering mode.*/

ciphering_key                        CipheringKeyType,
/*The ciphering key.*/

hfn                                  CipheringSequenceNumberType,
/*The hyper frame number.*/

leng                                 LengthType,
/*The number of SN_MRW fields in the MRW SUFI.*/

pdu_size                             OctetType,
/*The size in octets of an AMD PDU. It is indicated by MAC layer*/

pu_size                              OctetType,
/*The size in octets of a PU.*/

/*Sequence number variables:___________________________________*/

 n,  sn_ack, sq                      SequenceNumberType,
 /*A local sequence number.*/

 poll_window                         SequenceNumberType,
 /*The size of the poll_window.*/

 receive_window                      SequenceNumberType,
 /*The receive window size.*/

 transmit_window                     SequenceNumberType,
 /*The transmit window size.*/

 polled_sn                           SequenceNumberType,
 /*This variable stores a sequence number associated with the PDU that contained
   a poll request.*/

n_susp, sn_suspend                   SequenceNumberType,
/*These variables contains sequence numbers used after  a local suspend has
  been initiated.*/

 sn_mrw                              SequenceNumberType;
 /*This variable stores the sequence number associated with a MRW request.*/
```

Virtual Process Type Acknowledged_link 5_Declarations(74

;
SIGNALSET

```
DCL

/*Local variables declarations:____________________________________*/

logical_channel                          LogicalChannelType,
/*The logical channel associated with transmissions.*/

i, j                                     INTEGER,
/*A local counter.*/

mui                                      MuiType,
 /*The message unit identifier associated with a message to be transmitted.*/

muis                                     MuiArrayType,
 /*An array used to store message unit identifiers.*/

tx_rsn, rx_rsn                           PduIndexType,
 /*A local variable for maintaining knowledge of the latest reset sequence number of
   the transmitted/received RESET PDU.*/

tot_mui, k,  tot_rem, n_sq               PduIndexType,
/*Counters used to manage the amount of PUs and SDUs received.*/

tot_list                                 PduIndexType,
/*A local variable for maintaining knowledge of the total number of
   (SNi, Li)-pairs in a list super field.*/

tot_bitmap, tot_rlist                    PduIndexType,
/*A local variable for maintaining knowledge of the total length of a bitmap or codewords.*/

n_sdu                                    PduIndexType,
/*A local variable for maintaining knowledge of the number of SDUs reassembled PUs.*/

n_pdu                                    PduIndexType,
/*A local variable for maintaining knowledge of the number of AMD PDUs created from a SDU.*/

n_pu                                     PduIndexType,
/*A local variable for maintaining knowledge of the number of PUs included in a AMD PDU.*/

n_status                                 PduIndexType,
/*A local variable for maintaining knowledge of the number of STATUS PDUs
  which have been created.*/

n_pu_per_tti                             PduIndexType,
/*A local variable for maintaining knowledge of the number of PUs received within a TTI.*/

end_state                                EndStateType,
/*A variable used to ensure correct timer reset.*/

poll_win                                 REAL,
/*A local variable used to store the current transmit window usage.*/

 bitmap                                  IndicatorArrayType,
 /*This array of boolean values indicates losses experienced by the
   receiver.*/

codewords                                IndicatorArrayType,
 /*This array is used to store the codewords in the rlsit super field.*/

mrw                                      SufiArrayType;
 /*This array is used to store the MRW super field or the MRW_N_IFL
    super field.*/
```

Virtual Process Type Acknowledged_link 6_Declarations(74

;
SIGNALSET

DCL
/*State variable declarations:____*/

vt_s SequenceNumberType,
/*Send state variable: The sequence number of the next pu to be transmitted for the first time (i.e excluding retransmissions). It is updated after transmission of a PDU which includes not earlier transmitted PUs. The initial value of this variable is 0.*/

vt_a SequenceNumberType,
/*Acknowledge state variable: The sequence number of the next in-sequence PU expected to be acknowledged, thus forming the lower edge of the window of acceptable acknowledgements. The variable vt_a is updated based on receipt of a STATUS PDU including an ACK super-field. The initial value of this variable is 0.*/

vt_ms SequenceNumberType,
/*Maximum send state variable: The sequence number of the first PU not allowed by the peer receiver (i.e. the receiver will allow up t o vt_ms-1) vt_ms=vt_a+ window size. This value represents the upper edge of the transmit window. The transmitter shall not transmit a new PU if vt_s >= vt_ms. The variable vt_ms is updated based on receipt of a STATUS PDU incluiding an ACK and/or WINDOW super-field.*/

vt_pu SequenceNumberType,
/*This state variable is used when the poll every Poll_PU PU function is used. It is incremented with 1 for each PU that is transmitted. It should be incremented for both new and retransmitted PUs. When it reaches Poll_PU a new poll is transmitted and the state variable is set to zero. The initial value of this variable is 0.*/

vt_sdu SequenceNumberType,
/*This state variable is used when the poll every Poll_SDU SDU function is used. It is incremented with 1 for each SDU that is transmitted. When it reaches Poll_SDU a new poll is transmitted and the state variable is set to zero. The poll bit should be set in the PU that contains the last segment of the SDU. The initial value of this variable is 0.*/

vt_rst SequenceNumberType,
/*Reset state variable: This variable is used to count the number of times a RESET PDU is transmit-ted. It is incremented with 1 each time a RESET PDU is transmitted. It is reset upon reception of a RESET ACK PDU. The initial value of this variable is 0.*/

vr_r SequenceNumberType,
/*Receive state variable: The sequence number of the next in sequence PU expected to be received. It is updated upon receipt of the next in-sequence pdu. The initial value of this variable is 0.*/

vr_h SequenceNumberType,
/*Highest expected state variable: The sequence number of the next highest expected pdu. The vari-able is updated whenever a new pdu is received with SN>=vr_h. The initial value of this variable is 0.*/

vr_mr SequenceNumberType,
/*Maximum acceptable receive state variable: The sequence number of the first pdu not allowed by the receiver (i.e. the receiver will allow up to vr_mr-1), vr_mr=vr_r+window size. The receiver shall discard PUs with SN>=vr_mr, (in one case, such a PU may cause the transmission of an unsolicited STATUS PDU).*/

vr_ep SequenceNumberType;
/*Estimated PDU counter state variable: The number of PUs that should be received yet as a consequence of the transmission of the latest STATUS PDU. In acknowledged mode, this state variable is updated at the end of each transmission time interval. It is decremented by the number of PUs that should have been received during the transmission time interval. If VR(EP) is equal to zero, then check if all PUs requested for retransmission in the latest STATUS PDU have been received. */

Virtual Process Type Acknowledged_link 7_Declarations(74

```
;
SIGNALSET
 Crlc_amconfig_req
```

```
DCL
 /*State variable declarations:______________________________________________________*/

 vt_dat                                        SequenceNumberType,
 /*This state variable counts the number of times a PU has been transmitted. There is one
    VT(DAT) for each PU and it is incremented each time the PU is transmitted. The initial
    value of this variable is 0.*/

 vt_mrw                                        SequenceNumberType;
 /*It is used to count the number of times a MRW command is transmitted. VT(MRW) is
   incremented with 1 each time a  MRW command is transmitted. VT(MRW) is reset upon
   the reception of a STATUS PDU which suggests the acknowledgement of a MRW
   command in the receiver or the occurrence of discarding new SDU. The initial value
   of this variable is 0.*/
```

Virtual Process Type Acknowledged_link 8_Declarations(74

;
SIGNALSET

TIMER

Timer_Poll,
/*This timer is only used when the poll timer trigger is used. It is started when the transmitting side sends a poll to the peer entity. The timer is stopped when receiving a STATUS PDU that contains an acknowledgement or negative acknowledgement of the AMD PDU that triggered the timer. The value of the timer is signalled by RRC. If the timer expires and no STATUS PDU containing an acknowledgement or negative acknowledgement of the AMD PDU that triggered the timer has been received, the receiver is polled once more (either by the transmission of a PDU which was not yet sent, or by a retransmission) and the timer is restarted. If there is no PU to be transmitted and all PUs have already been acknowledged, the receiver shall not be polled. If a new poll is sent when the timer is running it is restarted.*/

Timer_Poll_Prohibit,
/*This timer is only used when the poll prohibit function is used. It is used to prohibit transmission of polls within a certain period. A poll shall be delayed until the timer expires if a poll is triggered when the timer is active. Only one poll shall be transmitted when the timer expires even if several polls were triggered when the timer was active. If there is no PU to be transmitted and all PUs have already been acknowledged, a poll shall not be transmitted. This timer will not be stopped by a STATUS PDU. The value of the timer is signalled by RRC. */

Timer_EPC,
/*This timer is only used when the EPC function is used and it accounts for the roundtrip delay, i.e. the time when the first retransmitted PU should be received after a STATUS has been sent. The timer is started when a STATUS report is transmitted and when it expires EPC can start decrease. The value of the timer is signalled by RRC.*/

Timer_EPC_check,
/*This timer is used to count down the state variable vr_ep at acertain interval.*/

Timer_Discard(MuiType),
/*This timer is used for the SDU discard function. In the transmitter, the timer is activated upon reception of a SDU from higher layer. If the SDU has not been acknowledged when the timer expires, the SDU is discarded. Following which, if the SDU discard function uses explicit signalling, a Move Receiving Window request is sent to the receiver. The value of the timer is signalled by RRC.*/

Timer_Poll_Periodic;
/*This timer is only used when the timer based polling is used. The timer is started when the RLC entity is created. Each time the timer expires a poll is transmitted and the timer is restarted. If there is no PU to be transmitted and all PUs have already been acknowledged, a poll shall not be transmitted and the timer shall only be restarted. The value of the timer is signalled by RRC.*/

Virtual Process Type Acknowledged_link 9_Declarations(74

;
SIGNALSET

TIMER

Timer_Status_Prohibit,
/*This timer is only used when the STATUS PDU prohibit function is used. It prohibits the receiving side from sending STATUS PDUs. The timer is started when a STATUS PDU is transmitted and no new STATUS PDU can be transmitted before the timer has expired. The value of the timer is signalled by RRC.*/

Timer_Status_Periodic,
/*This timer is only used when timer based STATUS PDU sending is used. The timer is started when the RLC entity is created. Each time the timer expires a STATUS PDU is transmitted and the timer is restarted. The value of the timer is signalled by RRC.*/

Timer_MRW,
/*This timer is used as part of the Move Receiving Window protocol. It is used to trigger the retransmission of a STATUS PDU containing an MRW SUFI field. The timer is started when the STATUS PDU is first transmitted. Each time the timer expires the STATUS PDU is retransmitted and the timer is restarted. It shall be stopped when a STATUS PDU is received that indicates that VR(R) ^{3} SN_MRW. It shall also be stopped if a new MRW procedure is triggered whilst it is running.*/

Timer_RST;
/*It is used to detect the loss of RESET ACK PDU from the peer RLC entity. This timer is set when the RESET PDU is transmitted. And it will be stopped upon reception of RESET ACK PDU. If it expires, RESET PDU will be retransmitted.*/

Virtual Process Type Acknowledged_link 1_LocalProcedures(74

;
SIGNALSET

Sdu_am_segmentation

This procedure manages segmentation and concatenation of sdus. If the poll_trigger EVERY_POLL_SDU is used, poll bit is set in accordance with the value POLL_SDU. In case a SDU is smaller than a PU and waiting next SDU, n_pdu=0 is returned.

FPAR

IN/OUT	sdu	OctetType,
IN	cfn	IndicatorType,
IN/OUT	np	SequenceNumberType,
IN/OUT	pdus	AmPduArrayType,
IN/OUT	qu	Queue,
IN	poll_trigg	PollTriggArrType,
IN	prtcl_parmeter	ProtocolParameterStructType,
IN/OUT	vt_sdu	SequenceNumberType,
IN	cip_m	CipheringModeType,
IN	cip_k	CipheringKeyType,
IN	cip_s	CipheringSequenceNumberType,
IN/OUT	mui	MuiType,
IN	pdu_s	OctetType,
IN	pu_s	OctetType;

Set_sequence_number

This procedure sets the sequence numbers within an AmPdu.

FPAR

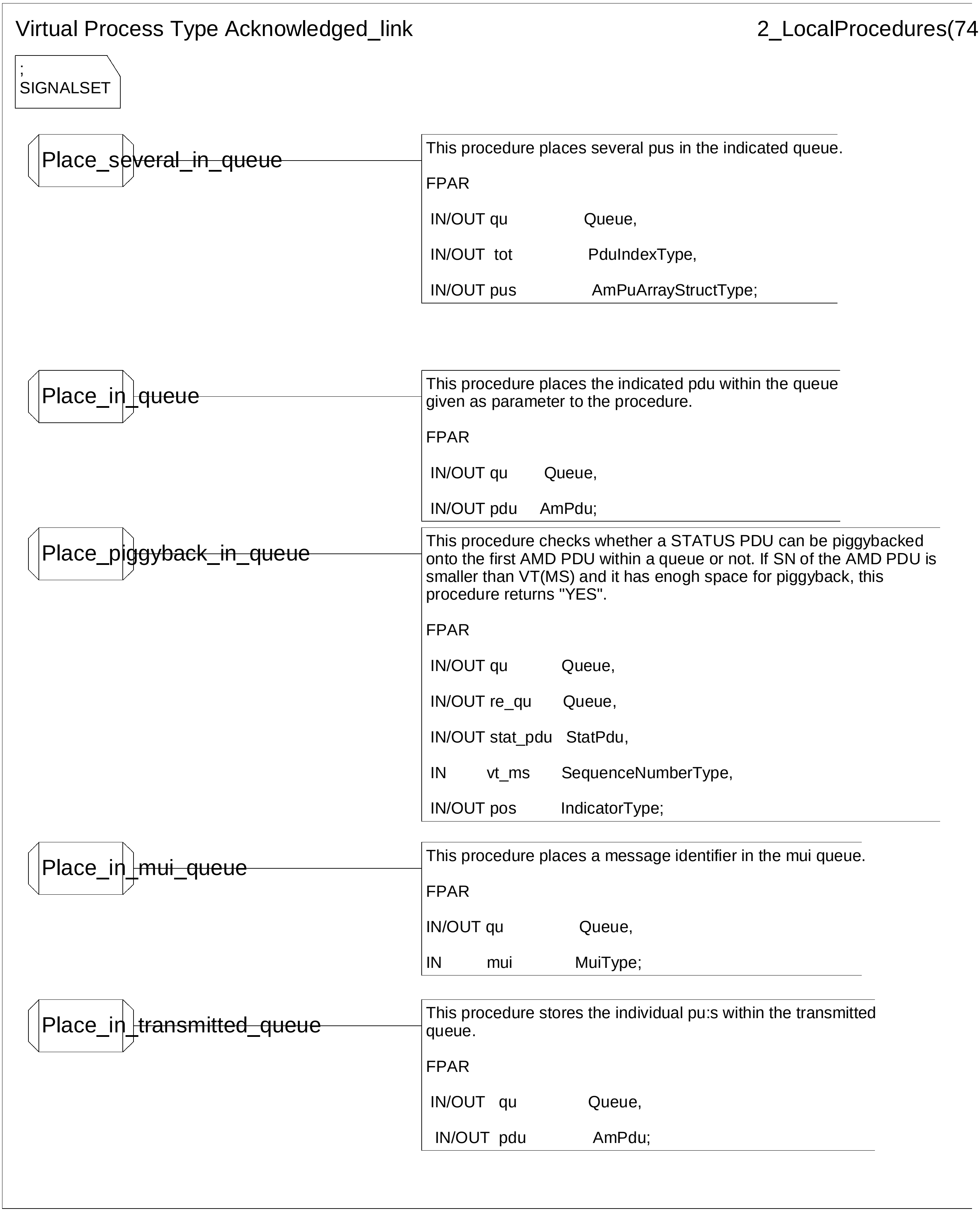
IN/OUT	pdu	AmPdu,
IN	vt_s	SequenceNumberType;

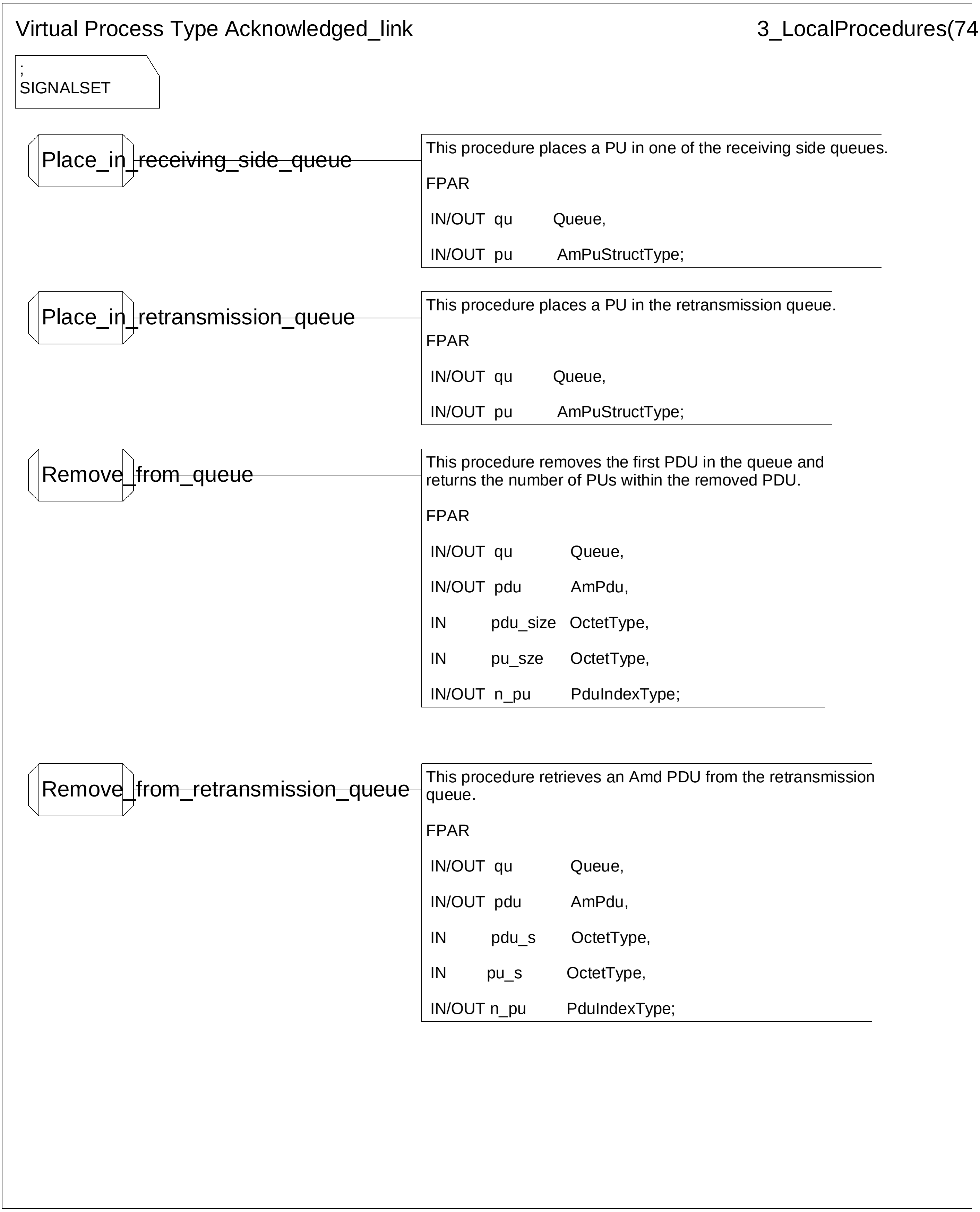
Read_pdu

This procedure retrieves a copy of the first entry in the queue indicated as parameter to the procedure.

FPAR

IN/OUT	qu	Queue,
IN/OUT	am_pdu	AmPdu;


Virtual Process Type Acknowledged_link
2_LocalProcedures(74
;
SIGNALSET
Place_several_in_queue
This procedure places several pus in the indicated queue.
FPAR
IN/OUT qu Queue,
IN/OUT tot PduIndexType,
IN/OUT pus AmPuArrayStructType;
Place_in_queue
This procedure places the indicated pdu within the queue given as parameter to the procedure.
FPAR
IN/OUT qu Queue,
IN/OUT pdu AmPdu;
Place_piggyback_in_queue
This procedure checks whether a STATUS PDU can be piggybacked onto the first AMD PDU within a queue or not. If SN of the AMD PDU is smaller than VT(MS) and it has enogh space for piggyback, this procedure returns "YES".
FPAR
IN/OUT qu Queue,
IN/OUT re_qu Queue,
IN/OUT stat_pdu StatPdu,
IN vt_ms SequenceNumberType,
IN/OUT pos IndicatorType;
Place_in_mui_queue
This procedure places a message identifier in the mui queue.
FPAR
IN/OUT qu Queue,
IN mui MuiType;
Place_in_transmitted_queue
This procedure stores the individual pu:s within the transmitted queue.
FPAR
IN/OUT qu Queue,
IN/OUT pdu AmPdu;


Virtual Process Type Acknowledged_link
3_LocalProcedures(74
;
SIGNALSET
Place_in_receiving_side_queue
This procedure places a PU in one of the receiving side queues.
FPAR
IN/OUT qu Queue,
IN/OUT pu AmPuStructType;
Place_in_retransmission_queue
This procedure places a PU in the retransmission queue.
FPAR
IN/OUT qu Queue,
IN/OUT pu AmPuStructType;
Remove_from_queue
This procedure removes the first PDU in the queue and returns the number of PUs within the removed PDU.
FPAR
IN/OUT qu Queue,
IN/OUT pdu AmPdu,
IN pdu_size OctetType,
IN pu_sze OctetType,
IN/OUT n_pu PduIndexType;
Remove_from_retransmission_queue
This procedure retrieves an Amd PDU from the retransmission queue.
FPAR
IN/OUT qu Queue,
IN/OUT pdu AmPdu,
IN pdu_s OctetType,
IN pu_s OctetType,
IN/OUT n_pu PduIndexType;

Virtual Process Type Acknowledged_link

4_LocalProcedures(74

;
SIGNALSET

Remove_identified_from_queue

This procedure removes a pu with a given sequence number from the queue identified.

FPAR

IN/OUT	qu	Queue,
IN	sn	SequenceNumberType,
IN/OUT	pu	AmPuStructType;

Remove_identified_from_mui_queue

This procedure removes a specific mui from the mui queue used to keep track of Timer_Discard instances.

FPAR

IN/OUT	sdu_queue	Queue,
IN	mui	MuiType;

Remove_list_from_queue

This procedure checks whether each sequence number of missing PU informed by LIST SUFI is within the value between vt_a and vt_s, and removes a list of pdus indicated by sequence numbersfrom the transmitted queue and retransmission_queue.

FPAR

IN/OUT	qu	Queue,
IN/OUT	re_qu	Queue,
IN	sq	SequenceNumberType,
IN/OUT	no	PduIndexType,
IN/OUT	tot	PduIndexType,
IN/OUT	pus	AmPuArrayStructType,
IN/OUT	pos	Indicator TYpe;

Virtual Process Type Acknowledged_link 5_LocalProcedures(74

;
SIGNALSET

Remove_bitmap_from_queue

This procedure checks whether each sequence number of missing PU informed by BITMAP SUFI is within the value between vt_a and vt_s, and removes a list of pdus in accordance with a bitmap from the transmitted queue and retranmission queue.

FPAR

IN/OUT qu Queue,

IN/OUT re_qu Queue,

IN sq SequenceNumberType,

IN/OUT no PduIndexType,

IN/OUT bmap IndicatorArrayType,

IN/OUT tot PduIndexType,

IN/OUT pus AmPuArrayStructType,

IN/OUT pos Indicator TYpe;

Remove_rlist_from_queue

This procedure checks whether each sequence number of missing PU informed by RLIST SUFI is within the value between vt_a and vt_s, and removes a list of pdus in accordance with a codewords from the transmissitted queue and retranmission queue.

FPAR

IN/OUT qu Queue,

IN/OUT re_qu Queue,

IN sq SequenceNumberType,

IN/OUT no PduIndexType,

IN/OUT cw IndicatorArrayType,

IN/OUT tot PduIndexType,

IN/OUT pus AmPuArrayType,

IN/OUT poss IndicatorType,

IN/OUT pos Indicator TYpe;

Virtual Process Type Acknowledged_link

6_LocalProcedures(74

;
SIGNALSET

Remove_mui_from_queue

This procedure removes all PUs associated with a given mui from the transmitted_queue.

FPAR

IN/OUT mui MuiType,

IN/OUT tx_qu Queue,

IN/OUT retx_qu Queue;

Remove_all_below_mrw_from_queue

This procedure removes all PUs below the move receiving window from all receiver queues.

FPAR

IN remove IndicatorType,

IN/OUT r_qu Queue,

IN/OUT a_qu Queue,

IN/OUT mrw SufiArrayType;

Remove_acks_and_get_muis

This procedure removes all pus that have been acknowledged from the indicated queue and stores the muis that are removed from the queue in a special array.

FPAR

IN/OUT tx_qu Queue,

IN re_qu Queue,

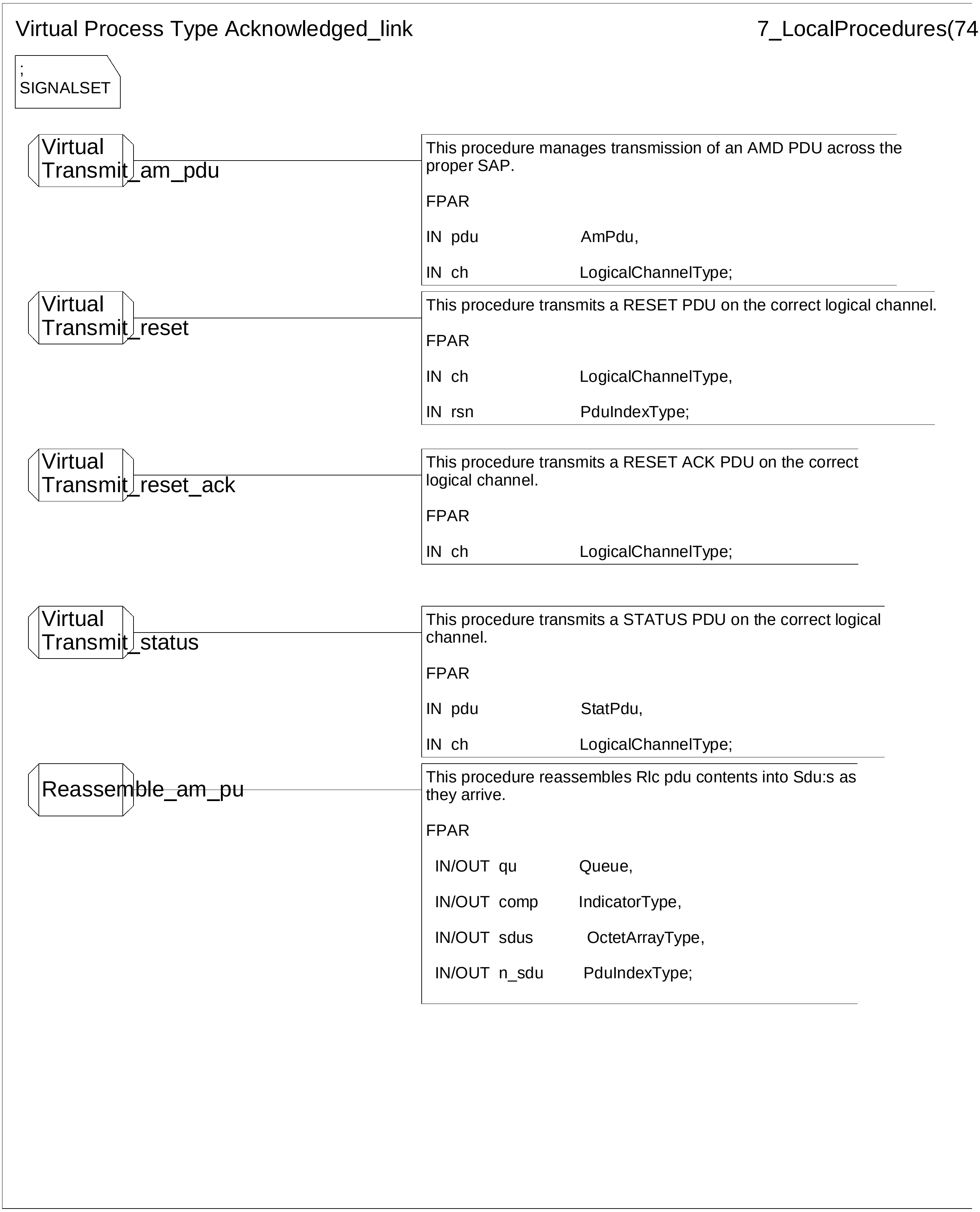
IN sn SequenceNumberType,

IN/OUT tot PduIndexType,

IN/OUT muis MuiArrayType,

IN/OUT poll_tot PduIndexType,

IN/OUT rem_poll SequenceNumberArrayType;


Virtual Process Type Acknowledged_link
7_LocalProcedures(74
;
SIGNALSET
Virtual Transmit_am_pdu
This procedure manages transmission of an AMD PDU across the proper SAP.
FPAR
IN pdu AmPdu,
IN ch LogicalChannelType;
Virtual Transmit_reset
This procedure transmits a RESET PDU on the correct logical channel.
FPAR
IN ch LogicalChannelType,
IN rsn PduIndexType;
Virtual Transmit_reset_ack
This procedure transmits a RESET ACK PDU on the correct logical channel.
FPAR
IN ch LogicalChannelType;
Virtual Transmit_status
This procedure transmits a STATUS PDU on the correct logical channel.
FPAR
IN pdu StatPdu,
IN ch LogicalChannelType;
Reassemble_am_pu
This procedure reassembles Rlc pdu contents into Sdu:s as they arrive.
FPAR
IN/OUT qu Queue,
IN/OUT comp IndicatorType,
IN/OUT sdus OctetArrayType,
IN/OUT n_sdu PduIndexType;

Virtual Process Type Acknowledged_link 8_LocalProcedures(74

;
SIGNALSET

Extract_status_from_pdu

This procedure extracts piggybacked status information from the received PDU.

FPAR

IN/OUT pdu AmPdu,
IN/OUT st_pdu StatPdu;

Extract_pus

This procedure places the pus in the received AMD PDU in an array in order to make them available for processing one by one and checks the number of PUs in the AMD PDU.

FPAR
IN/OUT pdu AmPdu,
IN/OUT pus AmPuArrayType,
IN/OUT n_pu PduIndexType;

Initialise_state_variables

This procedure sets the state variables appropriately.

FPAR
IN/OUT vt_s, vt_ms, vt_sdu, vt_pu, vt_a,
vr_r, vr_h, vr_mr SequenceNumberType;

Initialise_vtDAT

This procedure initialises the retransmission counters associated with the PUs within the PDU.

FPAR

IN/OUT pdu AmPdu;

Increment_vtDAT

This procedure increments the retransmission counters associated with the PUs within the PDU.

FPAR

IN/OUT pdu AmPdu;

Queue_initialisations

This procedure initialises all queues needed within the process.

FPAR

IN/OUT a_qu, t_qu, retx_qu, rx_qu,
as_qu, sdu_qu Queue;

Virtual Process Type Acknowledged_link 9_LocalProcedures(74

;
SIGNALSET

Create_status

This procedure creates a status report based on available information. The information can be split into several STATUS PDUs if it can not be mapped onto one STATUS PDU. At the same time, vr_ep is set equal to the number of requested PUs.

FPAR

IN	vr_r	SequenceNumberType,
IN	vr_h	SequenceNumberType,
IN	rx_win	SequenceNumberType,
IN	pdu_size	OctetType,
IN	rx_qu	Queue,
IN/OUT	stat_pdus	StatusPduArrayType,
IN/OUT	vr_ep	SequenceNumberType,
IN/OUT	n_stat	PduIndexType,
IN	sn_mrw	SequenceNumberType;

Exists_in_receiver_queue

This procedure checks if an identified pu exists within the receiver queue.

FPAR

IN	n	SequenceNumberType,
IN/OUT	qu	Queue,
IN/OUT	exists	IndicatorType;

Estimate_number_of_pus

This procedure estimates the number of PUs that have been received within aTTI.

FPAR

IN/OUT	n_pu_tti	PduIndexType;

Get_sn_mrw

This procedure sets the value of sn_mrw according to the queue status.

FPAR

IN/OUT	sn_mrw	SequenceNumberType,
IN	am_qu	Queue,
IN	tx_qu	Queue,
IN	retx_qu	Queue;

Virtual Process Type Acknowledged_link 10_LocalProcedures(74

;
SIGNALSET

Check_status_creation

This procedure checks if a status report should be generated.

FPAR

IN vr_r SequenceNumberType,

IN vr_h SequenceNumberType,

IN qu Queue,

IN/OUT status IndicatorType;

Check_if_queue_empty

This procedure checks if there are any PDUs remaining in the queue given as parameter to the procedure.

FPAR

IN qu Queue,

IN/OUT empty IndicatorType;

Check_and_delete_timer_discards

This procedure checks if any timer polls are active and returns the first message identifier associated with the discard. If the queue is empty, empty=YES is returned.

FPAR

IN/OUT qu Queue,

IN mui MuiType,

IN/OUT empty IndicatorType;

Check_if_piggyback

This procedure checks if the current AMD PDU to be transmitted contains a piggybacked STATUS PDU or not

FPAR
IN pdu AmPdu,
IN/OUT piggyback IndicatorType;

Check_if_MRW_answer

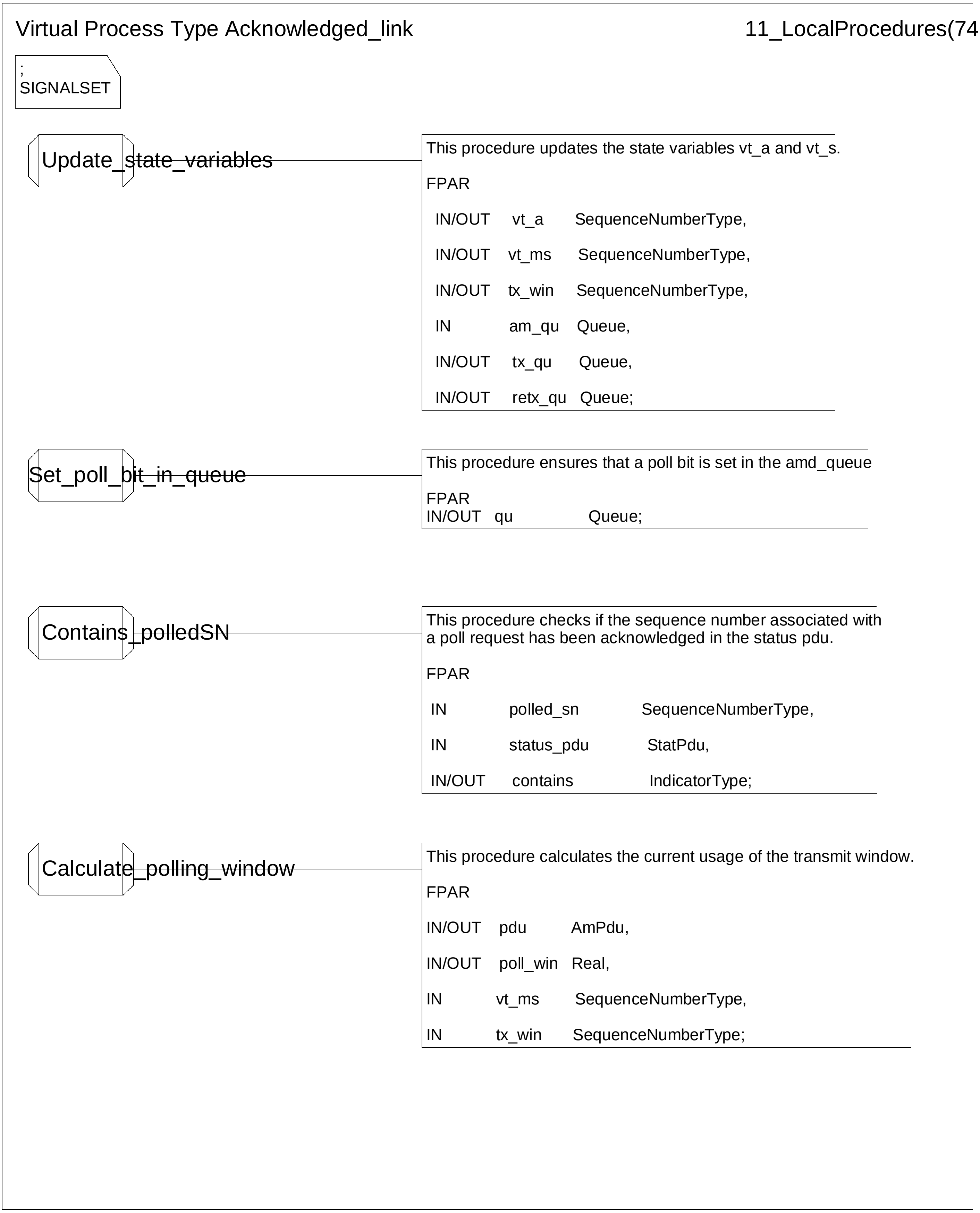
This procedure checks if the peer has responded to a MRW command.

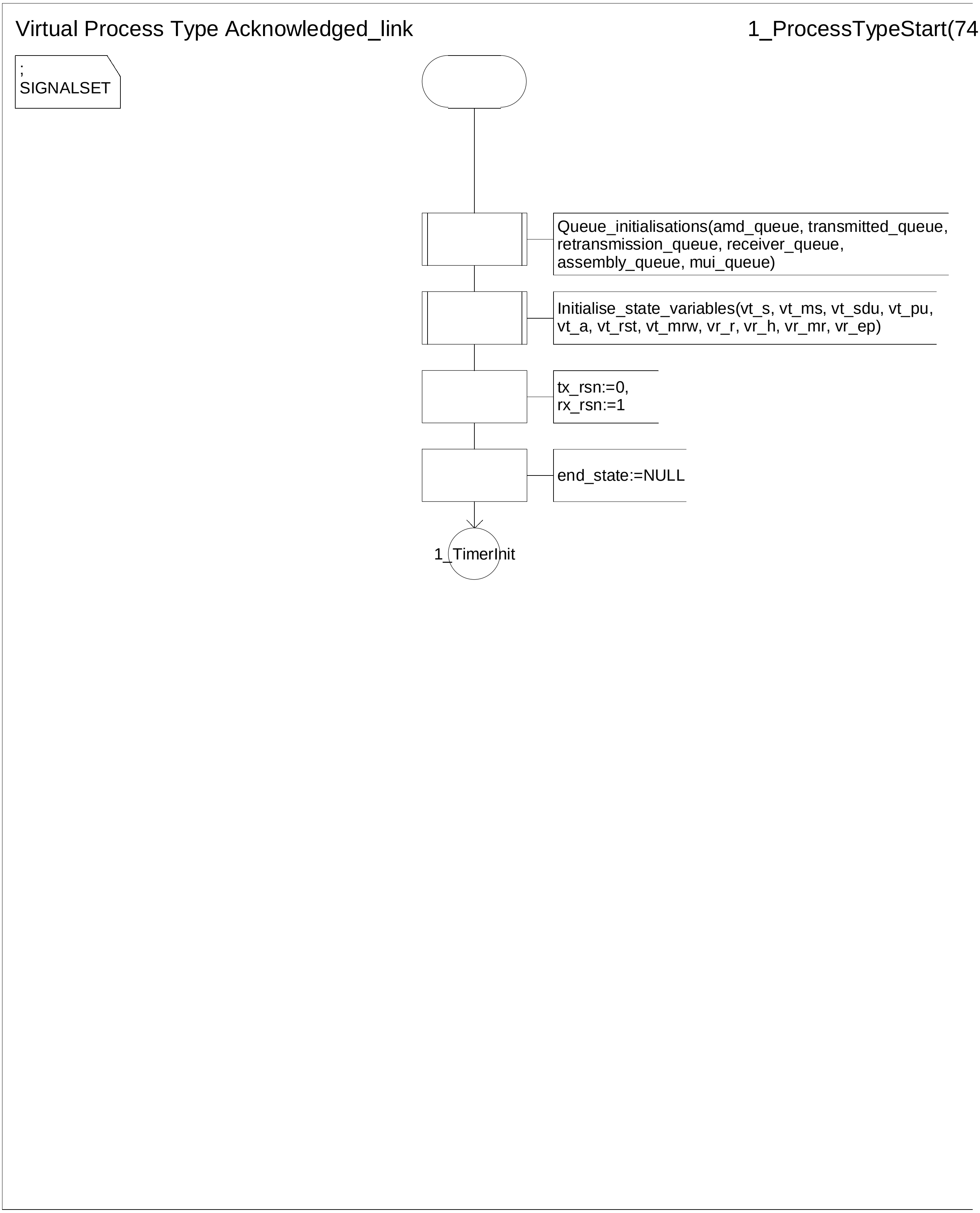
FPAR

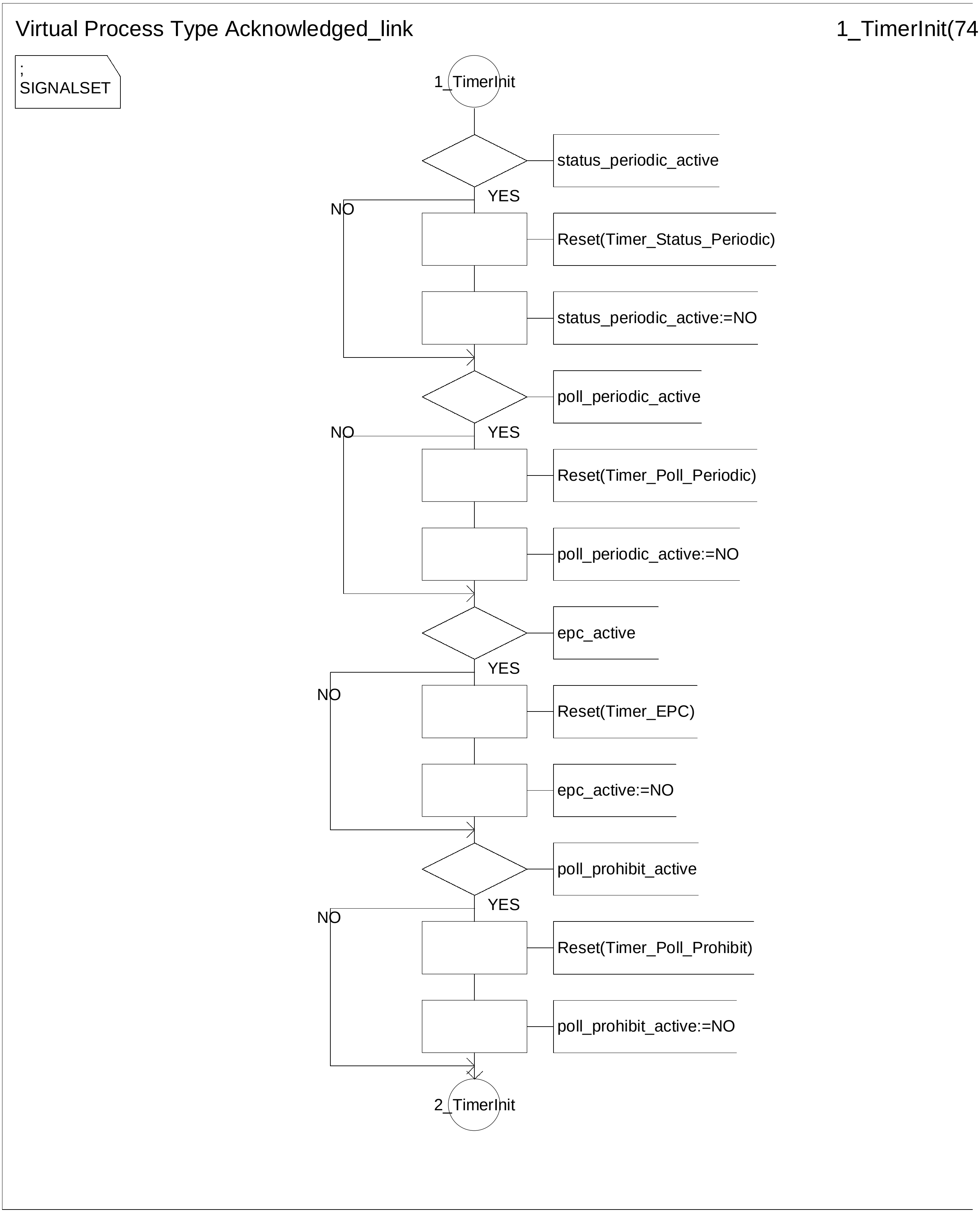
IN sn_mrw SequenceNumberType,

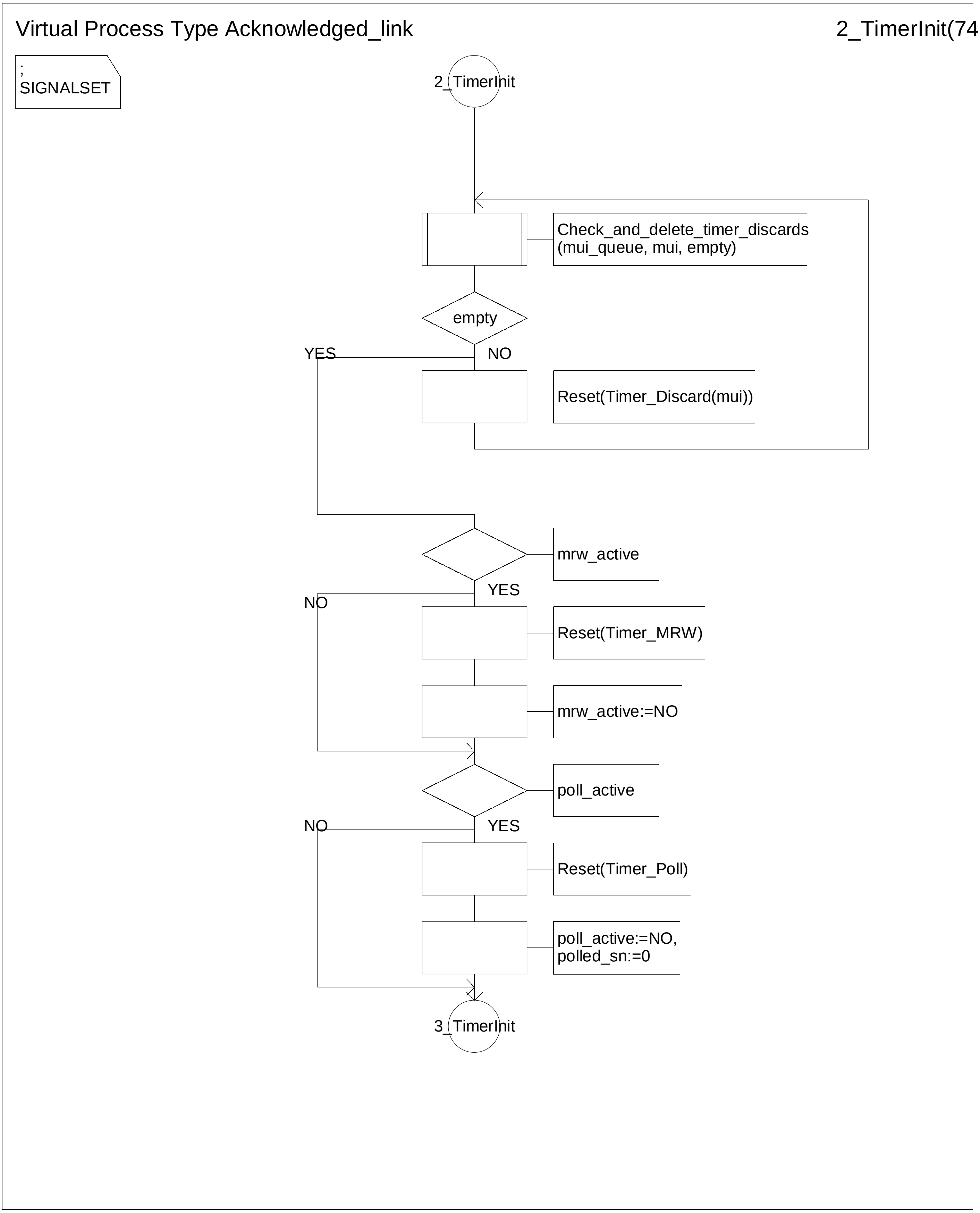
IN status_pdu StatPdu,

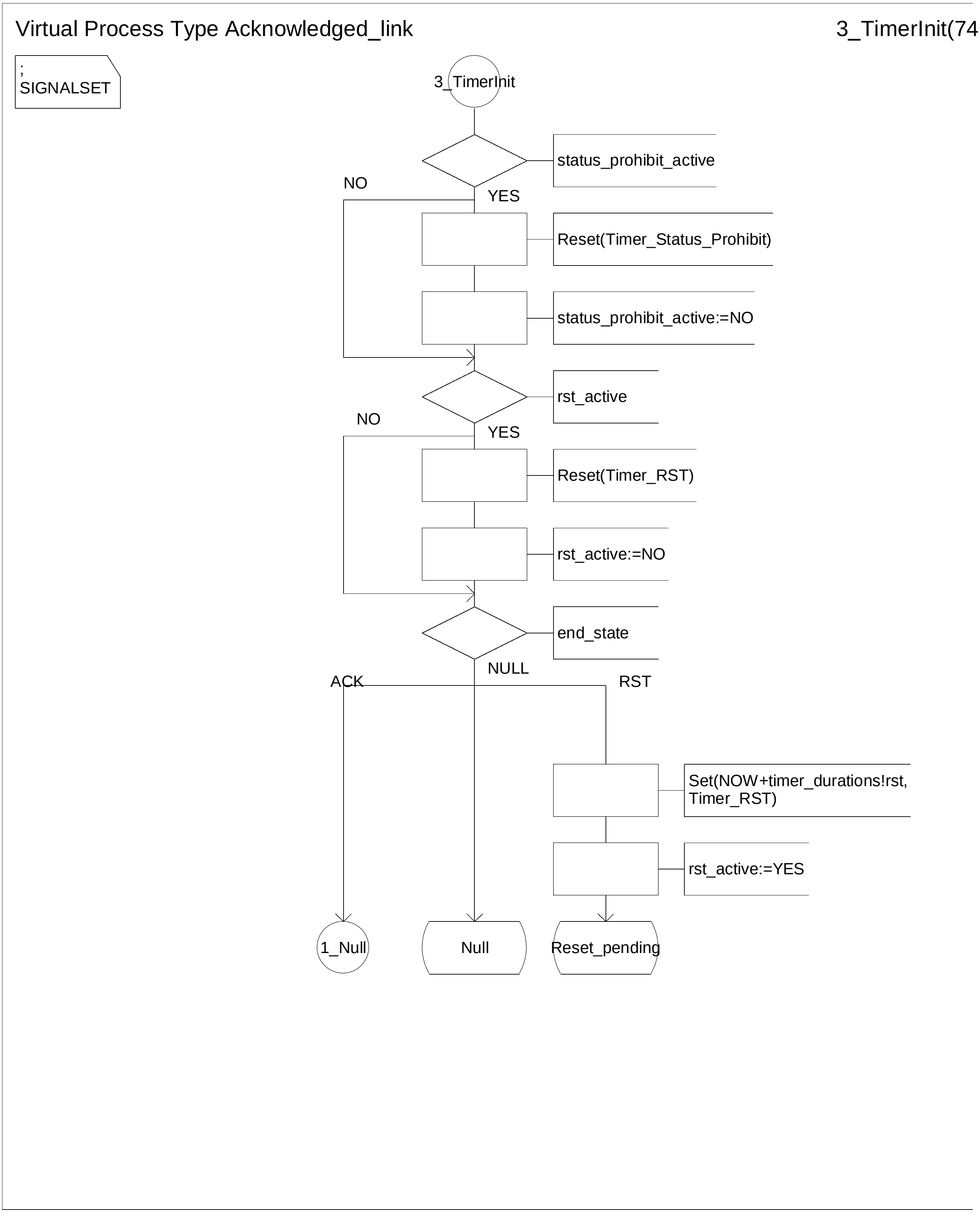
IN/OUT mrw_ans IndicatorType;

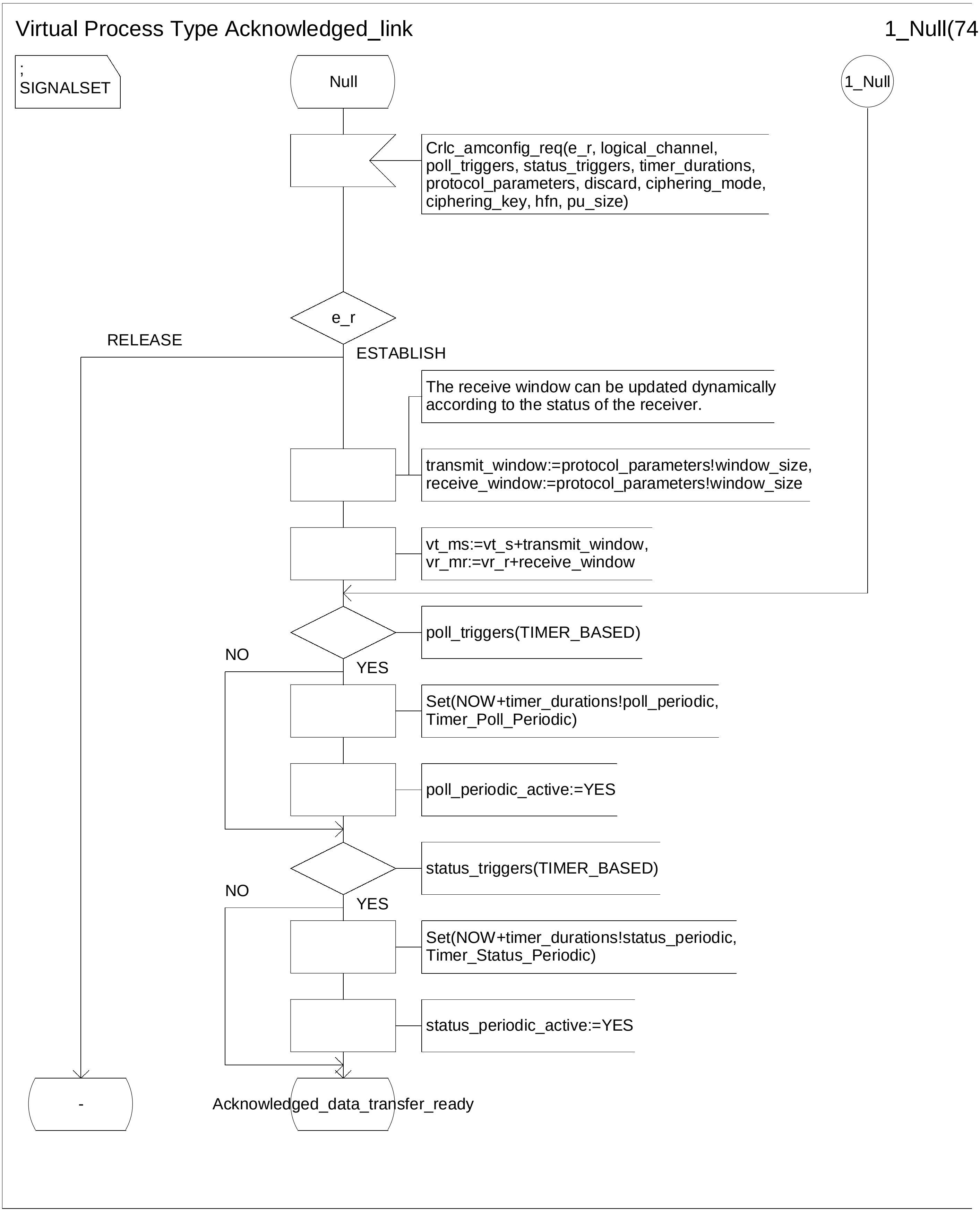

Virtual Process Type Acknowledged_link
11_LocalProcedures(74
;
SIGNALSET
Update_state_variables
This procedure updates the state variables vt_a and vt_s.
FPAR
IN/OUT vt_a SequenceNumberType,
IN/OUT vt_ms SequenceNumberType,
IN/OUT tx_win SequenceNumberType,
IN am_qu Queue,
IN/OUT tx_qu Queue,
IN/OUT retx_qu Queue;
Set_poll_bit_in_queue
This procedure ensures that a poll bit is set in the amd_queue
FPAR
IN/OUT qu Queue;
Contains_polledSN
This procedure checks if the sequence number associated with a poll request has been acknowledged in the status pdu.
FPAR
IN polled_sn SequenceNumberType,
IN status_pdu StatPdu,
IN/OUT contains IndicatorType;
Calculate_polling_window
This procedure calculates the current usage of the transmit window.
FPAR
IN/OUT pdu AmPdu,
IN/OUT poll_win Real,
IN vt_ms SequenceNumberType,
IN tx_win SequenceNumberType;

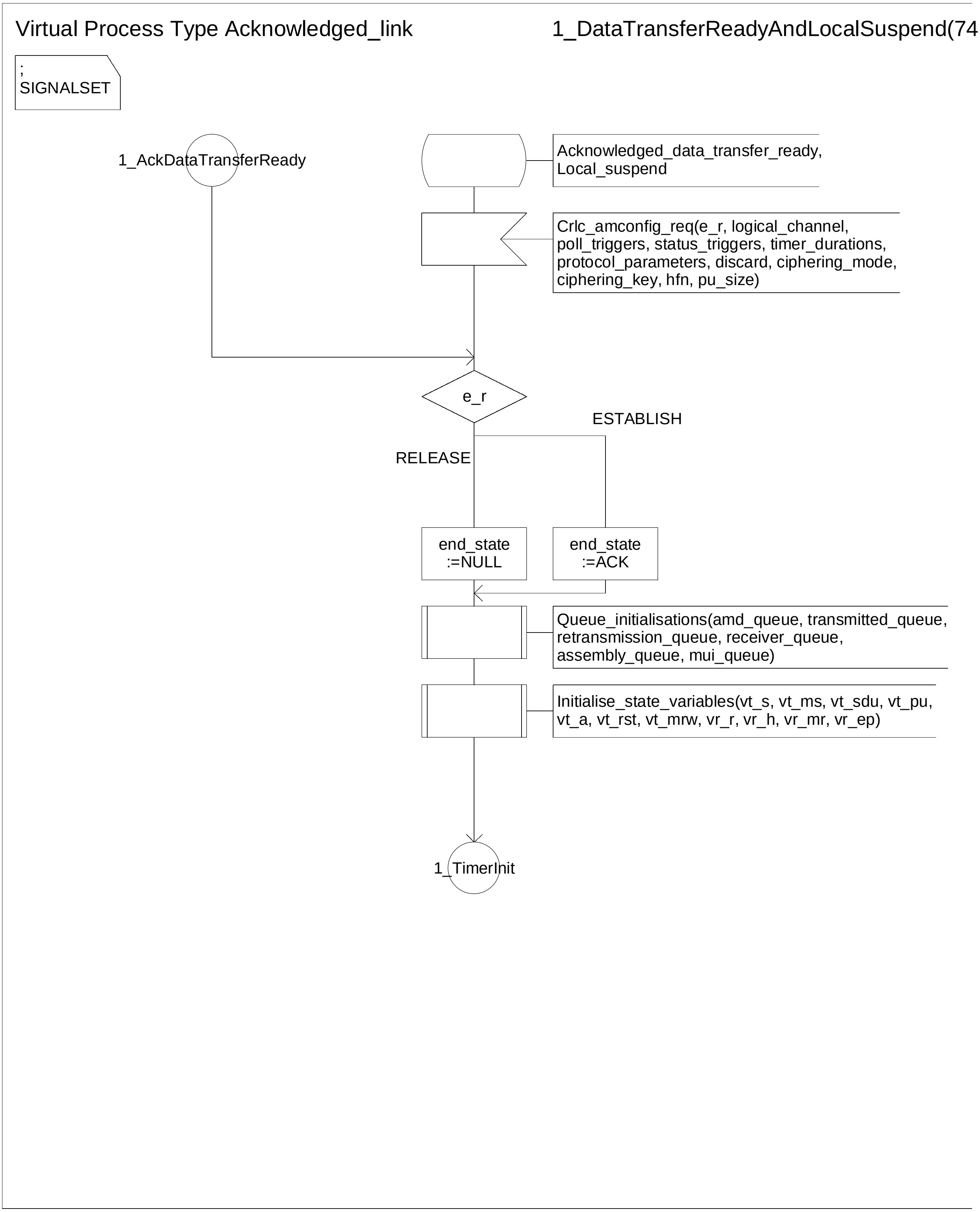

Virtual Process Type Acknowledged_link
1_ProcessTypeStart(74
;
SIGNALSET
Queue_initialisations(amd_queue, transmitted_queue, retransmission_queue, receiver_queue, assembly_queue, mui_queue)
Initialise_state_variables(vt_s, vt_ms, vt_sdu, vt_pu, vt_a, vt_rst, vt_mrw, vr_r, vr_h, vr_mr, vr_ep)
tx_rsn:=0,
rx_rsn:=1
end_state:=NULL
1_TimerInit

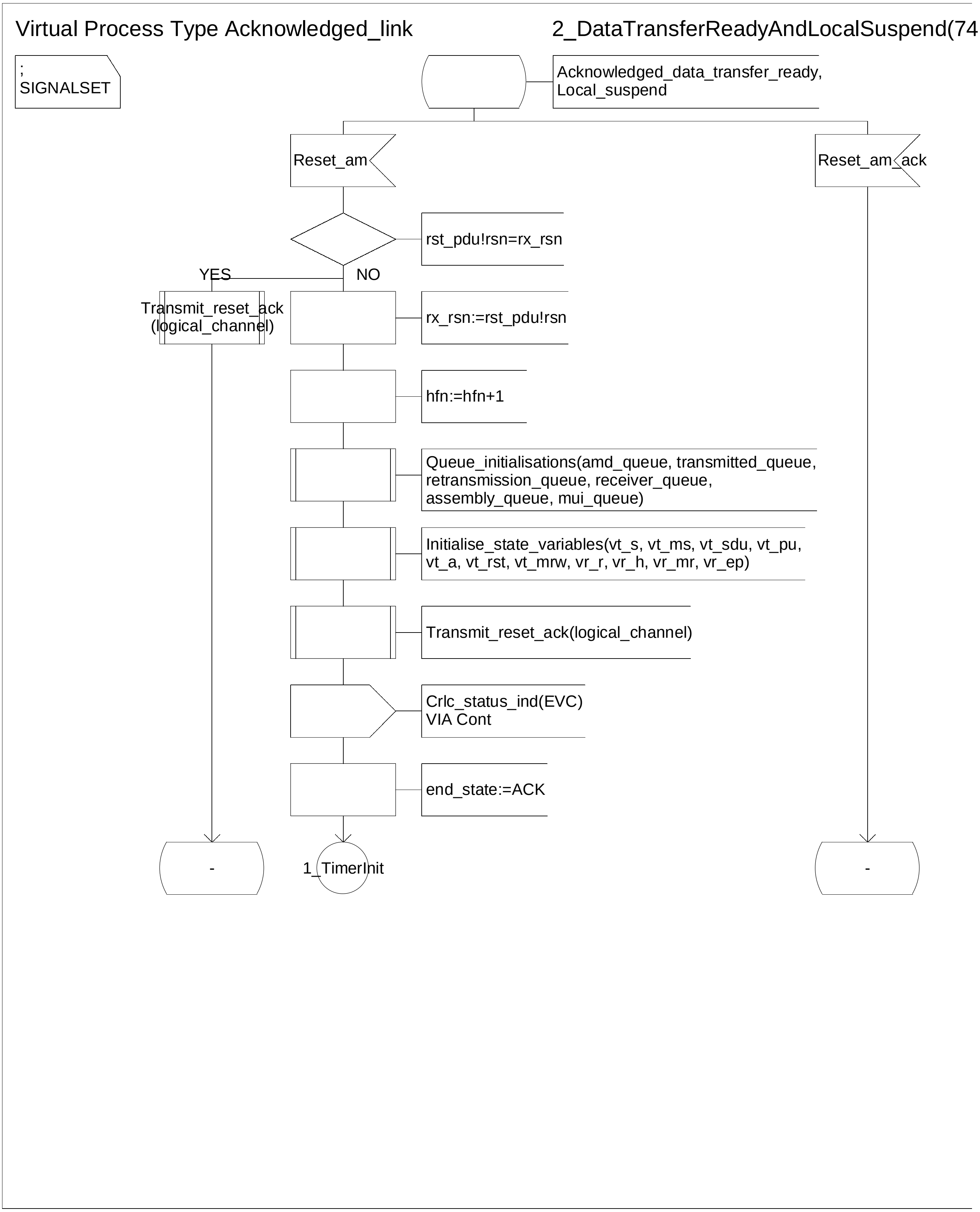

Virtual Process Type Acknowledged_link
1_TimerInit(74
;
SIGNALSET
1_TimerInit
status_periodic_active
YES
NO
Reset(Timer_Status_Periodic)
status_periodic_active:=NO
poll_periodic_active
YES
NO
Reset(Timer_Poll_Periodic)
poll_periodic_active:=NO
epc_active
YES
NO
Reset(Timer_EPC)
epc_active:=NO
poll_prohibit_active
YES
NO
Reset(Timer_Poll_Prohibit)
poll_prohibit_active:=NO
2_TimerInit

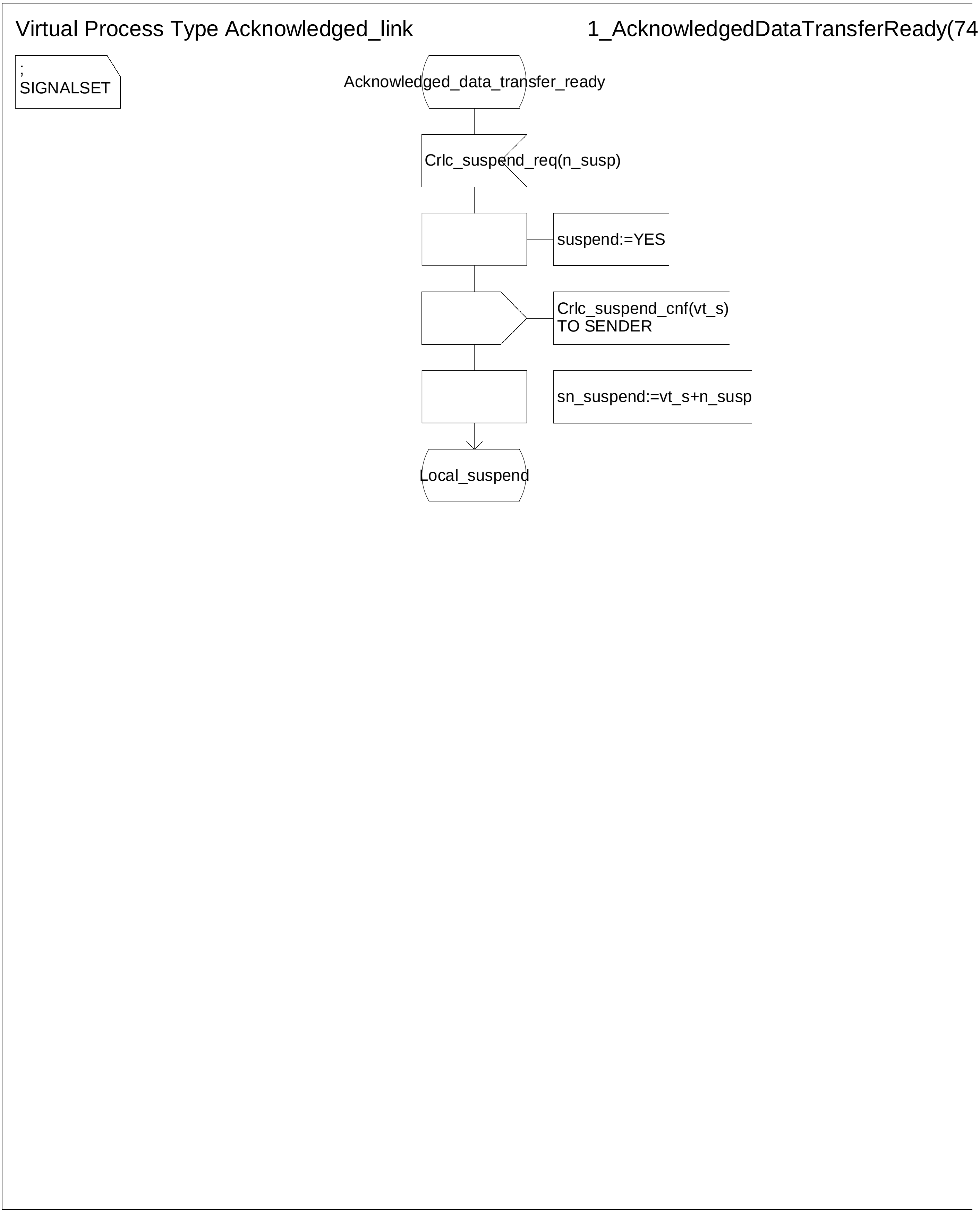

Virtual Process Type Acknowledged_link
2_TimerInit(74
;
SIGNALSET
2_TimerInit
Check_and_delete_timer_discards
(mui_queue, mui, empty)
empty
YES
NO
Reset(Timer_Discard(mui))
mrw_active
YES
NO
Reset(Timer_MRW)
mrw_active:=NO
poll_active
NO
YES
Reset(Timer_Poll)
poll_active:=NO,
polled_sn:=0
3_TimerInit


Virtual Process Type Acknowledged_link
3_TimerInit(74
;
SIGNALSET
3_TimerInit
status_prohibit_active
NO
YES
Reset(Timer_Status_Prohibit)
status_prohibit_active:=NO
rst_active
NO
YES
Reset(Timer_RST)
rst_active:=NO
end_state
NULL
ACK
RST
Set(NOW+timer_durations!rst,
Timer_RST)
rst_active:=YES
1_Null
Null
Reset_pending

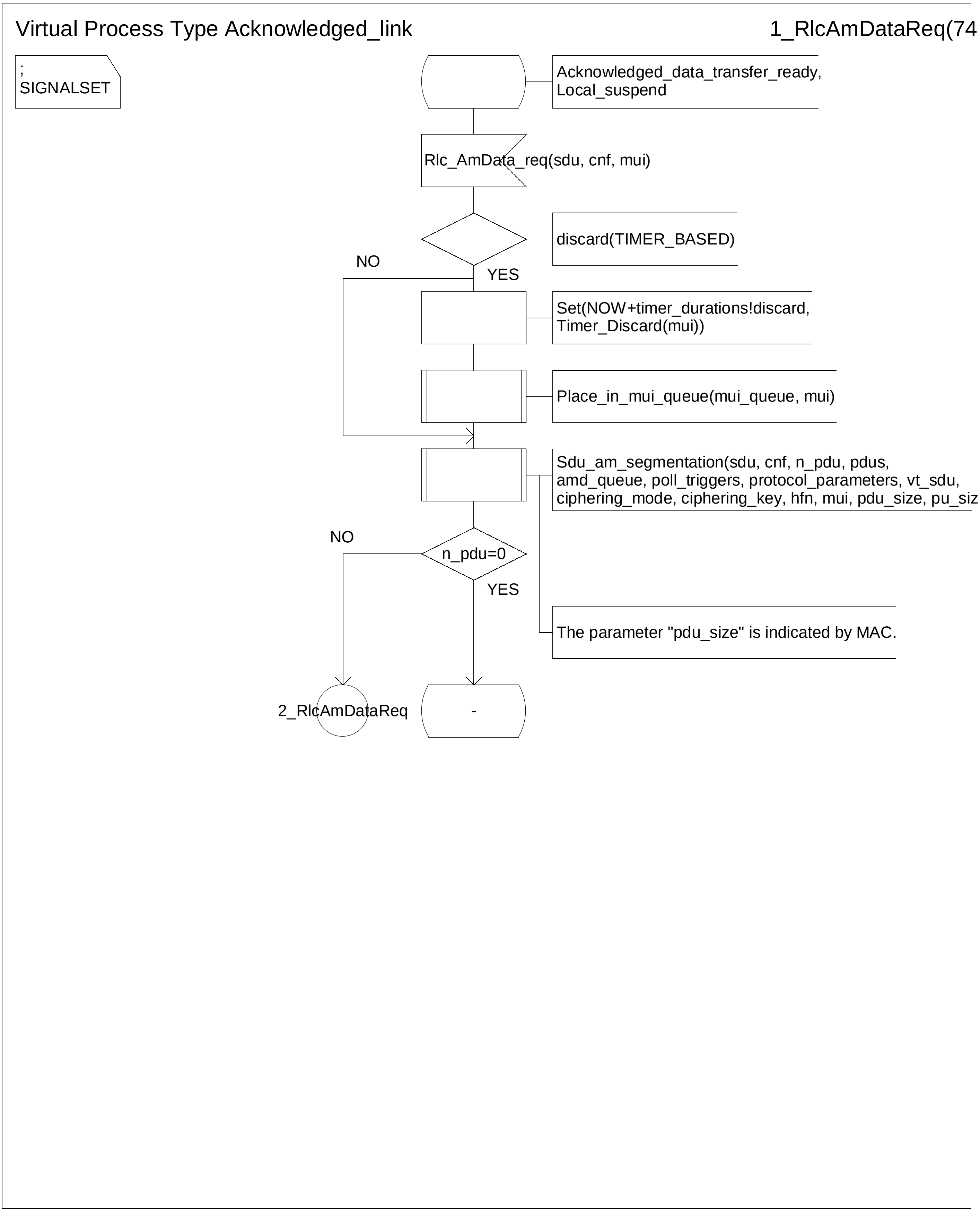

Virtual Process Type Acknowledged_link
1_Null(74
;
SIGNALSET
Null
1_Null
Crlc_amconfig_req(e_r, logical_channel, poll_triggers, status_triggers, timer_durations, protocol_parameters, discard, ciphering_mode, ciphering_key, hfn, pu_size)
e_r
RELEASE
ESTABLISH
The receive window can be updated dynamically according to the status of the receiver.
transmit_window:=protocol_parameters!window_size, receive_window:=protocol_parameters!window_size
vt_ms:=vt_s+transmit_window, vr_mr:=vr_r+receive_window
poll_triggers(TIMER_BASED)
NO
YES
Set(NOW+timer_durations!poll_periodic, Timer_Poll_Periodic)
poll_periodic_active:=YES
status_triggers(TIMER_BASED)
NO
YES
Set(NOW+timer_durations!status_periodic, Timer_Status_Periodic)
status_periodic_active:=YES
-
Acknowledged_data_transfer_ready

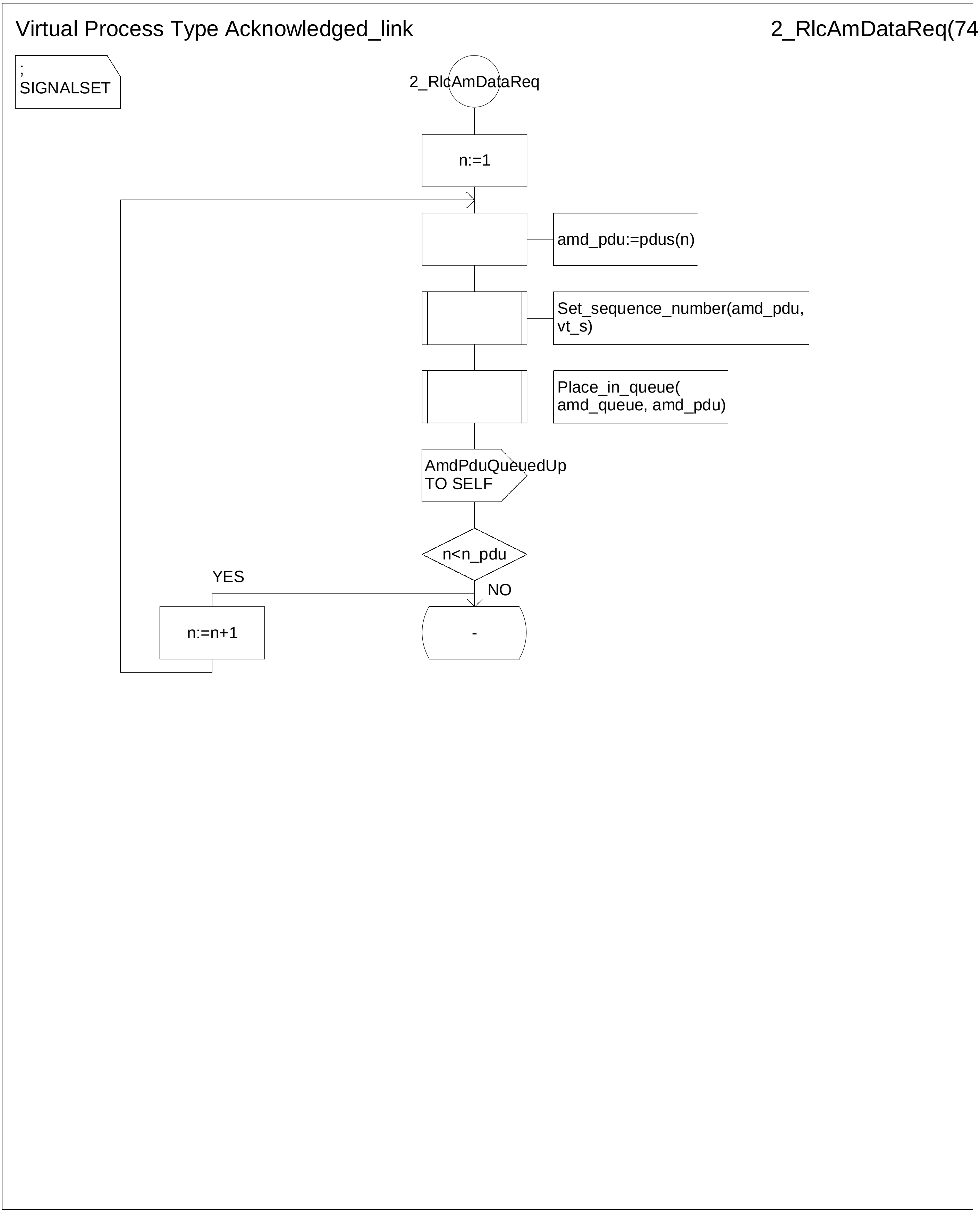

Virtual Process Type Acknowledged_link
1_DataTransferReadyAndLocalSuspend(74
;
SIGNALSET
1_AckDataTransferReady
Acknowledged_data_transfer_ready,
Local_suspend
Crlc_amconfig_req(e_r, logical_channel,
poll_triggers, status_triggers, timer_durations,
protocol_parameters, discard, ciphering_mode,
ciphering_key, hfn, pu_size)
e_r
ESTABLISH
RELEASE
end_state
:=NULL
end_state
:=ACK
Queue_initialisations(amd_queue, transmitted_queue,
retransmission_queue, receiver_queue,
assembly_queue, mui_queue)
Initialise_state_variables(vt_s, vt_ms, vt_sdu, vt_pu,
vt_a, vt_rst, vt_mrw, vr_r, vr_h, vr_mr, vr_ep)
1_TimerInit

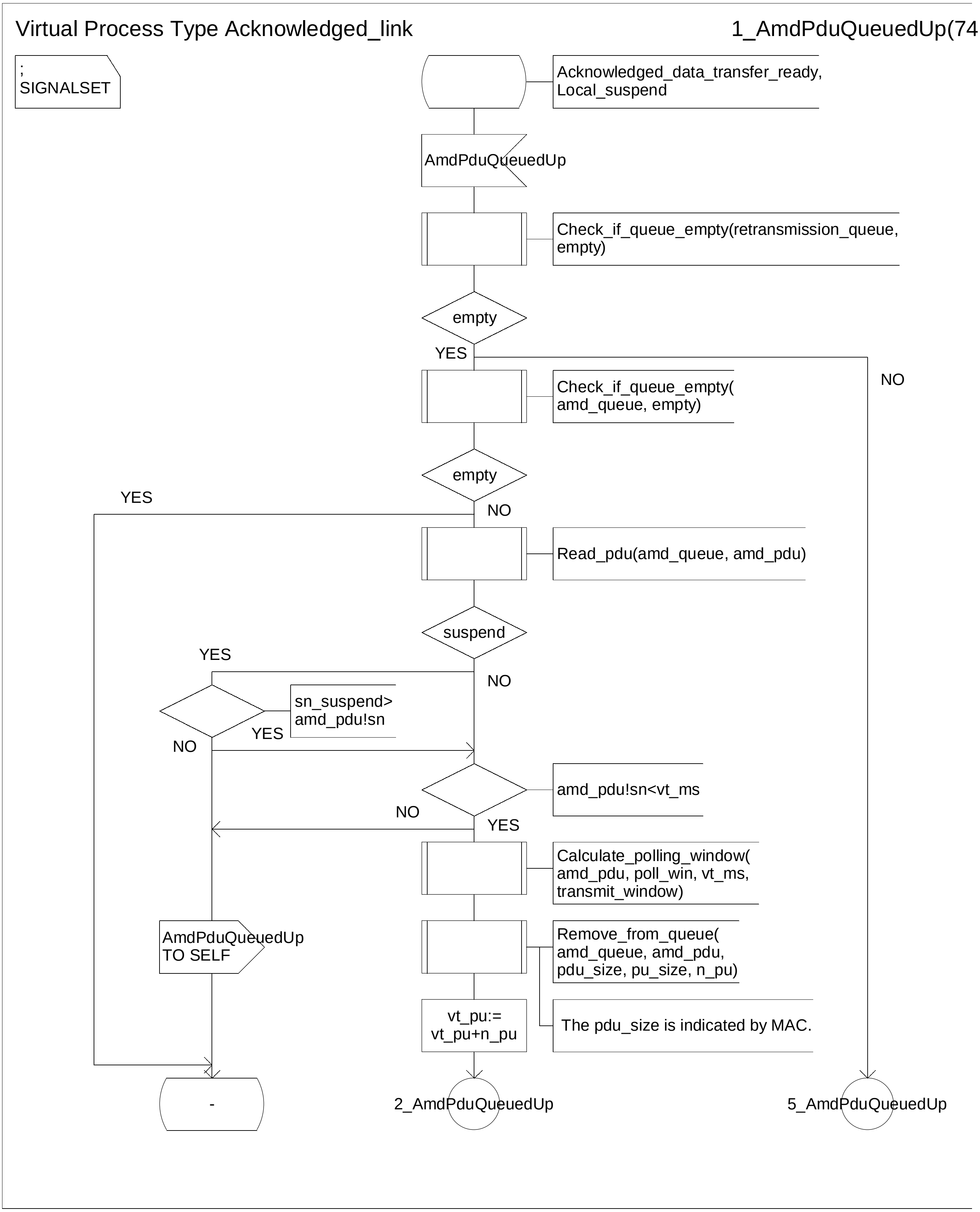

Virtual Process Type Acknowledged_link
2_DataTransferReadyAndLocalSuspend(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Reset_am
Reset_am_ack
rst_pdu!rsn=rx_rsn
YES
NO
Transmit_reset_ack
(logical_channel)
rx_rsn:=rst_pdu!rsn
hfn:=hfn+1
Queue_initialisations(amd_queue, transmitted_queue,
retransmission_queue, receiver_queue,
assembly_queue, mui_queue)
Initialise_state_variables(vt_s, vt_ms, vt_sdu, vt_pu,
vt_a, vt_rst, vt_mrw, vr_r, vr_h, vr_mr, vr_ep)
Transmit_reset_ack(logical_channel)
Crlc_status_ind(EVC)
VIA Cont
end_state:=ACK
-
1_TimerInit
-


Virtual Process Type Acknowledged_link
1_AcknowledgedDataTransferReady(74
;
SIGNALSET
Acknowledged_data_transfer_ready
Crlc_suspend_req(n_susp)
suspend:=YES
Crlc_suspend_cnf(vt_s)
TO SENDER
sn_suspend:=vt_s+n_susp
Local_suspend

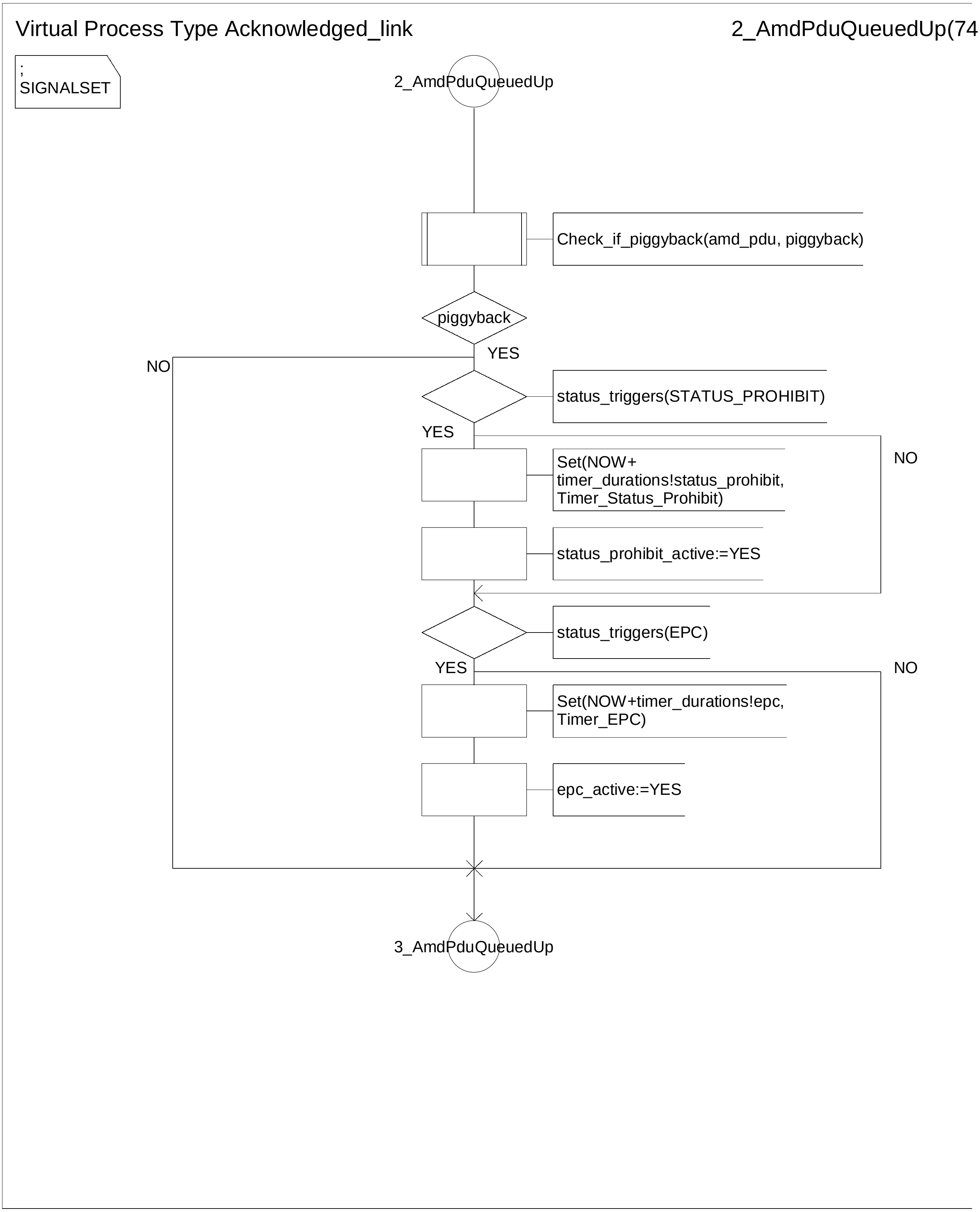

Virtual Process Type Acknowledged_link
1_LocalSuspend(74
;
SIGNALSET
Local_suspend
Crlc_resume_req
suspend:=NO
Acknowledged_data_transfer_ready

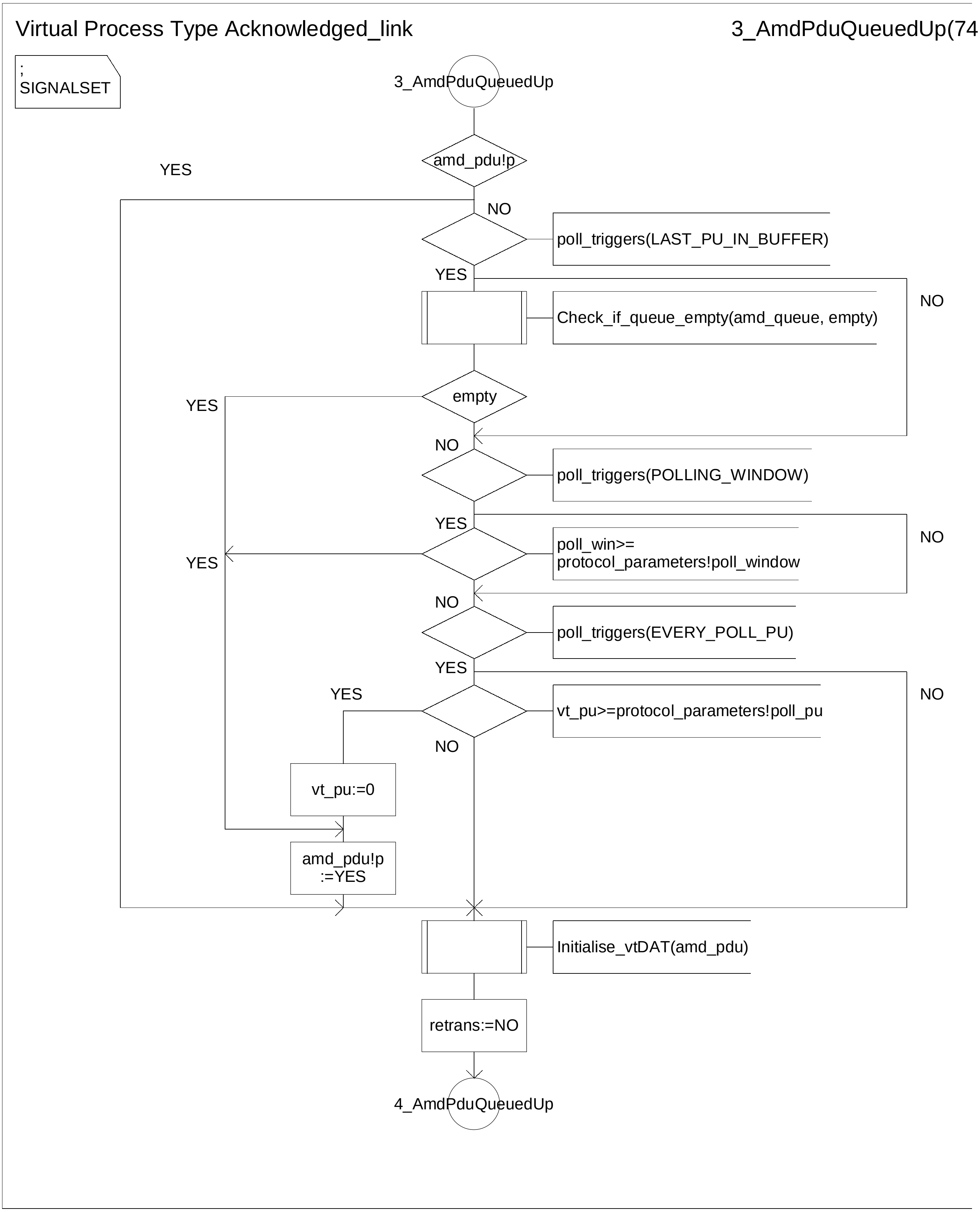

Virtual Process Type Acknowledged_link
1_RlcAmDataReq(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Rlc_AmData_req(sdu, cnf, mui)
discard(TIMER_BASED)
NO
YES
Set(NOW+timer_durations!discard,
Timer_Discard(mui))
Place_in_mui_queue(mui_queue, mui)
Sdu_am_segmentation(sdu, cnf, n_pdu, pdus,
amd_queue, poll_triggers, protocol_parameters, vt_sdu,
ciphering_mode, ciphering_key, hfn, mui, pdu_size, pu_siz
NO
n_pdu=0
YES
The parameter "pdu_size" is indicated by MAC.
2_RlcAmDataReq
-

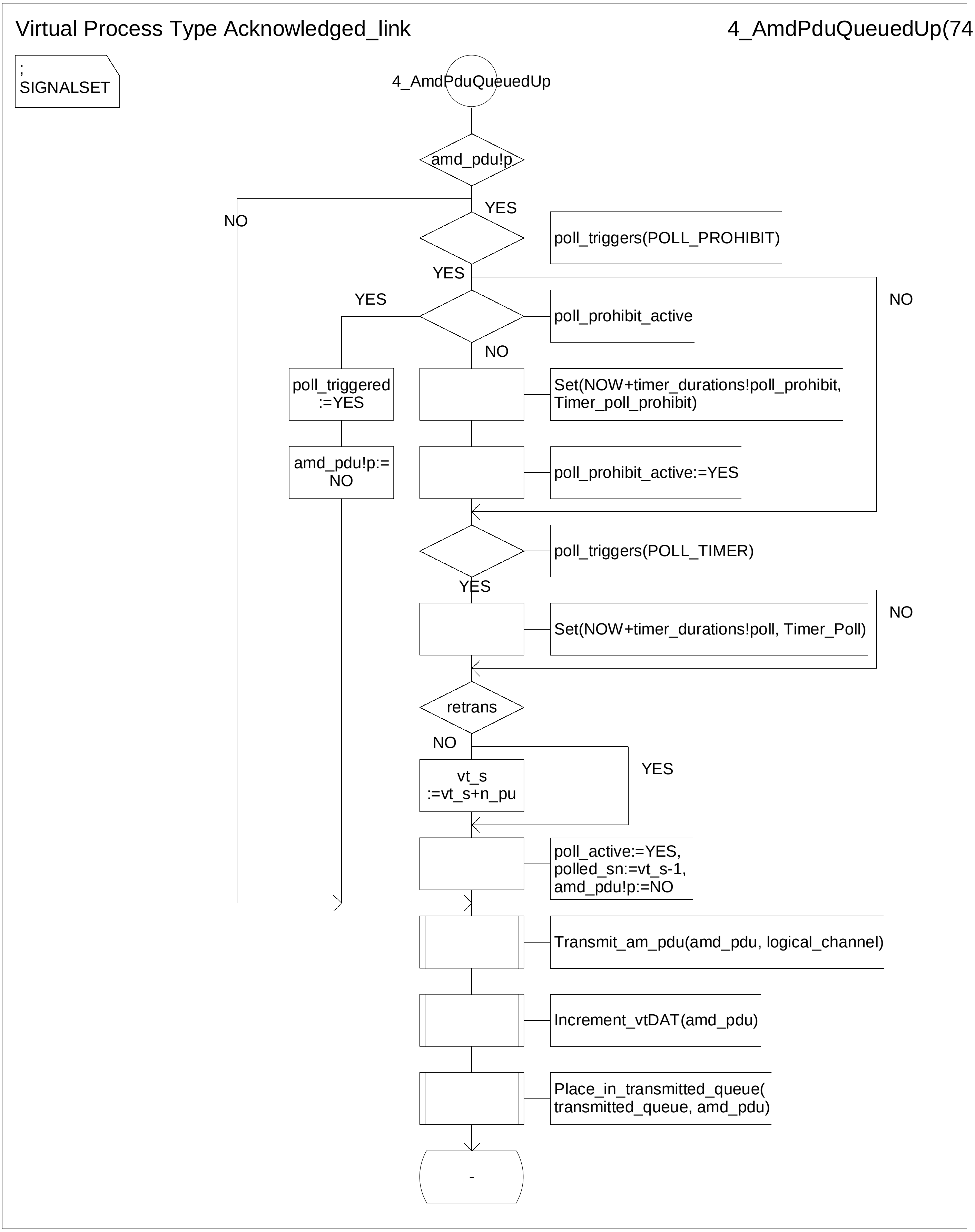

Virtual Process Type Acknowledged_link
2_RlcAmDataReq(74
;
SIGNALSET
2_RlcAmDataReq
n:=1
amd_pdu:=pdus(n)
Set_sequence_number(amd_pdu, vt_s)
Place_in_queue(amd_queue, amd_pdu)
AmdPduQueuedUp TO SELF
n<n_pdu
YES
NO
n:=n+1
-



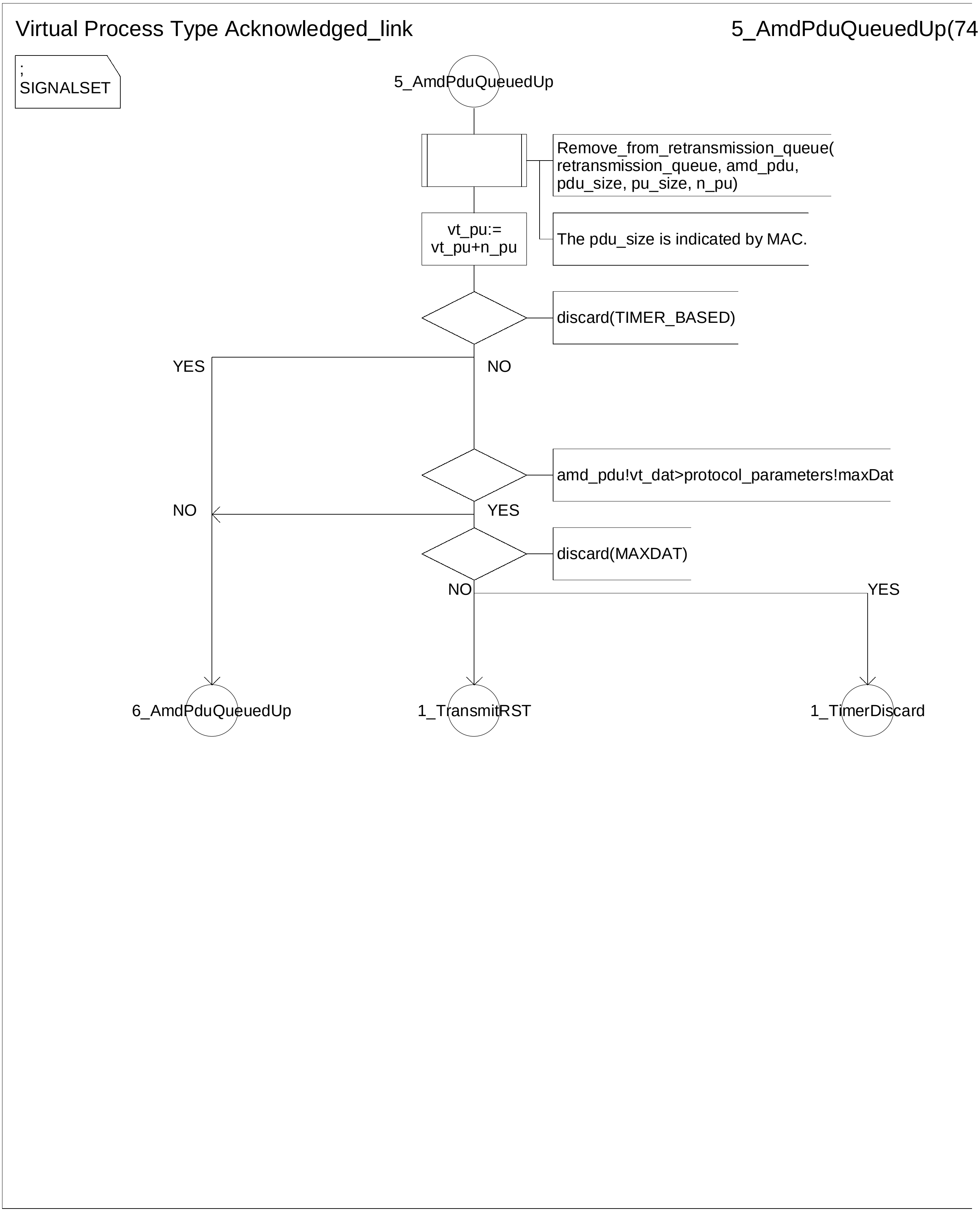
Virtual Process Type Acknowledged_link
1_AmdPduQueuedUp(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
AmdPduQueuedUp
Check_if_queue_empty(retransmission_queue,
empty)
empty
YES
NO
Check_if_queue_empty(
amd_queue, empty)
empty
YES
NO
Read_pdu(amd_queue, amd_pdu)
suspend
YES
NO
sn_suspend>
amd_pdu!sn
YES
NO
amd_pdu!sn<vt_ms
NO
YES
Calculate_polling_window(
amd_pdu, poll_win, vt_ms,
transmit_window)
AmdPduQueuedUp
TO SELF
Remove_from_queue(
amd_queue, amd_pdu,
pdu_size, pu_size, n_pu)
vt_pu:=
vt_pu+n_pu
The pdu_size is indicated by MAC.
-
2_AmdPduQueuedUp
5_AmdPduQueuedUp

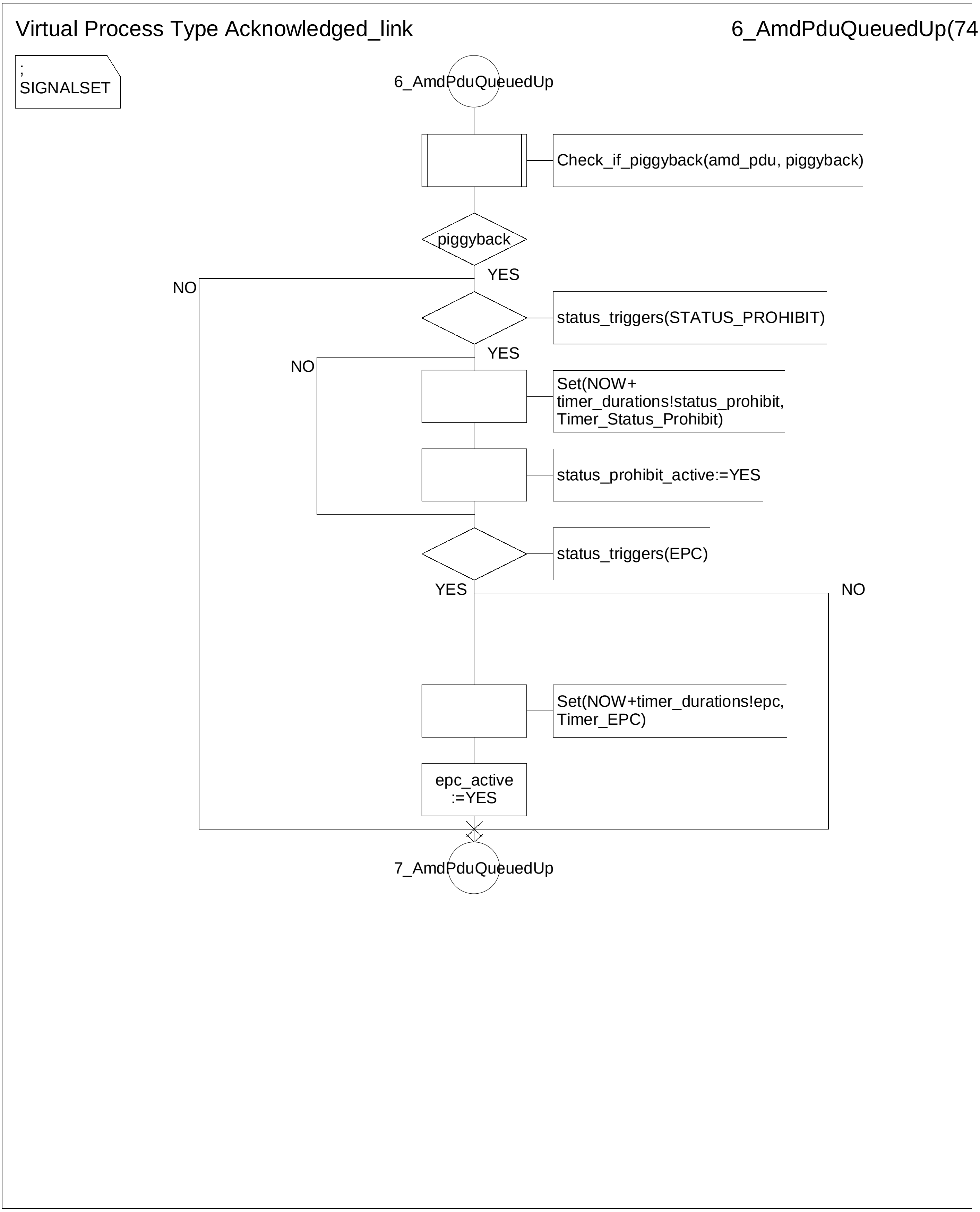

Virtual Process Type Acknowledged_link
2_AmdPduQueuedUp(74
;
SIGNALSET
2_AmdPduQueuedUp
Check_if_piggyback(amd_pdu, piggyback)
piggyback
YES
NO
status_triggers(STATUS_PROHIBIT)
YES
NO
Set(NOW+
timer_durations!status_prohibit,
Timer_Status_Prohibit)
status_prohibit_active:=YES
status_triggers(EPC)
YES
NO
Set(NOW+timer_durations!epc,
Timer_EPC)
epc_active:=YES
3_AmdPduQueuedUp

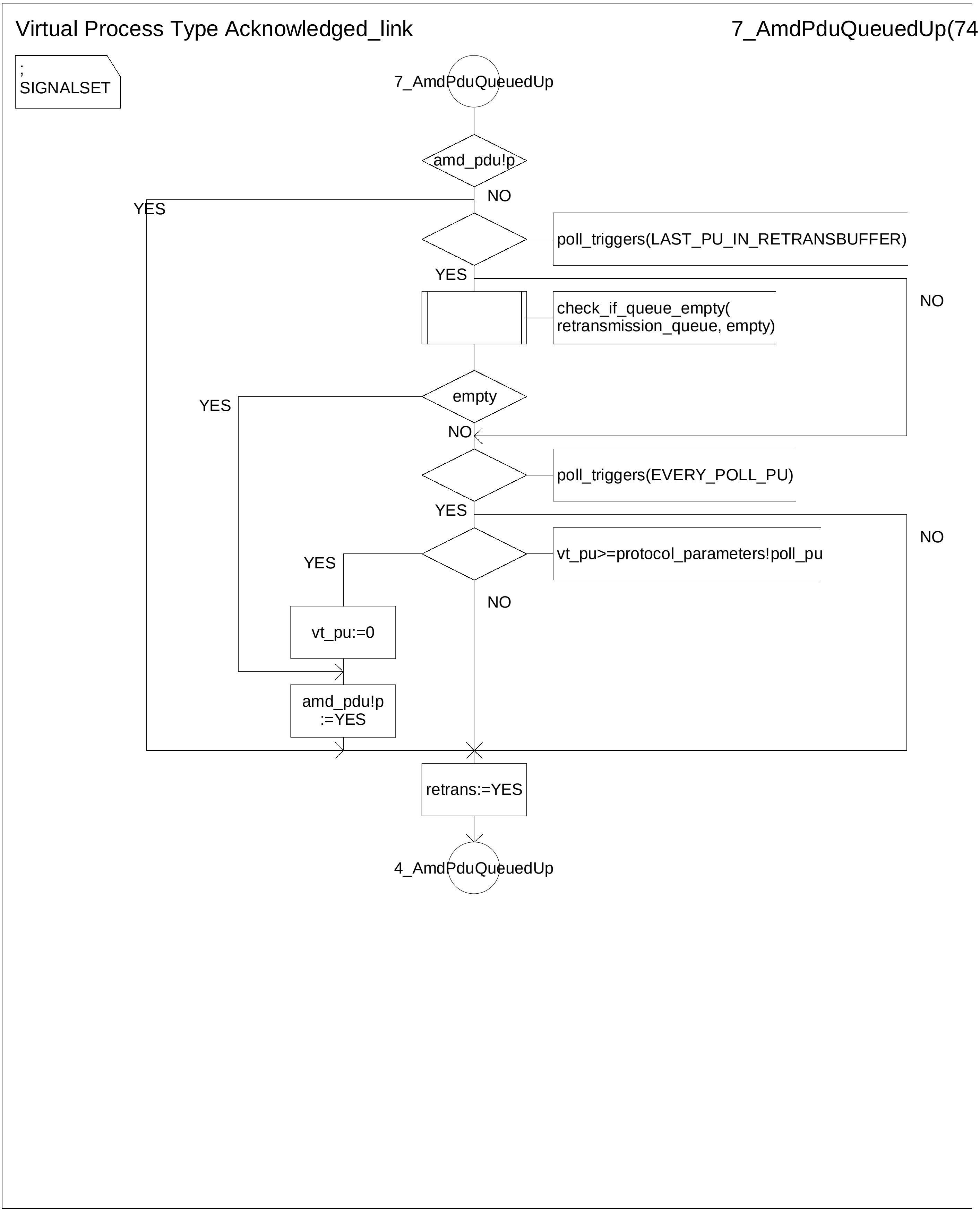

Virtual Process Type Acknowledged_link
3_AmdPduQueuedUp(74
;
SIGNALSET
3_AmdPduQueuedUp
amd_pdu!p
YES
NO
poll_triggers(LAST_PU_IN_BUFFER)
YES
NO
Check_if_queue_empty(amd_queue, empty)
empty
YES
NO
poll_triggers(POLLING_WINDOW)
YES
NO
poll_win>=
protocol_parameters!poll_window
YES
NO
poll_triggers(EVERY_POLL_PU)
YES
NO
vt_pu>=protocol_parameters!poll_pu
YES
NO
vt_pu:=0
amd_pdu!p
:=YES
Initialise_vtDAT(amd_pdu)
retrans:=NO
4_AmdPduQueuedUp

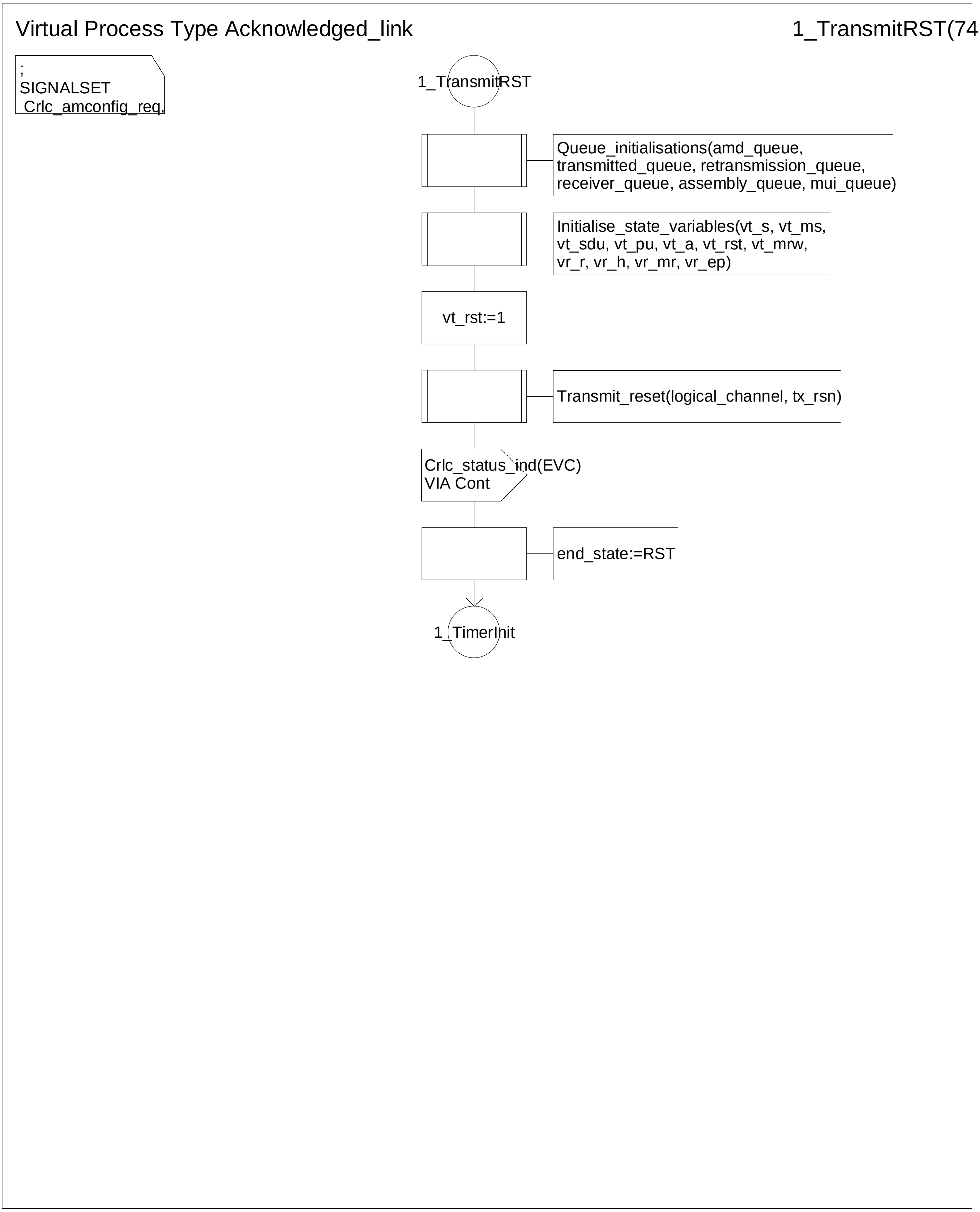


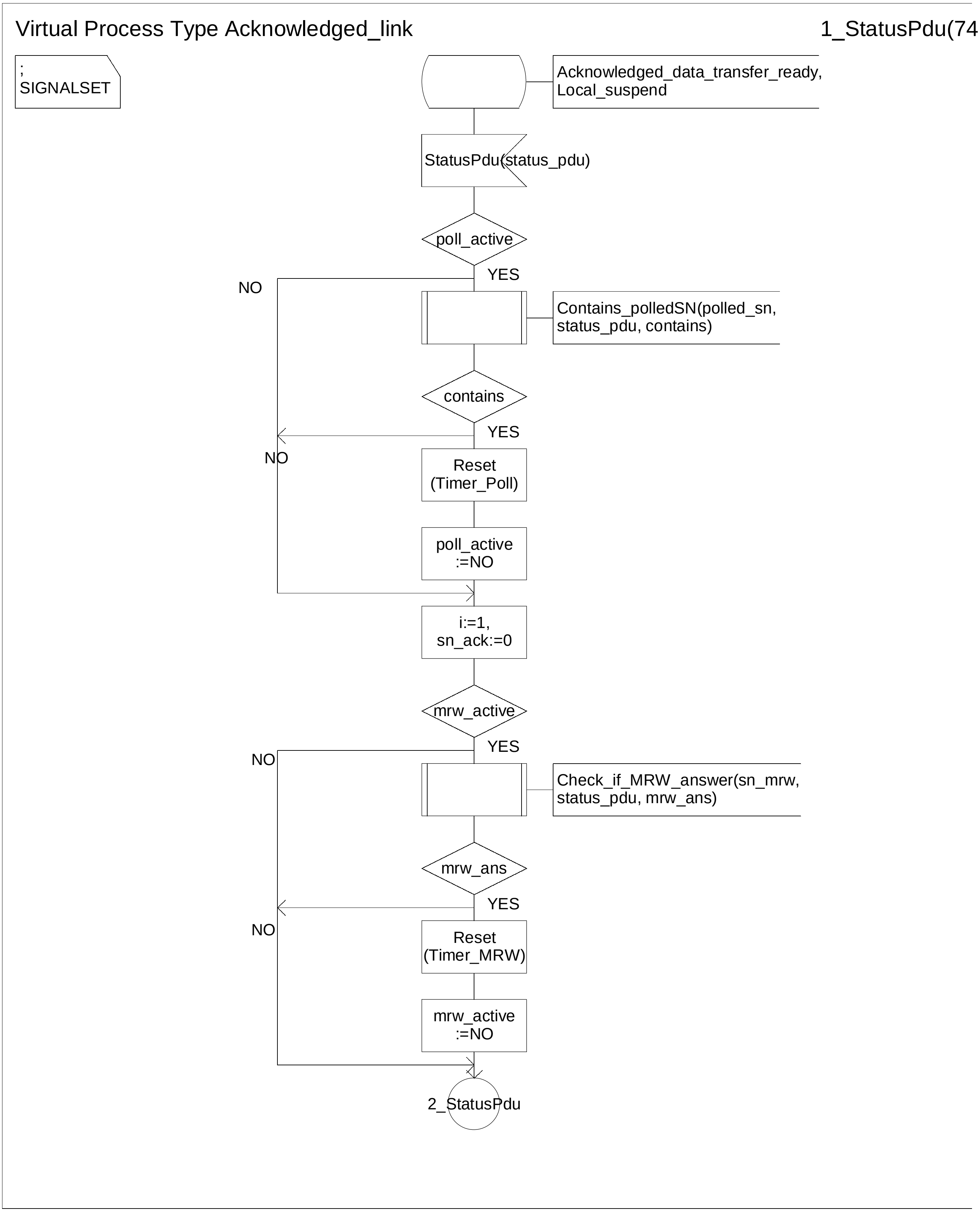
Virtual Process Type Acknowledged_link
4_AmdPduQueuedUp(74
;
SIGNALSET
4_AmdPduQueuedUp
amd_pdu!p
YES
NO
poll_triggers(POLL_PROHIBIT)
YES
NO
YES
poll_prohibit_active
NO
poll_triggered
:=YES
Set(NOW+timer_durations!poll_prohibit,
Timer_poll_prohibit)
amd_pdu!p:=
NO
poll_prohibit_active:=YES
poll_triggers(POLL_TIMER)
YES
NO
Set(NOW+timer_durations!poll, Timer_Poll)
retrans
NO
YES
vt_s
:=vt_s+n_pu
poll_active:=YES,
polled_sn:=vt_s-1,
amd_pdu!p:=NO
Transmit_am_pdu(amd_pdu, logical_channel)
Increment_vtDAT(amd_pdu)
Place_in_transmitted_queue(
transmitted_queue, amd_pdu)
-


Virtual Process Type Acknowledged_link
5_AmdPduQueuedUp(74
;
SIGNALSET
5_AmdPduQueuedUp
Remove_from_retransmission_queue(
retransmission_queue, amd_pdu,
pdu_size, pu_size, n_pu)
vt_pu:=
vt_pu+n_pu
The pdu_size is indicated by MAC.
discard(TIMER_BASED)
YES
NO
amd_pdu!vt_dat>protocol_parameters!maxDat
NO
YES
discard(MAXDAT)
NO
YES
6_AmdPduQueuedUp
1_TransmitRST
1_TimerDiscard

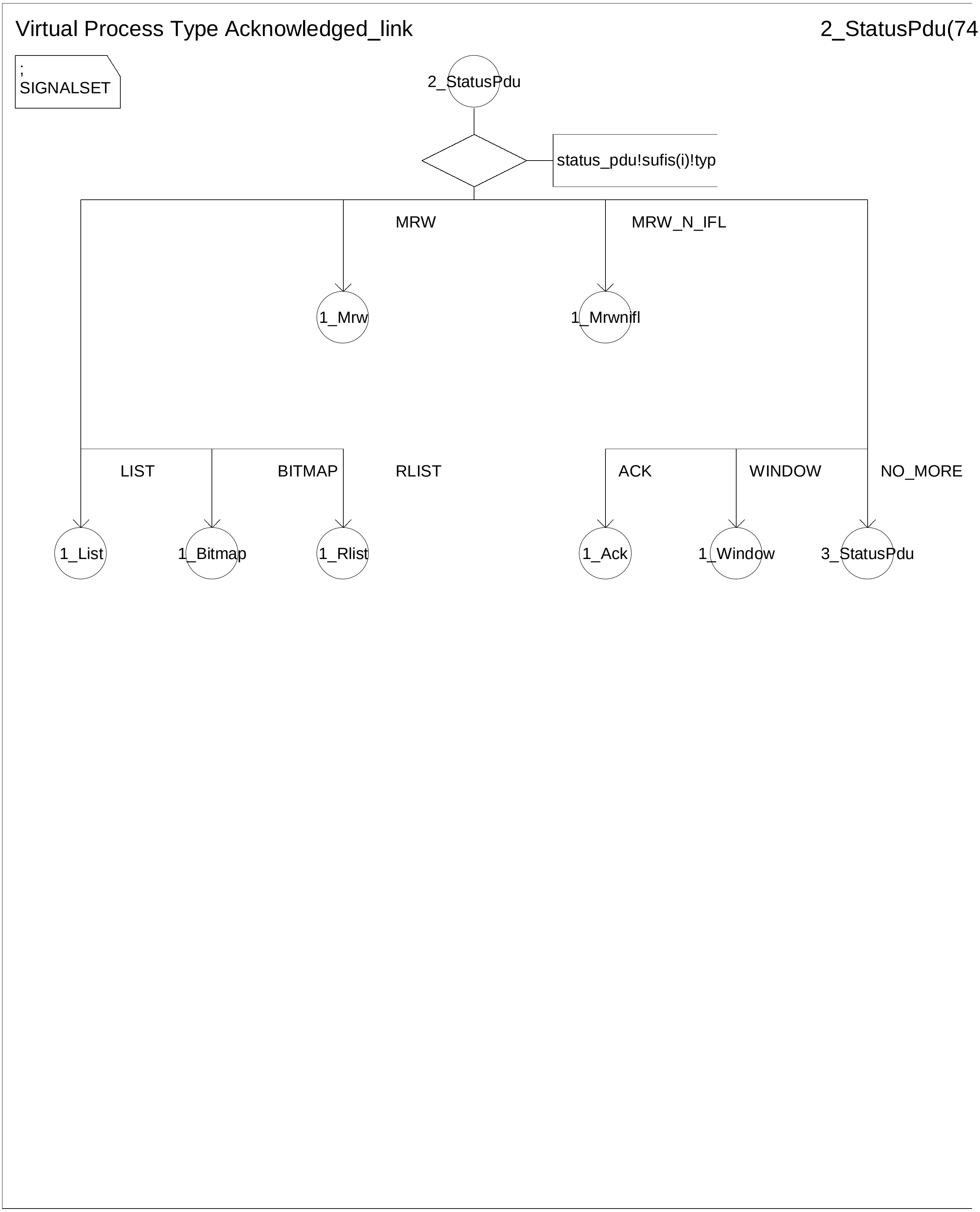

Virtual Process Type Acknowledged_link
6_AmdPduQueuedUp(74
;
SIGNALSET
6_AmdPduQueuedUp
Check_if_piggyback(amd_pdu, piggyback)
piggyback
YES
NO
status_triggers(STATUS_PROHIBIT)
YES
NO
Set(NOW+
timer_durations!status_prohibit,
Timer_Status_Prohibit)
status_prohibit_active:=YES
status_triggers(EPC)
YES
NO
Set(NOW+timer_durations!epc,
Timer_EPC)
epc_active
:=YES
7_AmdPduQueuedUp

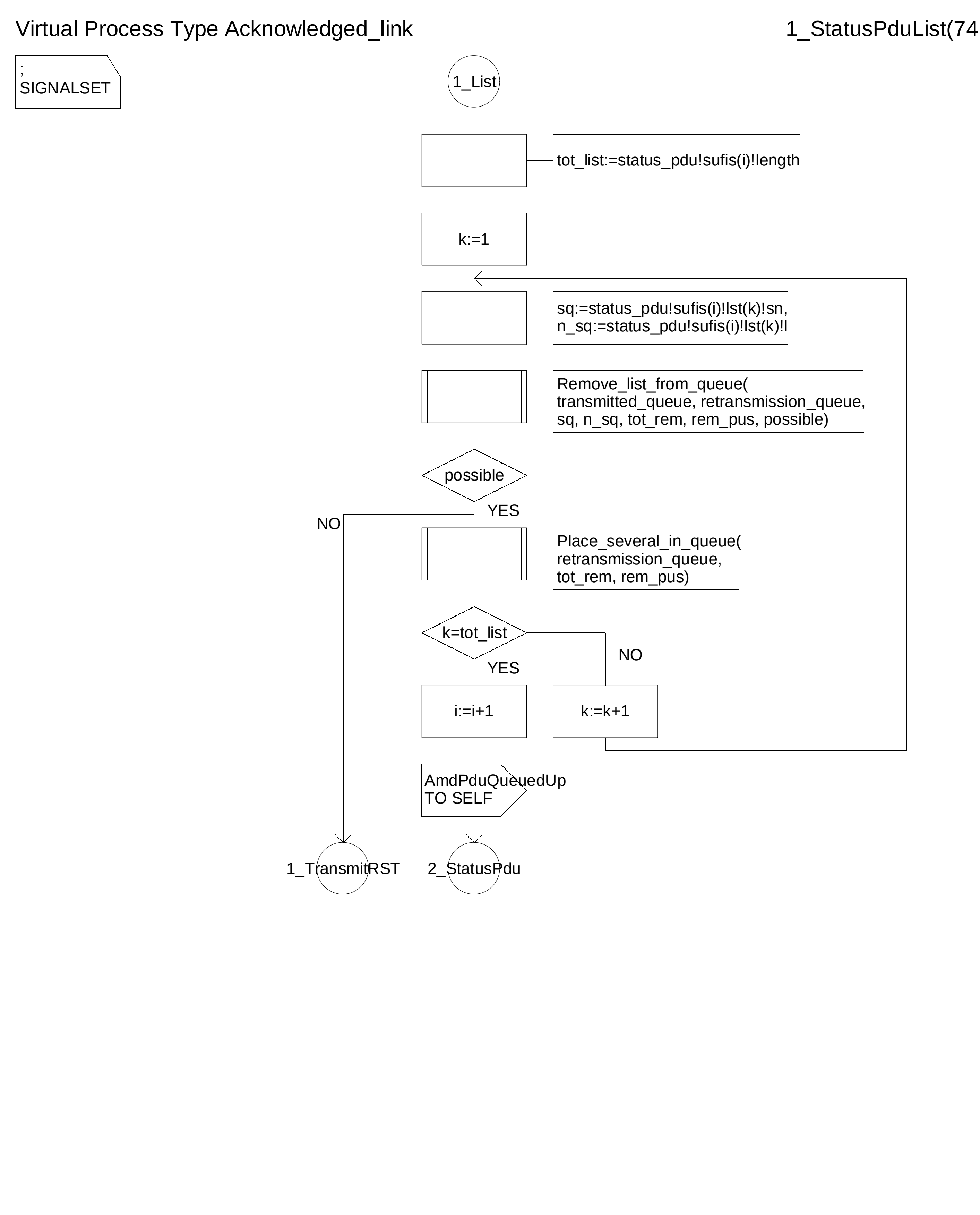

Virtual Process Type Acknowledged_link
7_AmdPduQueuedUp(74
;
SIGNALSET
7_AmdPduQueuedUp
amd_pdu!p
NO
YES
poll_triggers(LAST_PU_IN_RETRANSBUFFER)
YES
NO
check_if_queue_empty(
retransmission_queue, empty)
YES
empty
NO
poll_triggers(EVERY_POLL_PU)
YES
NO
YES
vt_pu>=protocol_parameters!poll_pu
NO
vt_pu:=0
amd_pdu!p
:=YES
retrans:=YES
4_AmdPduQueuedUp

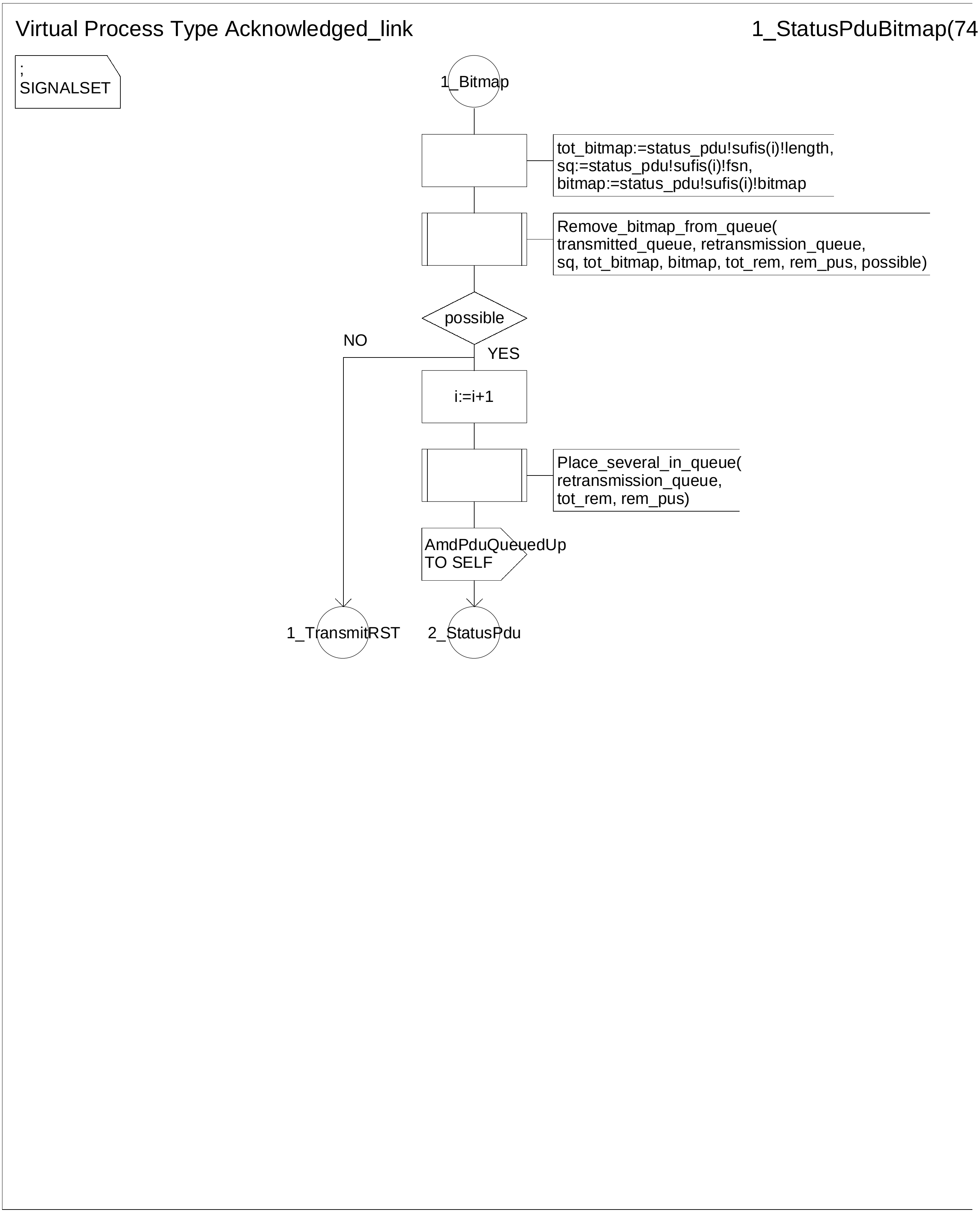

Virtual Process Type Acknowledged_link
1_TransmitRST(74
;
SIGNALSET
Crlc_amconfig_req,
1_TransmitRST
Queue_initialisations(amd_queue, transmitted_queue, retransmission_queue, receiver_queue, assembly_queue, mui_queue)
Initialise_state_variables(vt_s, vt_ms, vt_sdu, vt_pu, vt_a, vt_rst, vt_mrw, vr_r, vr_h, vr_mr, vr_ep)
vt_rst:=1
Transmit_reset(logical_channel, tx_rsn)
Crlc_status_ind(EVC)
VIA Cont
end_state:=RST
1_TimerInit

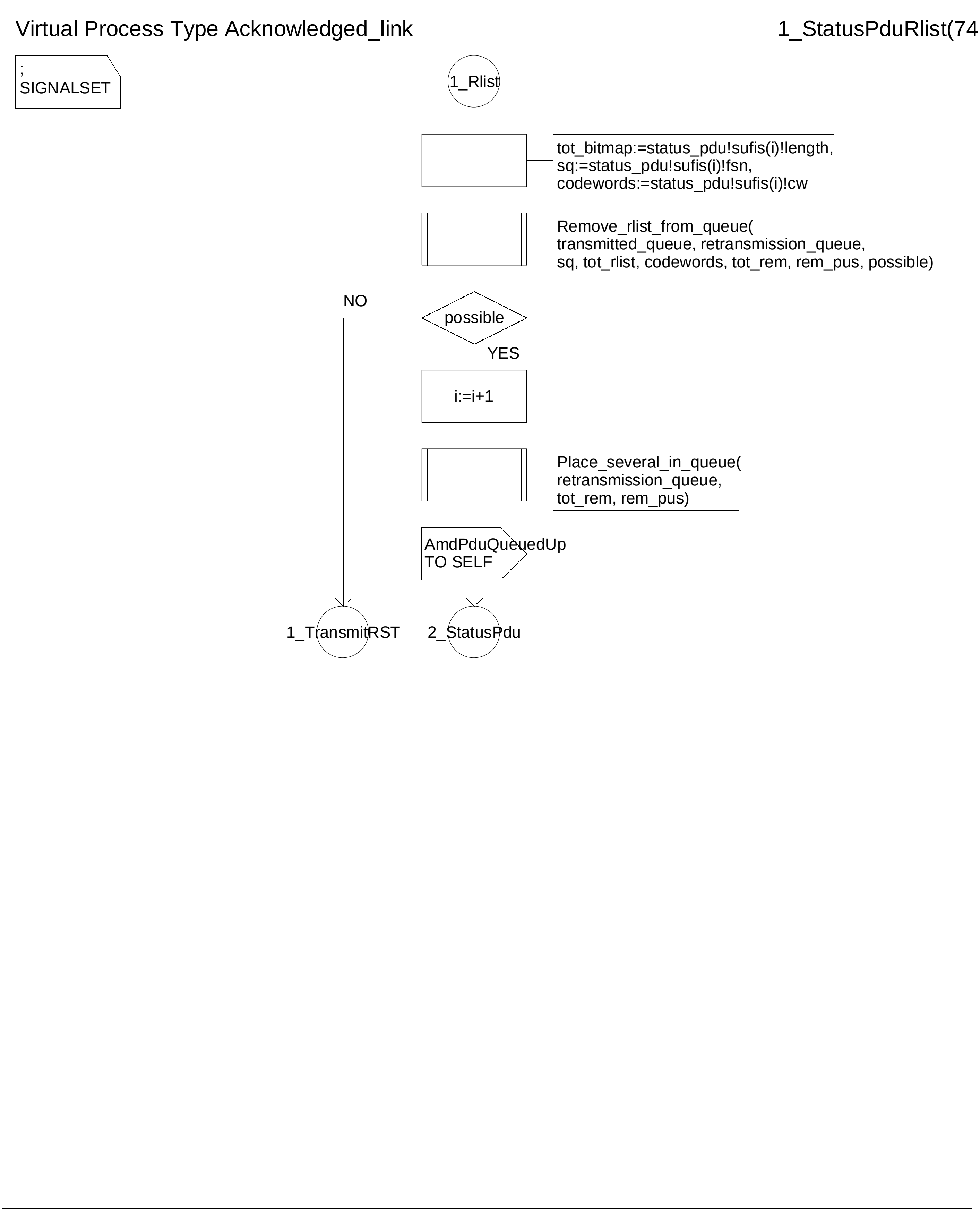


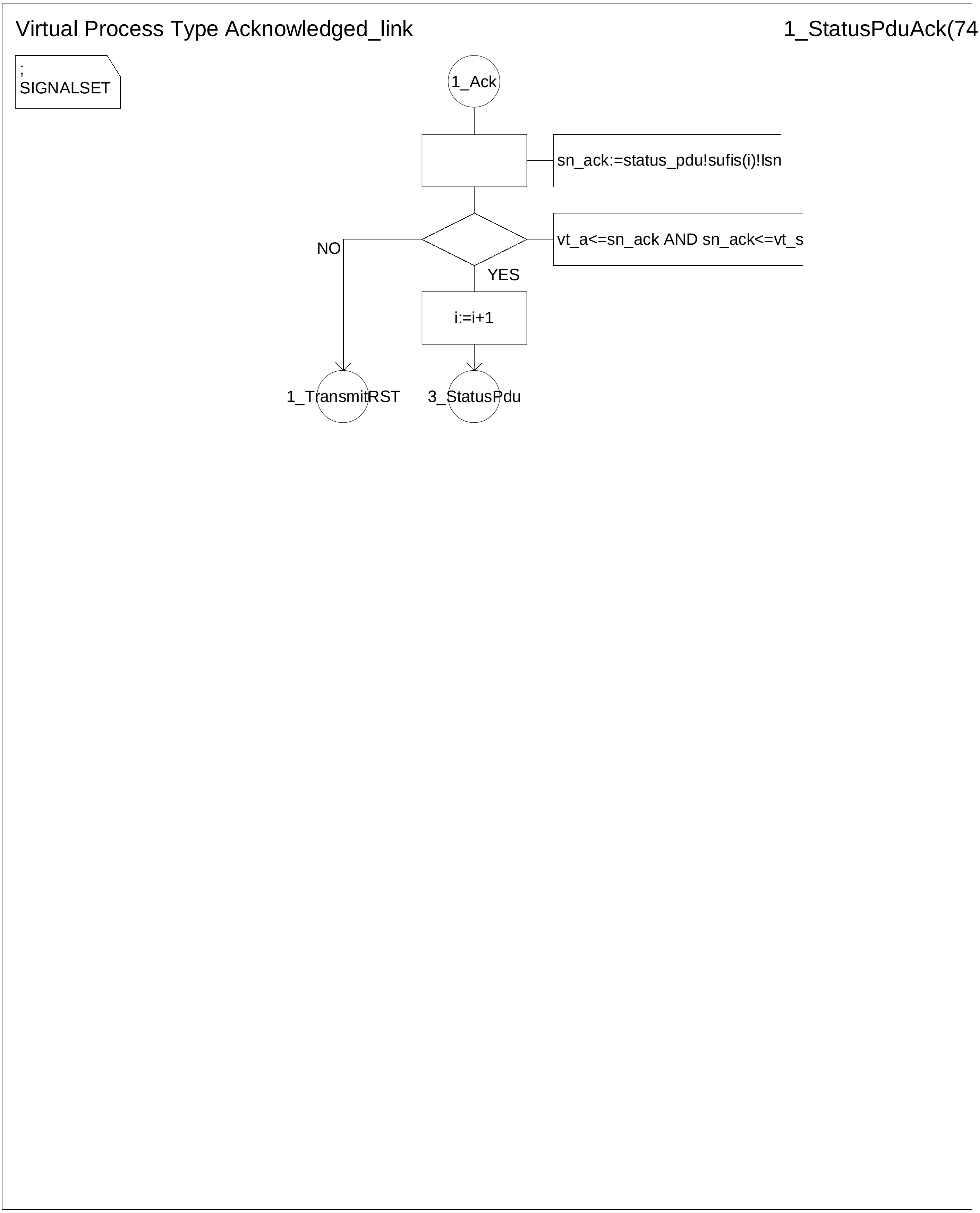
Virtual Process Type Acknowledged_link
1_StatusPdu(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
StatusPdu(status_pdu)
poll_active
YES
NO
Contains_polledSN(polled_sn,
status_pdu, contains)
contains
YES
NO
Reset
(Timer_Poll)
poll_active
:=NO
i:=1,
sn_ack:=0
mrw_active
YES
NO
Check_if_MRW_answer(sn_mrw,
status_pdu, mrw_ans)
mrw_ans
YES
NO
Reset
(Timer_MRW)
mrw_active
:=NO
2_StatusPdu

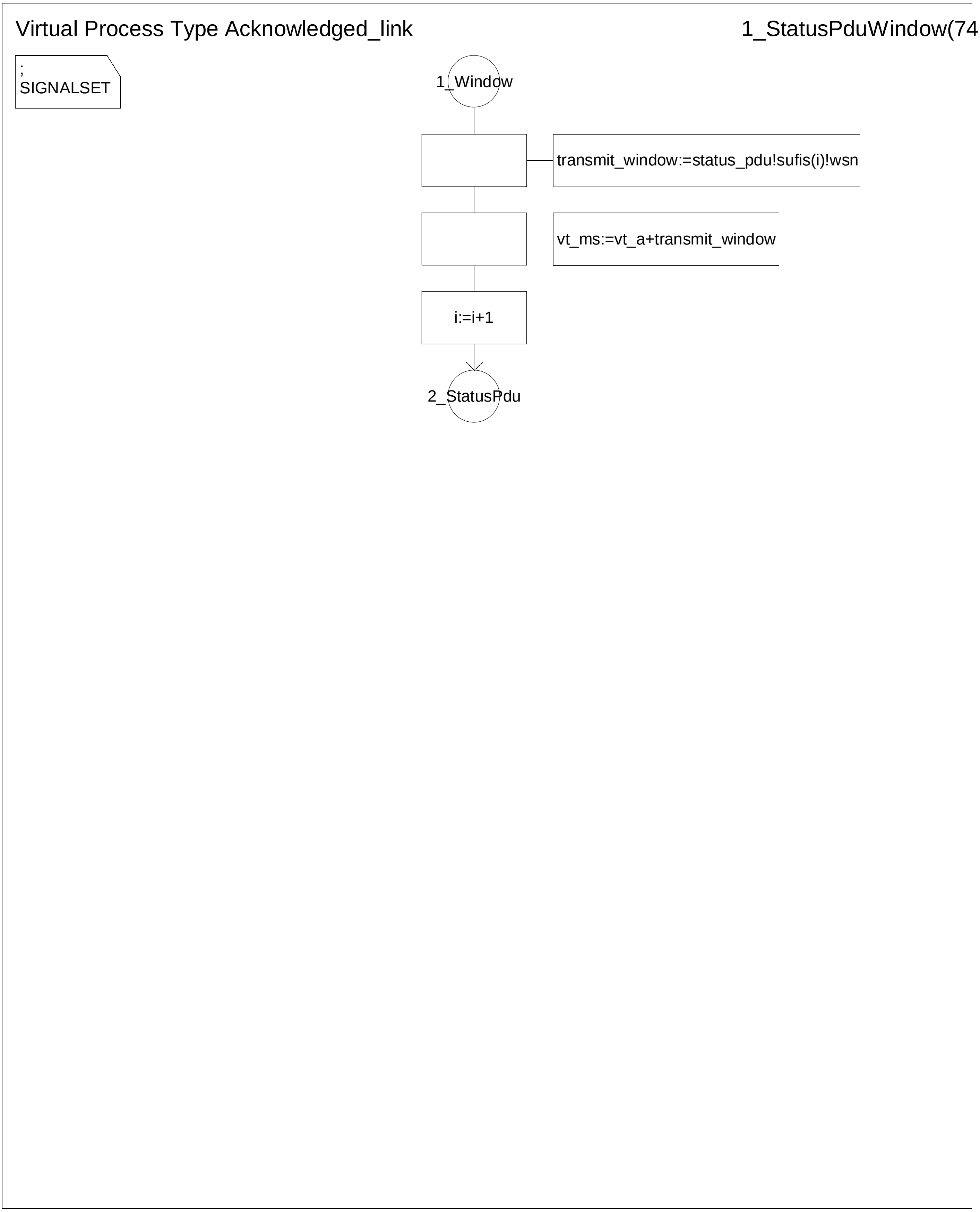

Virtual Process Type Acknowledged_link
2_StatusPdu(74
;
SIGNALSET
2_StatusPdu
status_pdu!sufis(i)!typ
MRW
MRW_N_IFL
1_Mrw
1_Mrwnifl
LIST
BITMAP
RLIST
ACK
WINDOW
NO_MORE
1_List
1_Bitmap
1_Rlist
1_Ack
1_Window
3_StatusPdu

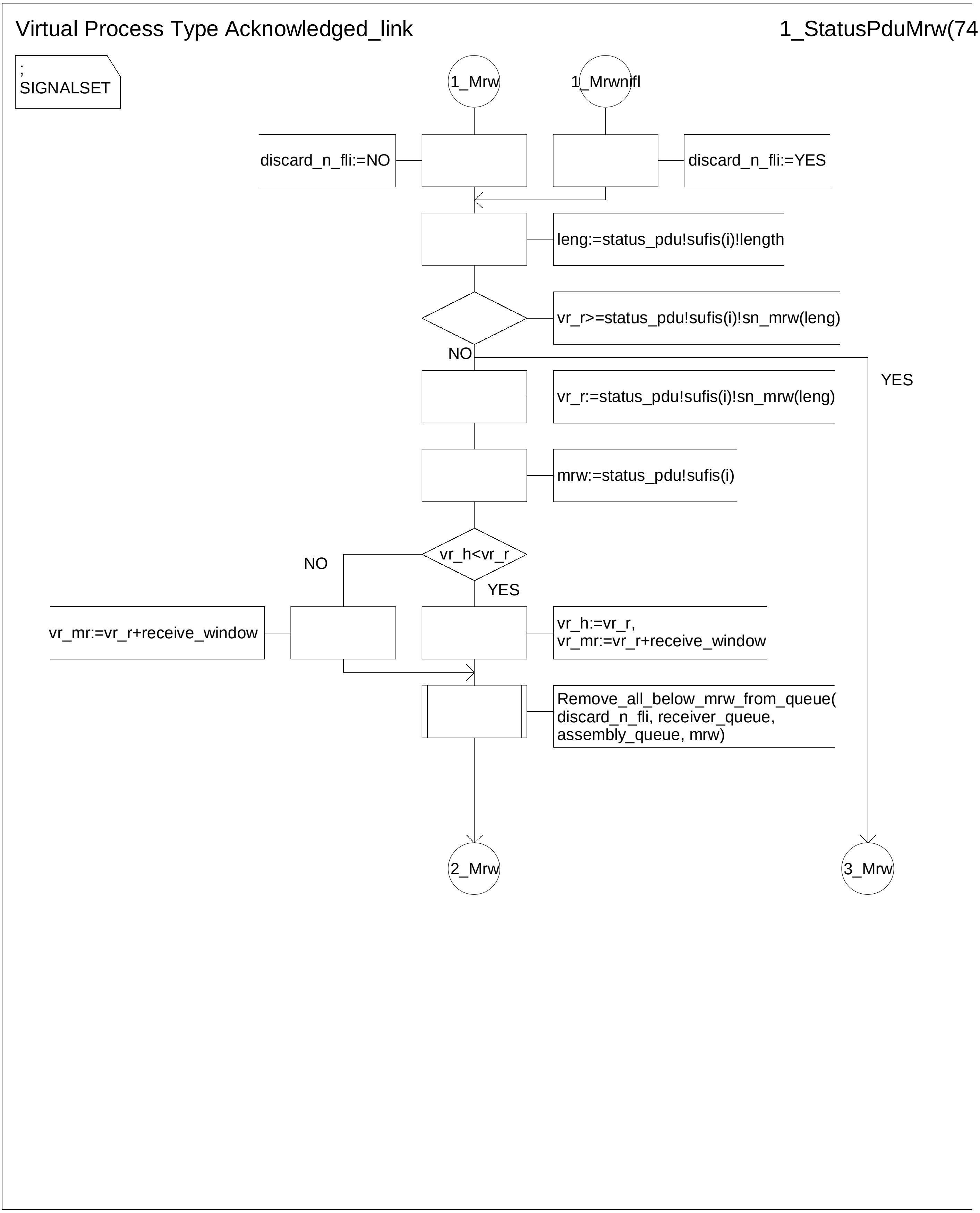

Virtual Process Type Acknowledged_link
1_StatusPduList(74
;
SIGNALSET
1_List
tot_list:=status_pdu!sufis(i)!length
k:=1
sq:=status_pdu!sufis(i)!lst(k)!sn,
n_sq:=status_pdu!sufis(i)!lst(k)!l
Remove_list_from_queue(
transmitted_queue, retransmission_queue,
sq, n_sq, tot_rem, rem_pus, possible)
possible
YES
NO
Place_several_in_queue(
retransmission_queue,
tot_rem, rem_pus)
k=tot_list
YES
NO
i:=i+1
k:=k+1
AmdPduQueuedUp
TO SELF
1_TransmitRST
2_StatusPdu

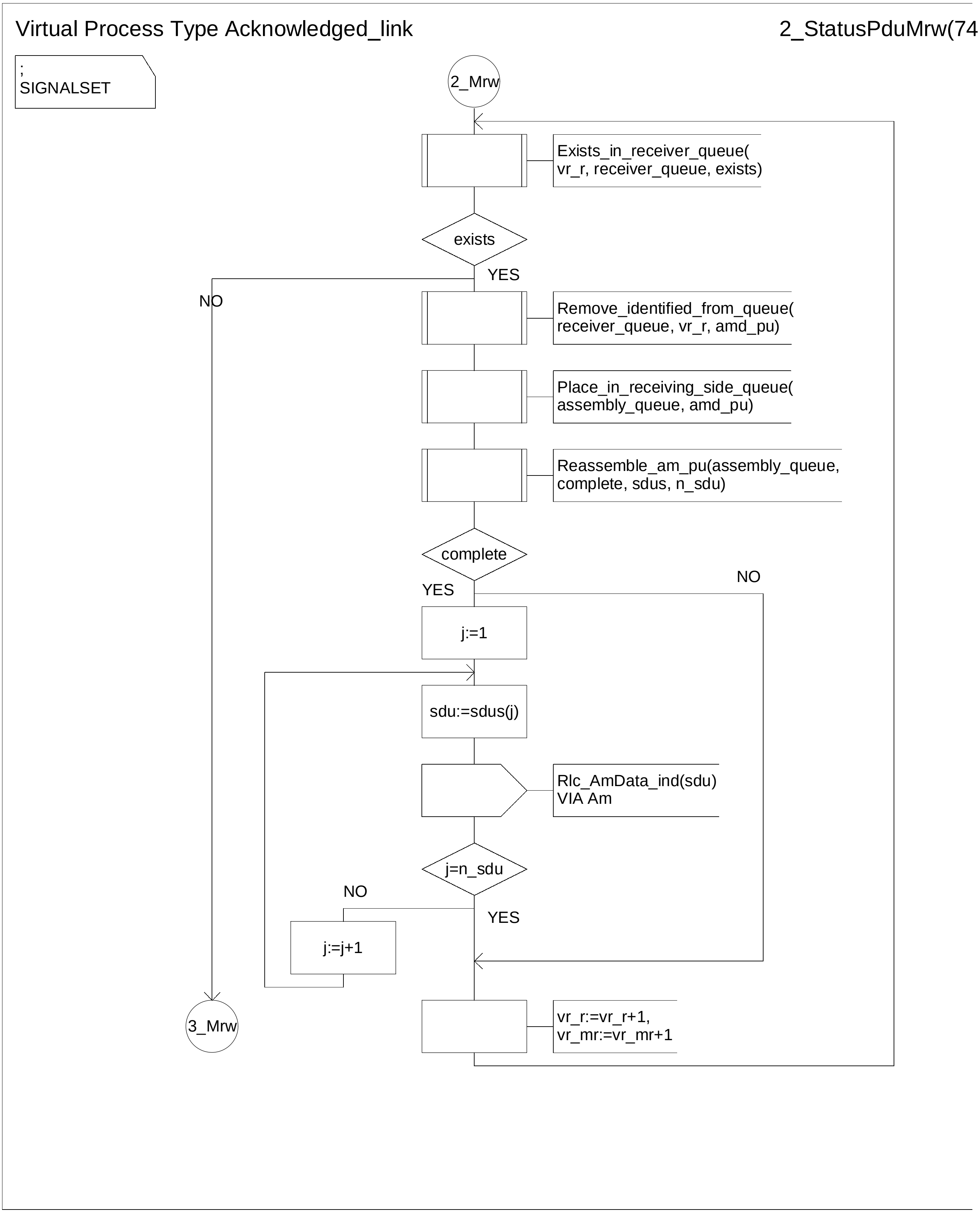

Virtual Process Type Acknowledged_link
1_StatusPduBitmap(74
;
SIGNALSET
1_Bitmap
tot_bitmap:=status_pdu!sufis(i)!length,
sq:=status_pdu!sufis(i)!fsn,
bitmap:=status_pdu!sufis(i)!bitmap
Remove_bitmap_from_queue(
transmitted_queue, retransmission_queue,
sq, tot_bitmap, bitmap, tot_rem, rem_pus, possible)
possible
NO
YES
i:=i+1
Place_several_in_queue(
retransmission_queue,
tot_rem, rem_pus)
AmdPduQueuedUp
TO SELF
1_TransmitRST
2_StatusPdu

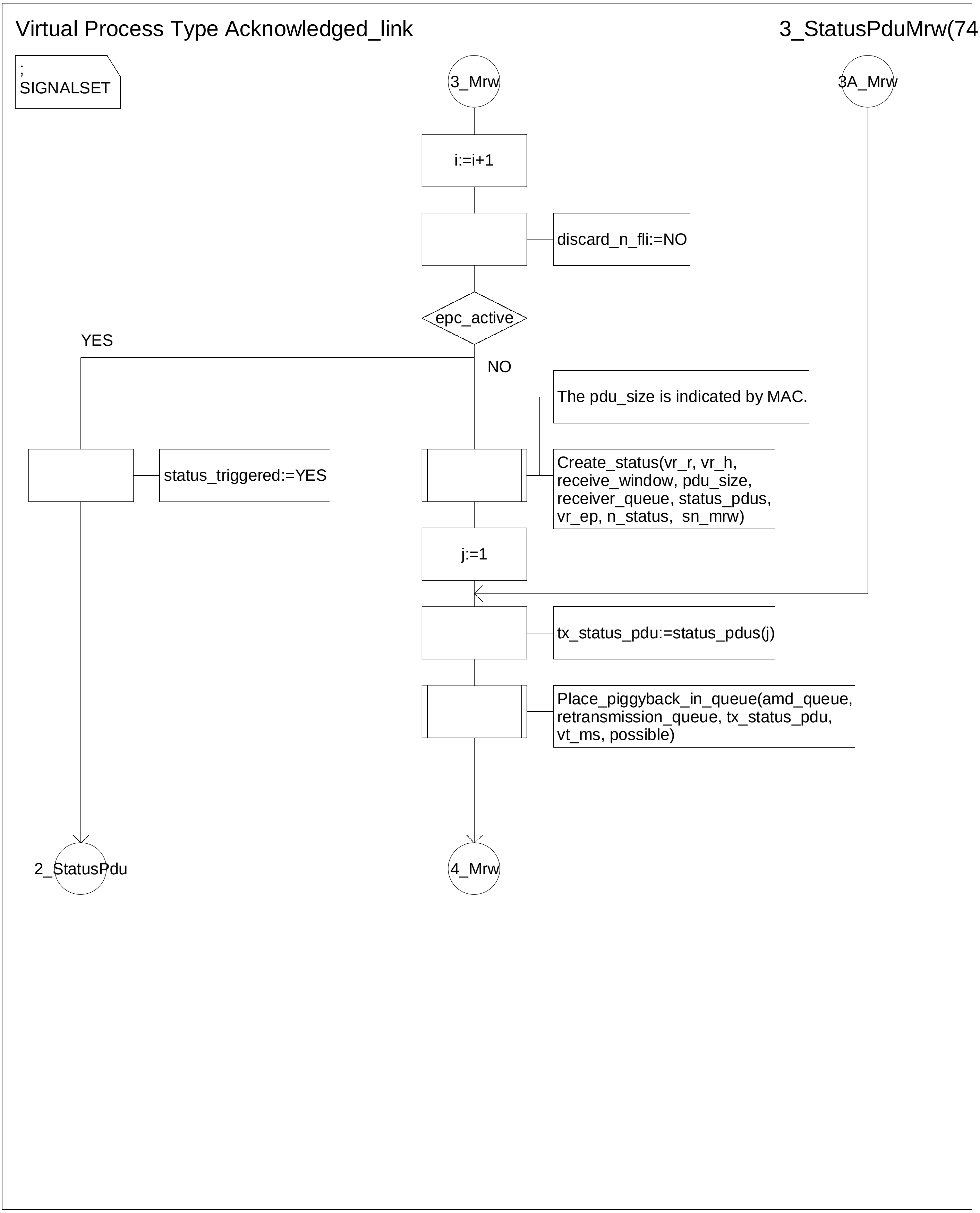

Virtual Process Type Acknowledged_link
1_StatusPduRlist(74
;
SIGNALSET
1_Rlist
tot_bitmap:=status_pdu!sufis(i)!length,
sq:=status_pdu!sufis(i)!fsn,
codewords:=status_pdu!sufis(i)!cw
Remove_rlist_from_queue(
transmitted_queue, retransmission_queue,
sq, tot_rlist, codewords, tot_rem, rem_pus, possible)
NO
possible
YES
i:=i+1
Place_several_in_queue(
retransmission_queue,
tot_rem, rem_pus)
AmdPduQueuedUp
TO SELF
1_TransmitRST
2_StatusPdu

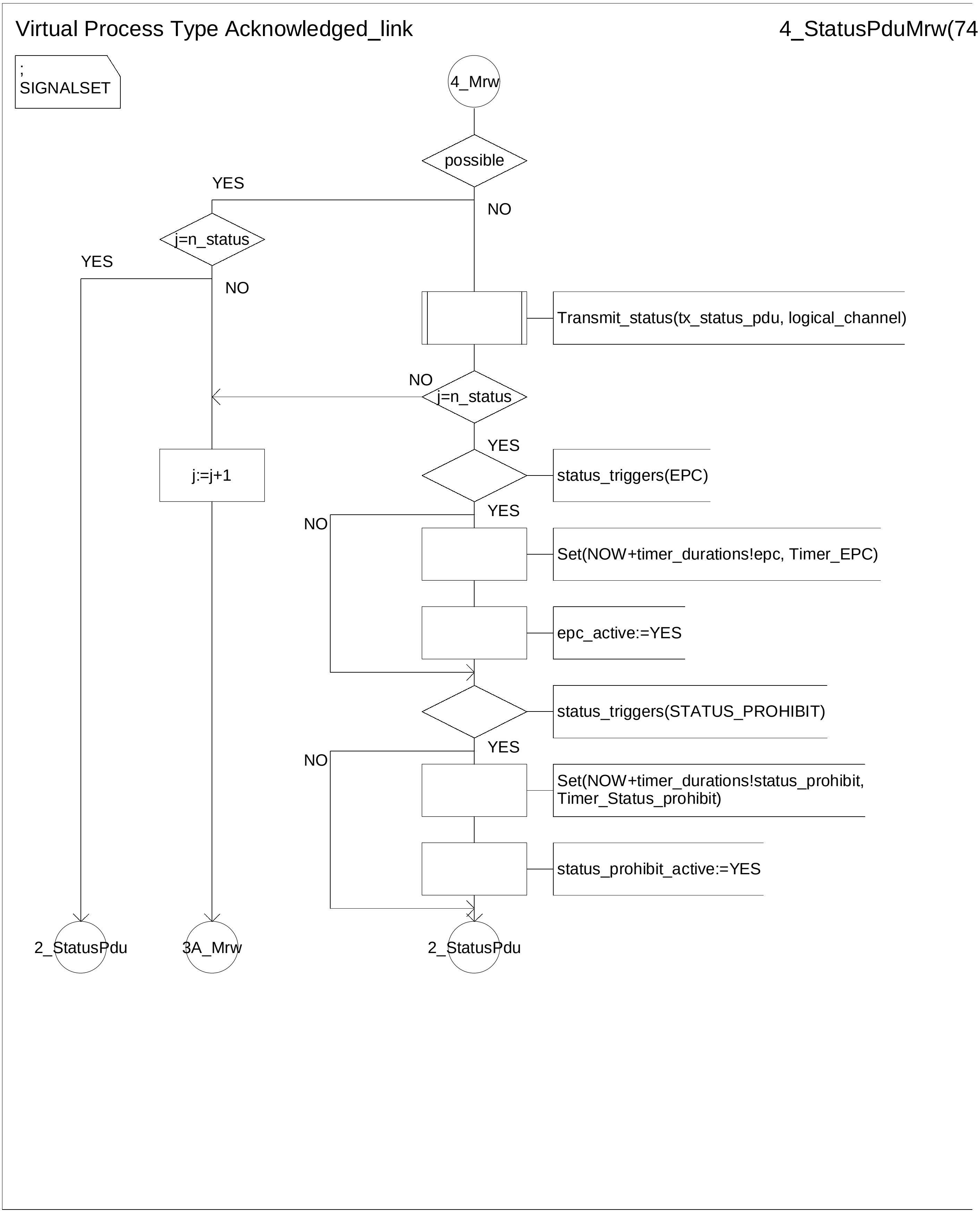

Virtual Process Type Acknowledged_link
1_StatusPduAck(74
;
SIGNALSET
1_Ack
sn_ack:=status_pdu!sufis(i)!lsn
vt_a<=sn_ack AND sn_ack<=vt_s
NO
YES
i:=i+1
1_TransmitRST
3_StatusPdu

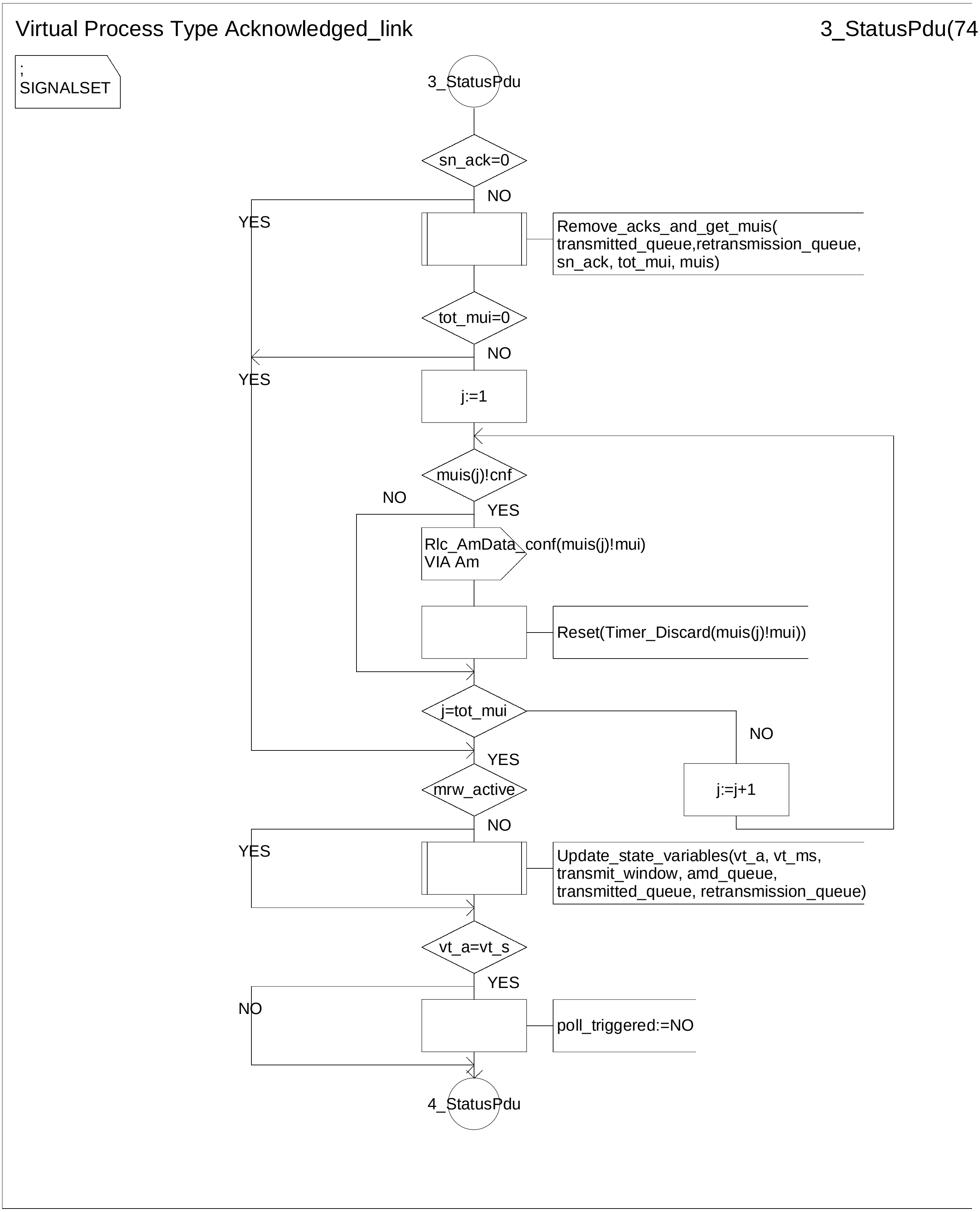

Virtual Process Type Acknowledged_link
1_StatusPduWindow(74
;
SIGNALSET
1_Window
transmit_window:=status_pdu!sufis(i)!wsn
vt_ms:=vt_a+transmit_window
i:=i+1
2_StatusPdu

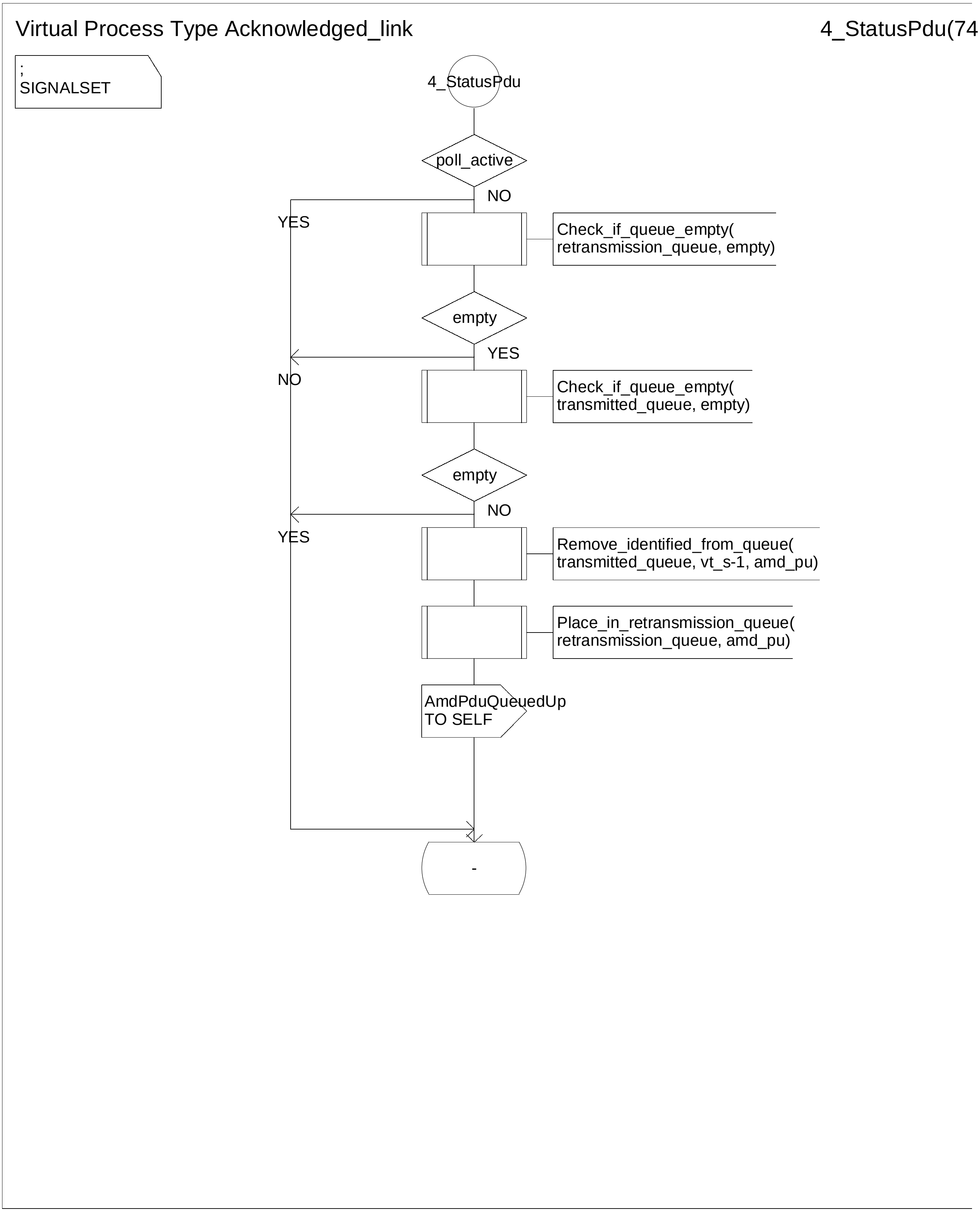

Virtual Process Type Acknowledged_link
1_StatusPduMrw(74
;
SIGNALSET
1_Mrw
1_Mrwnifl
discard_n_fli:=NO
discard_n_fli:=YES
leng:=status_pdu!sufis(i)!length
vr_r>=status_pdu!sufis(i)!sn_mrw(leng)
NO
YES
vr_r:=status_pdu!sufis(i)!sn_mrw(leng)
mrw:=status_pdu!sufis(i)
vr_h<vr_r
NO
YES
vr_mr:=vr_r+receive_window
vr_h:=vr_r,
vr_mr:=vr_r+receive_window
Remove_all_below_mrw_from_queue(
discard_n_fli, receiver_queue,
assembly_queue, mrw)
2_Mrw
3_Mrw

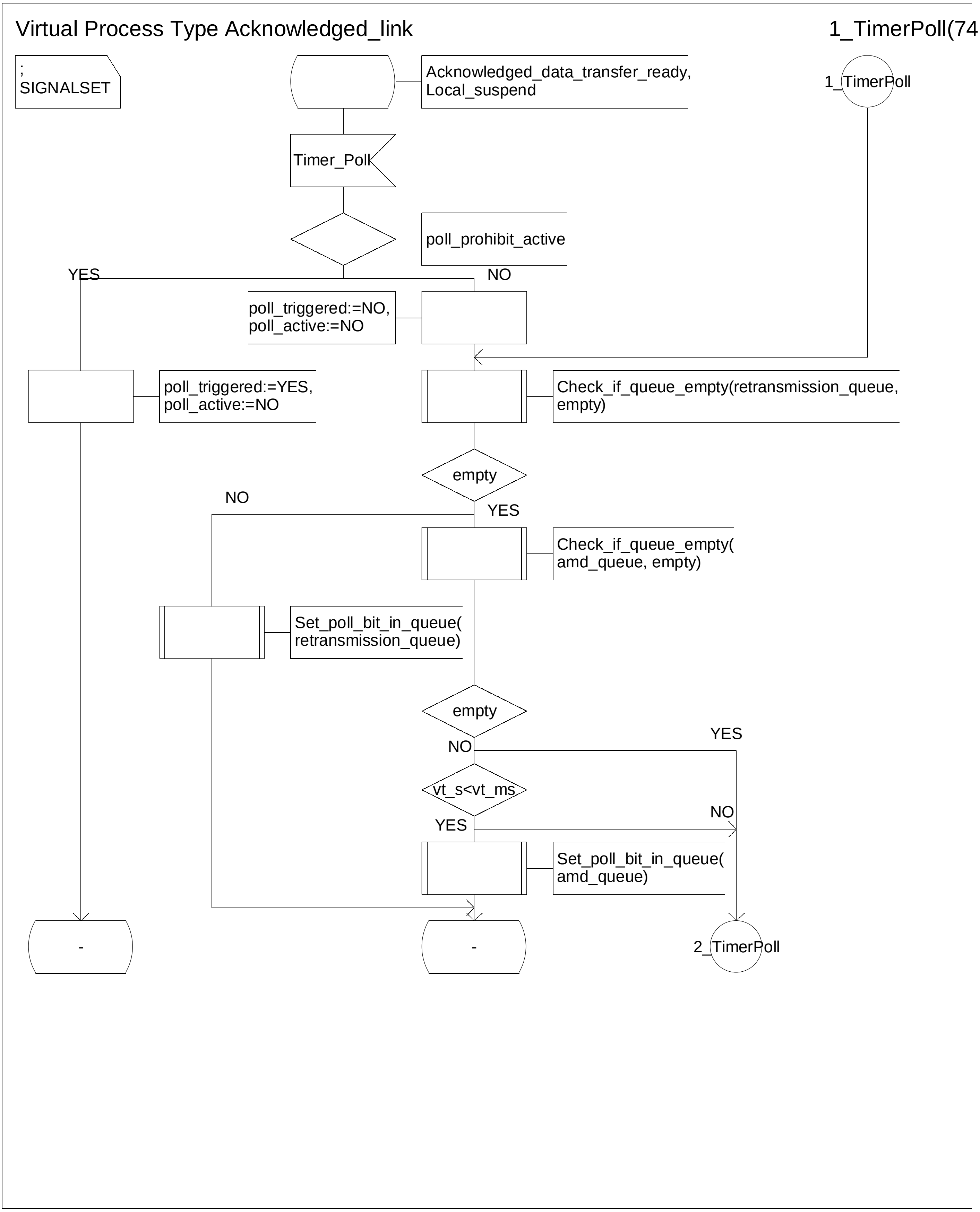

Virtual Process Type Acknowledged_link
2_StatusPduMrw(74
;
SIGNALSET
2_Mrw
Exists_in_receiver_queue(
vr_r, receiver_queue, exists)
exists
YES
NO
Remove_identified_from_queue(
receiver_queue, vr_r, amd_pu)
Place_in_receiving_side_queue(
assembly_queue, amd_pu)
Reassemble_am_pu(assembly_queue,
complete, sdus, n_sdu)
complete
YES
NO
j:=1
sdu:=sdus(j)
Rlc_AmData_ind(sdu)
VIA Am
j=n_sdu
NO
YES
j:=j+1
3_Mrw
vr_r:=vr_r+1,
vr_mr:=vr_mr+1


Virtual Process Type Acknowledged_link
3_StatusPduMrw(74
;
SIGNALSET
3_Mrw
3A_Mrw
i:=i+1
discard_n_fli:=NO
epc_active
YES
NO
The pdu_size is indicated by MAC.
status_triggered:=YES
Create_status(vr_r, vr_h, receive_window, pdu_size, receiver_queue, status_pdus, vr_ep, n_status, sn_mrw)
j:=1
tx_status_pdu:=status_pdus(j)
Place_piggyback_in_queue(amd_queue, retransmission_queue, tx_status_pdu, vt_ms, possible)
2_StatusPdu
4_Mrw

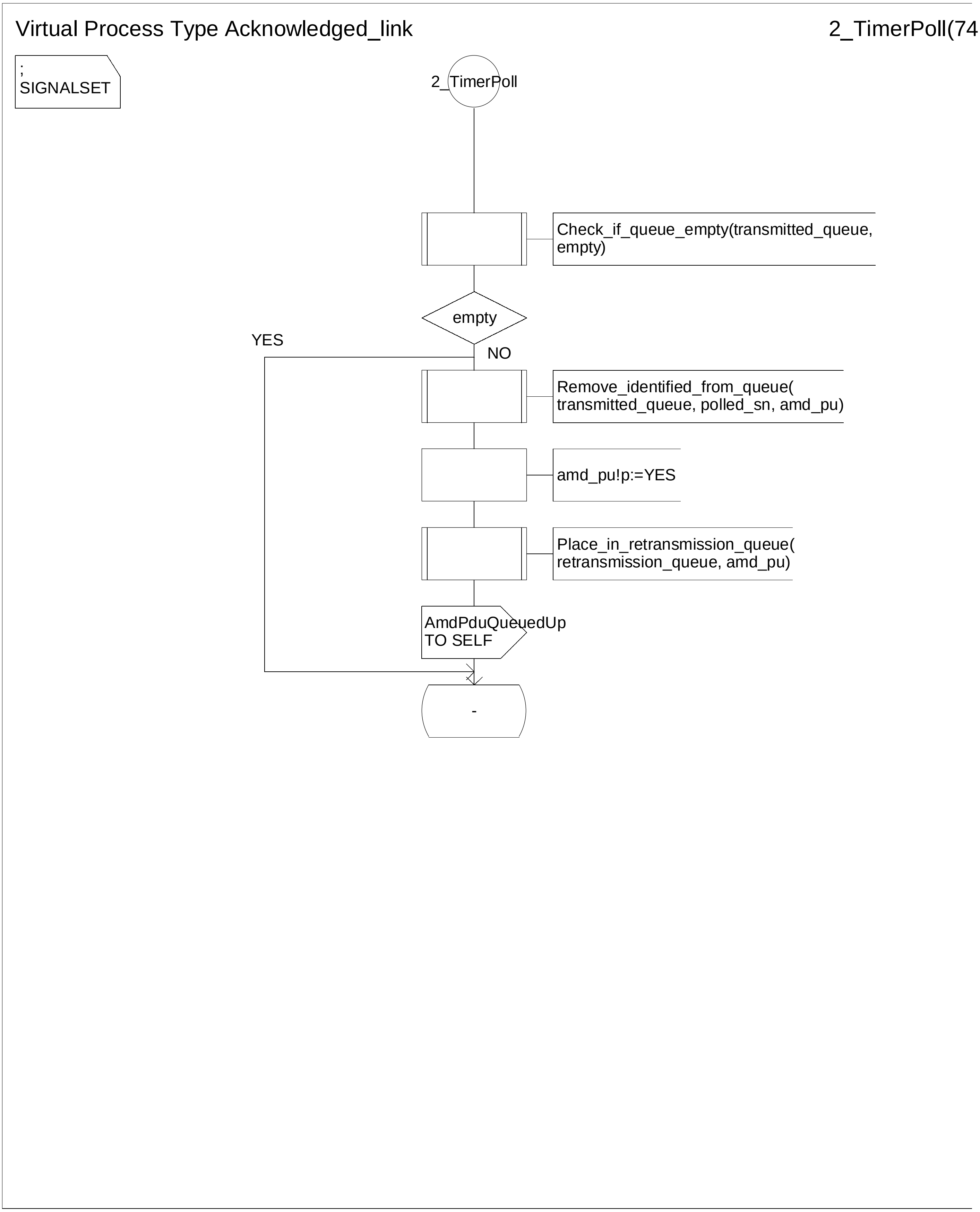

Virtual Process Type Acknowledged_link
4_StatusPduMrw(74
;
SIGNALSET
4_Mrw
possible
YES
NO
j=n_status
YES
NO
Transmit_status(tx_status_pdu, logical_channel)
NO
j=n_status
YES
j:=j+1
status_triggers(EPC)
YES
NO
Set(NOW+timer_durations!epc, Timer_EPC)
epc_active:=YES
status_triggers(STATUS_PROHIBIT)
YES
NO
Set(NOW+timer_durations!status_prohibit,
Timer_Status_prohibit)
status_prohibit_active:=YES
2_StatusPdu
3A_Mrw
2_StatusPdu

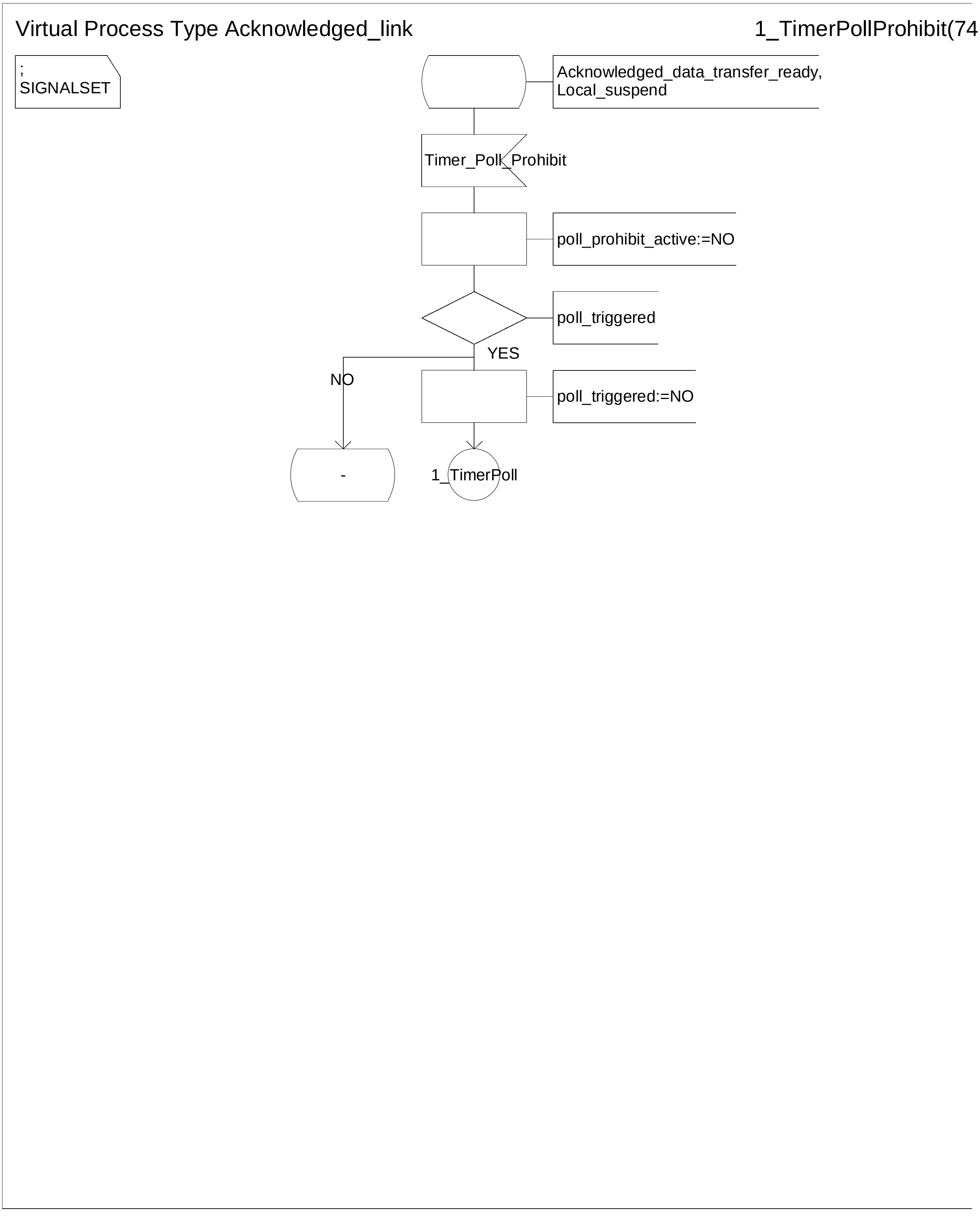

Virtual Process Type Acknowledged_link
3_StatusPdu(74
;
SIGNALSET
3_StatusPdu
sn_ack=0
NO
YES
Remove_acks_and_get_muis(
transmitted_queue,retransmission_queue,
sn_ack, tot_mui, muis)
tot_mui=0
NO
YES
j:=1
muis(j)!cnf
NO
YES
Rlc_AmData_conf(muis(j)!mui)
VIA Am
Reset(Timer_Discard(muis(j)!mui))
j=tot_mui
NO
YES
j:=j+1
mrw_active
NO
YES
Update_state_variables(vt_a, vt_ms,
transmit_window, amd_queue,
transmitted_queue, retransmission_queue)
vt_a=vt_s
YES
NO
poll_triggered:=NO
4_StatusPdu


Virtual Process Type Acknowledged_link
4_StatusPdu(74
;
SIGNALSET
4_StatusPdu
poll_active
NO
YES
Check_if_queue_empty(
retransmission_queue, empty)
empty
YES
NO
Check_if_queue_empty(
transmitted_queue, empty)
empty
NO
YES
Remove_identified_from_queue(
transmitted_queue, vt_s-1, amd_pu)
Place_in_retransmission_queue(
retransmission_queue, amd_pu)
AmdPduQueuedUp
TO SELF
-

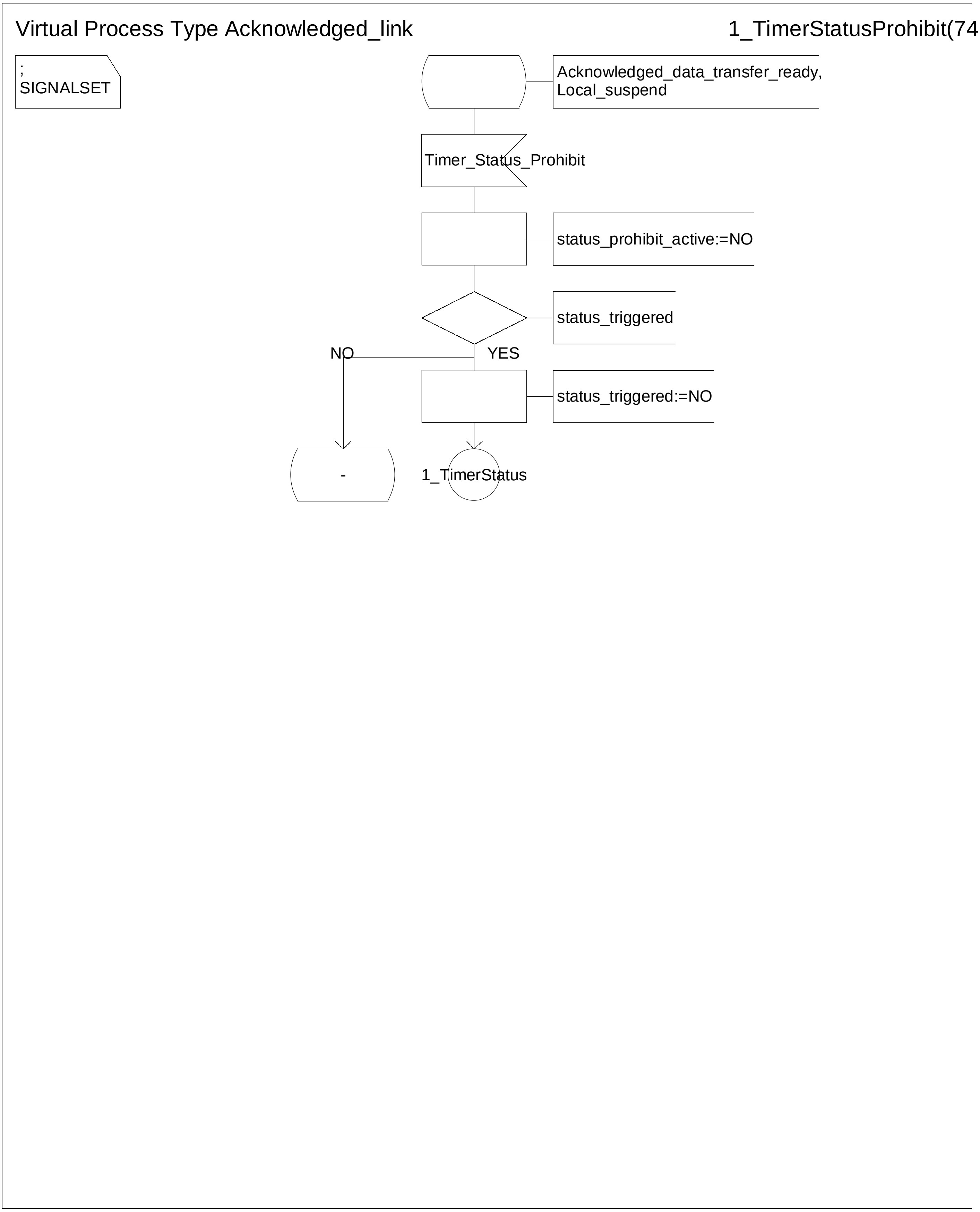


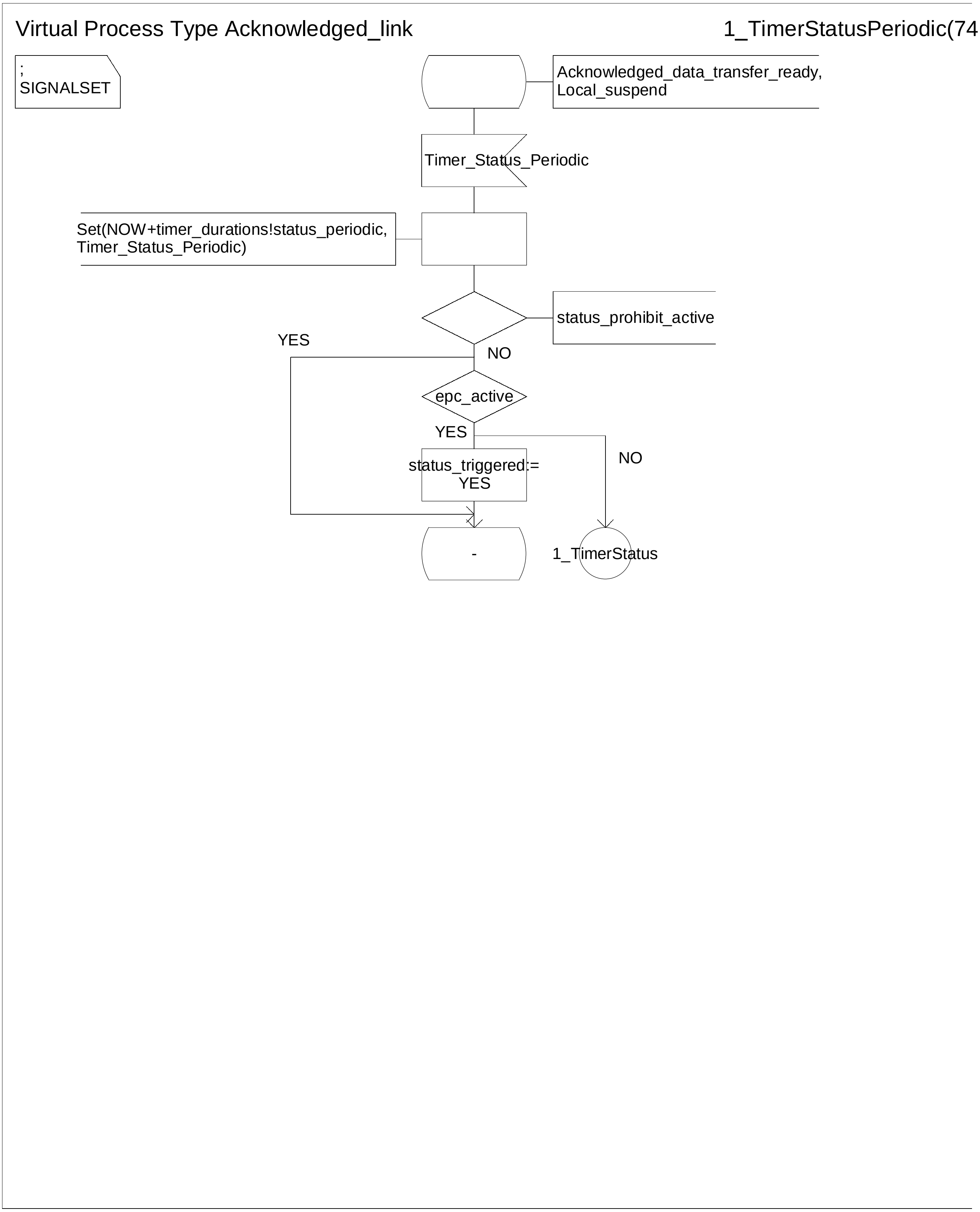
Virtual Process Type Acknowledged_link
1_TimerPoll(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
1_TimerPoll
Timer_Poll
poll_prohibit_active
YES
NO
poll_triggered:=NO,
poll_active:=NO
poll_triggered:=YES,
poll_active:=NO
Check_if_queue_empty(retransmission_queue,
empty)
empty
NO
YES
Check_if_queue_empty(
amd_queue, empty)
Set_poll_bit_in_queue(
retransmission_queue)
empty
YES
NO
vt_s<vt_ms
NO
YES
Set_poll_bit_in_queue(
amd_queue)
-
-
2_TimerPoll

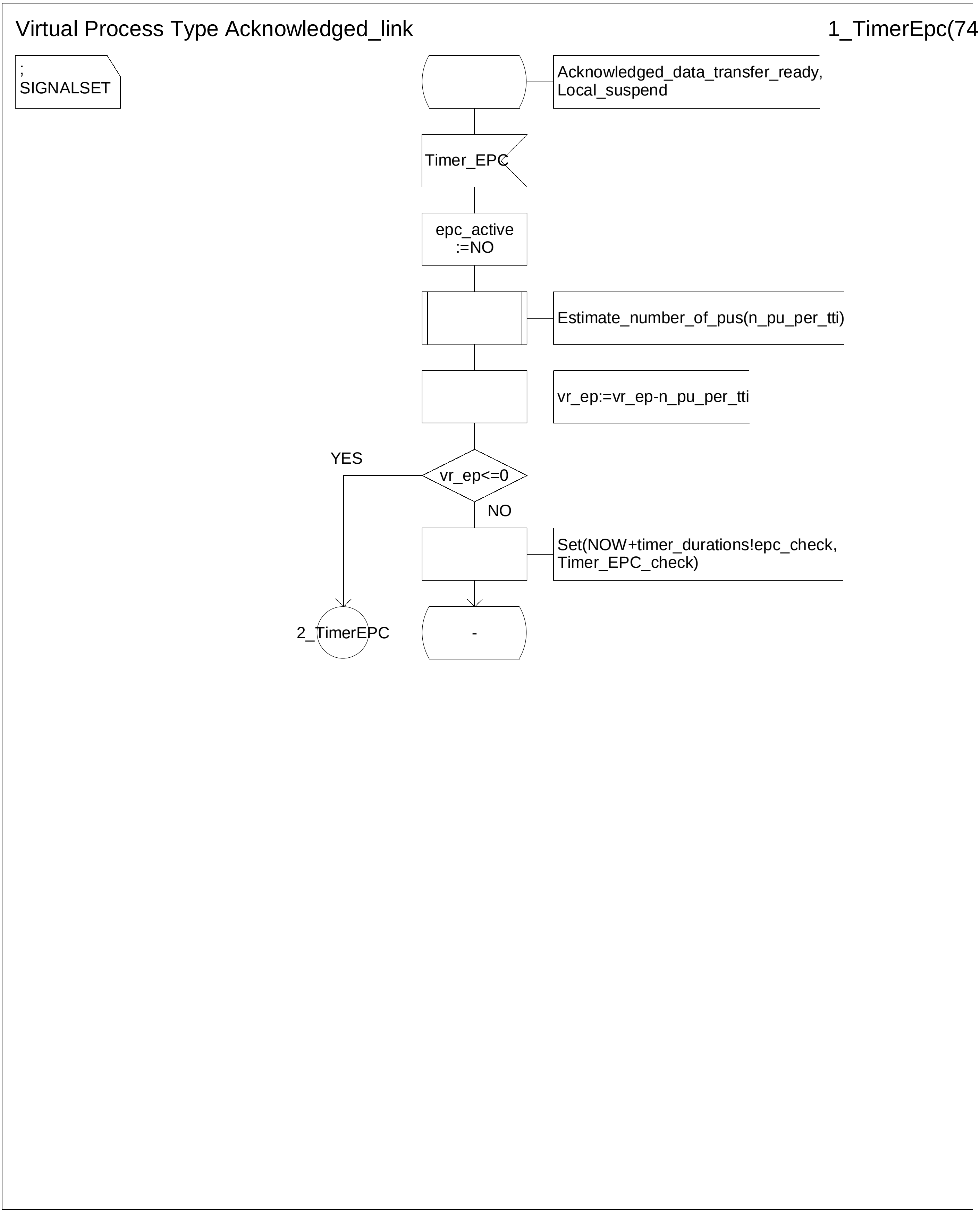

Virtual Process Type Acknowledged_link
2_TimerPoll(74
;
SIGNALSET
2_TimerPoll
Check_if_queue_empty(transmitted_queue, empty)
empty
YES
NO
Remove_identified_from_queue(transmitted_queue, polled_sn, amd_pu)
amd_pu!p:=YES
Place_in_retransmission_queue(retransmission_queue, amd_pu)
AmdPduQueuedUp TO SELF
-

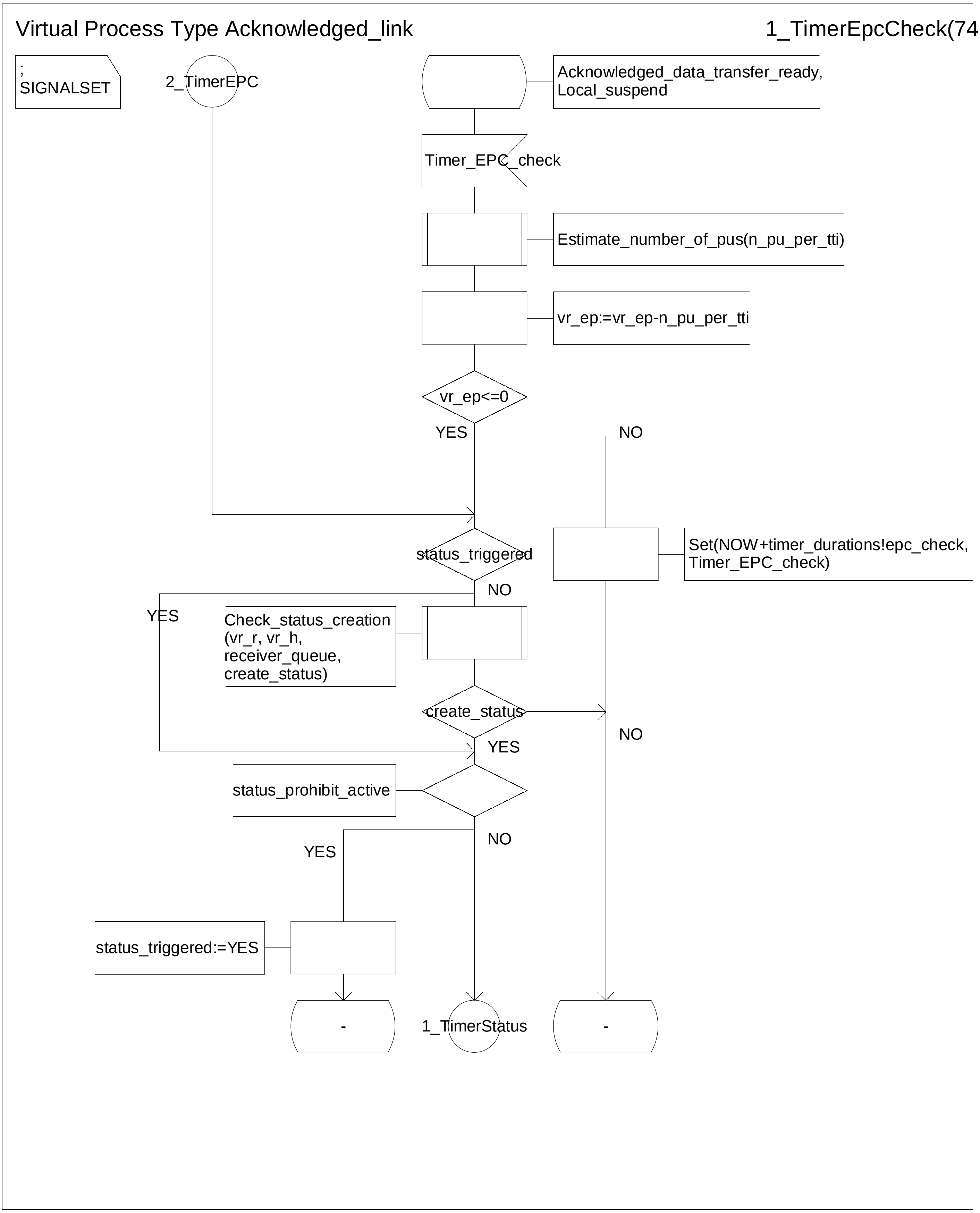


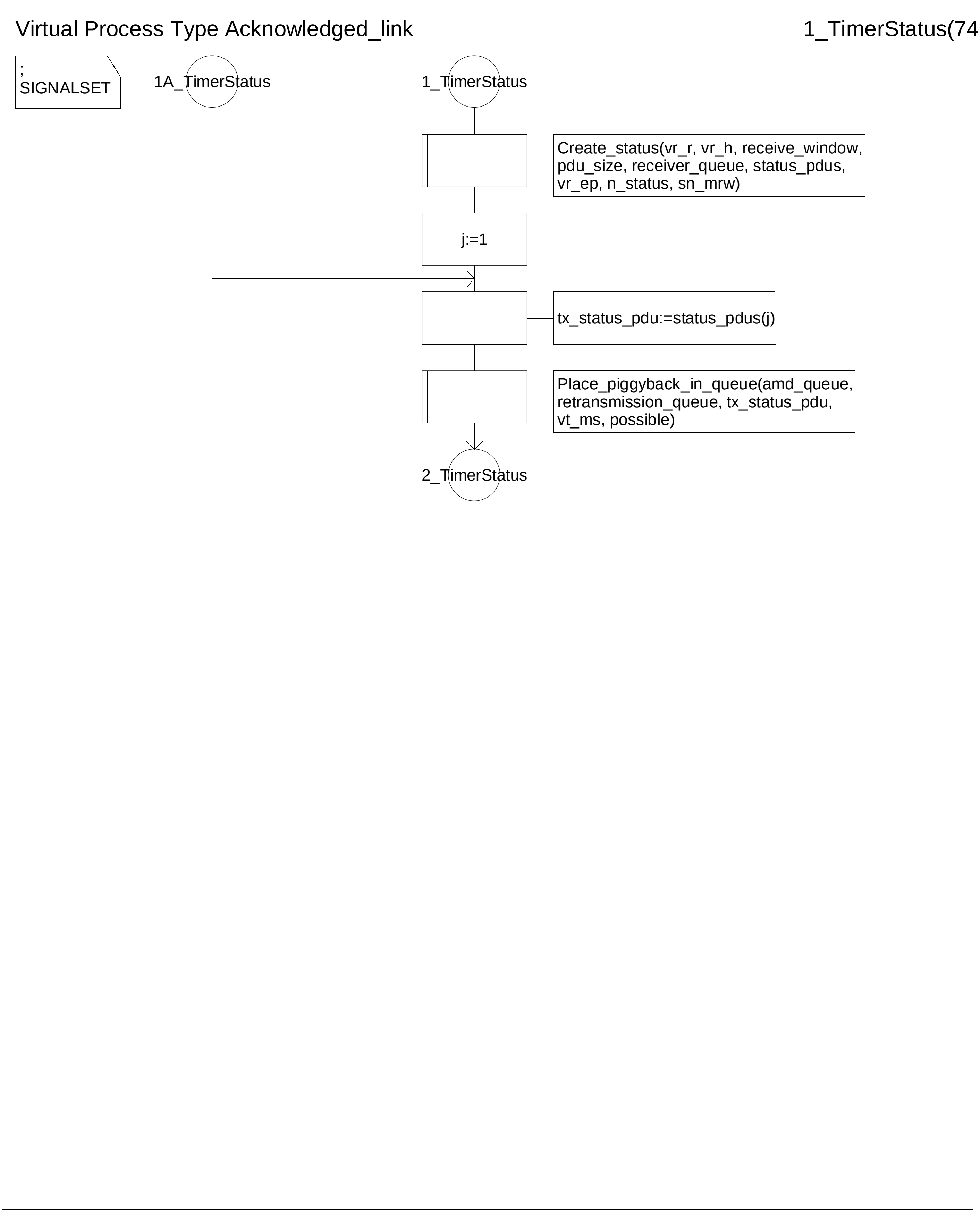
Virtual Process Type Acknowledged_link
1_TimerPollProhibit(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_Poll_Prohibit
poll_prohibit_active:=NO
poll_triggered
YES
NO
poll_triggered:=NO
-
1_TimerPoll

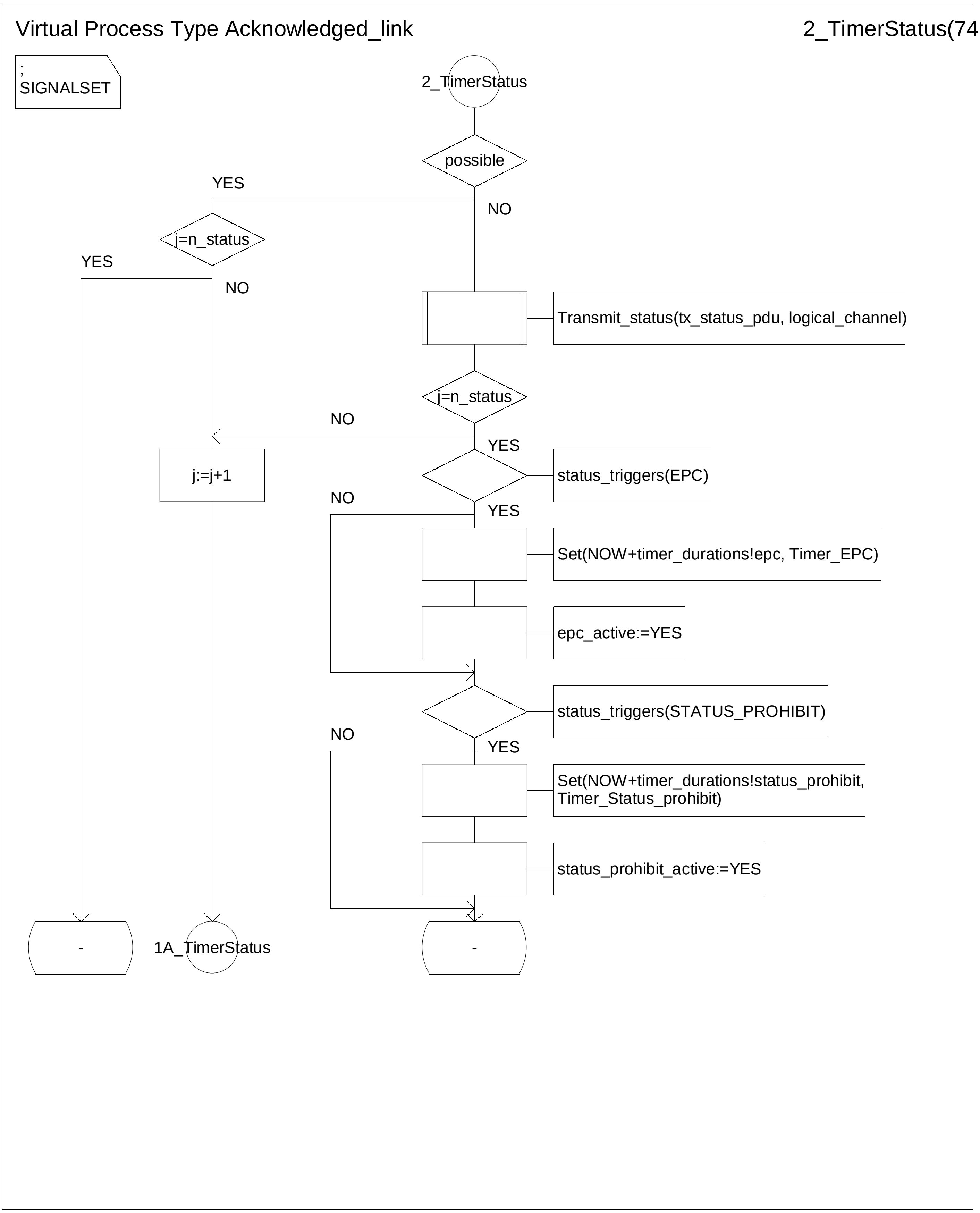

Virtual Process Type Acknowledged_link
1_TimerStatusProhibit(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_Status_Prohibit
status_prohibit_active:=NO
status_triggered
NO
YES
status_triggered:=NO
-
1_TimerStatus


Virtual Process Type Acknowledged_link
1_TimerStatusPeriodic(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_Status_Periodic
Set(NOW+timer_durations!status_periodic,
Timer_Status_Periodic)
status_prohibit_active
YES
NO
epc_active
YES
NO
status_triggered:=
YES
-
1_TimerStatus

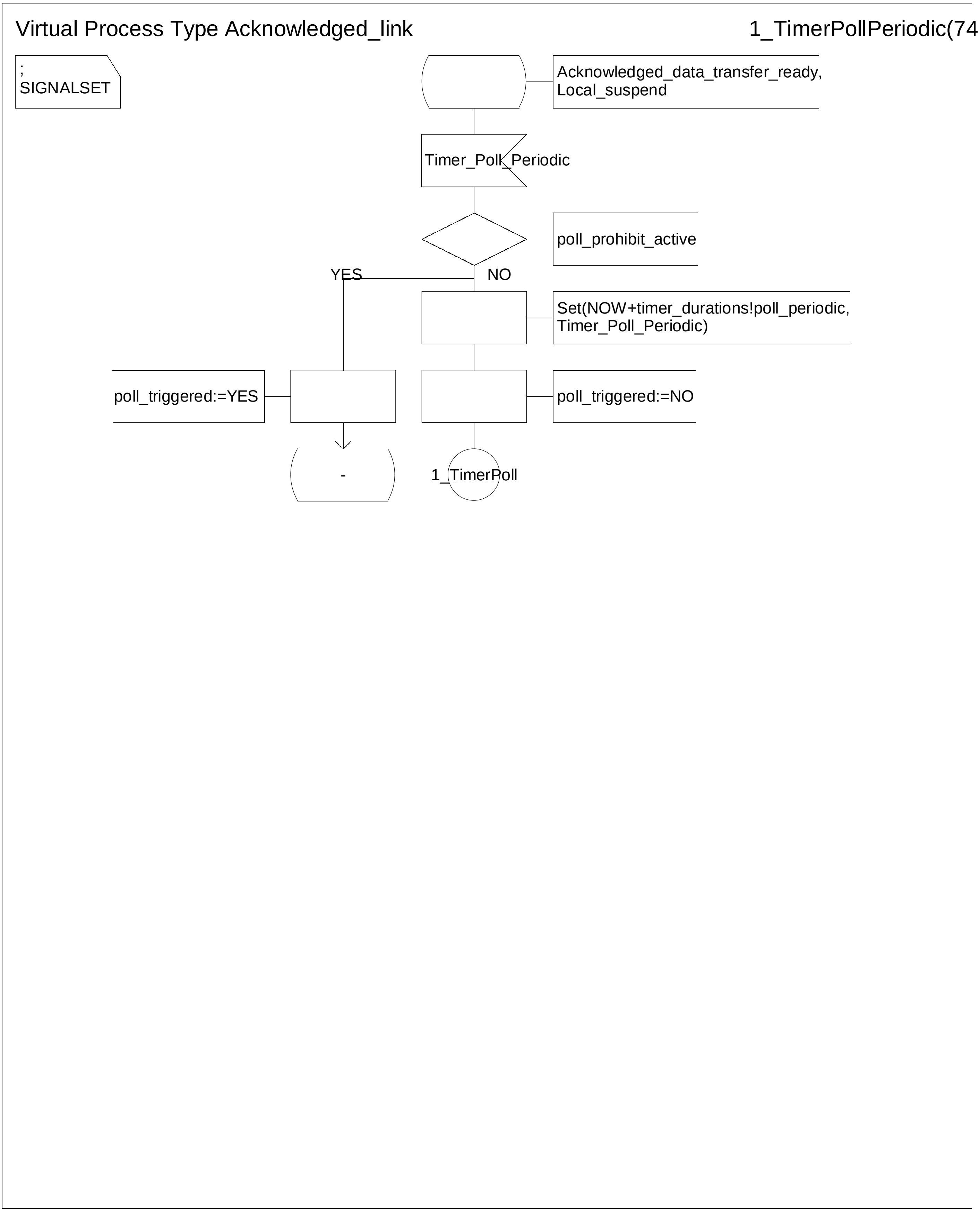

Virtual Process Type Acknowledged_link
1_TimerEpc(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_EPC
epc_active
:=NO
Estimate_number_of_pus(n_pu_per_tti)
vr_ep:=vr_ep-n_pu_per_tti
YES
vr_ep<=0
NO
Set(NOW+timer_durations!epc_check,
Timer_EPC_check)
2_TimerEPC
-

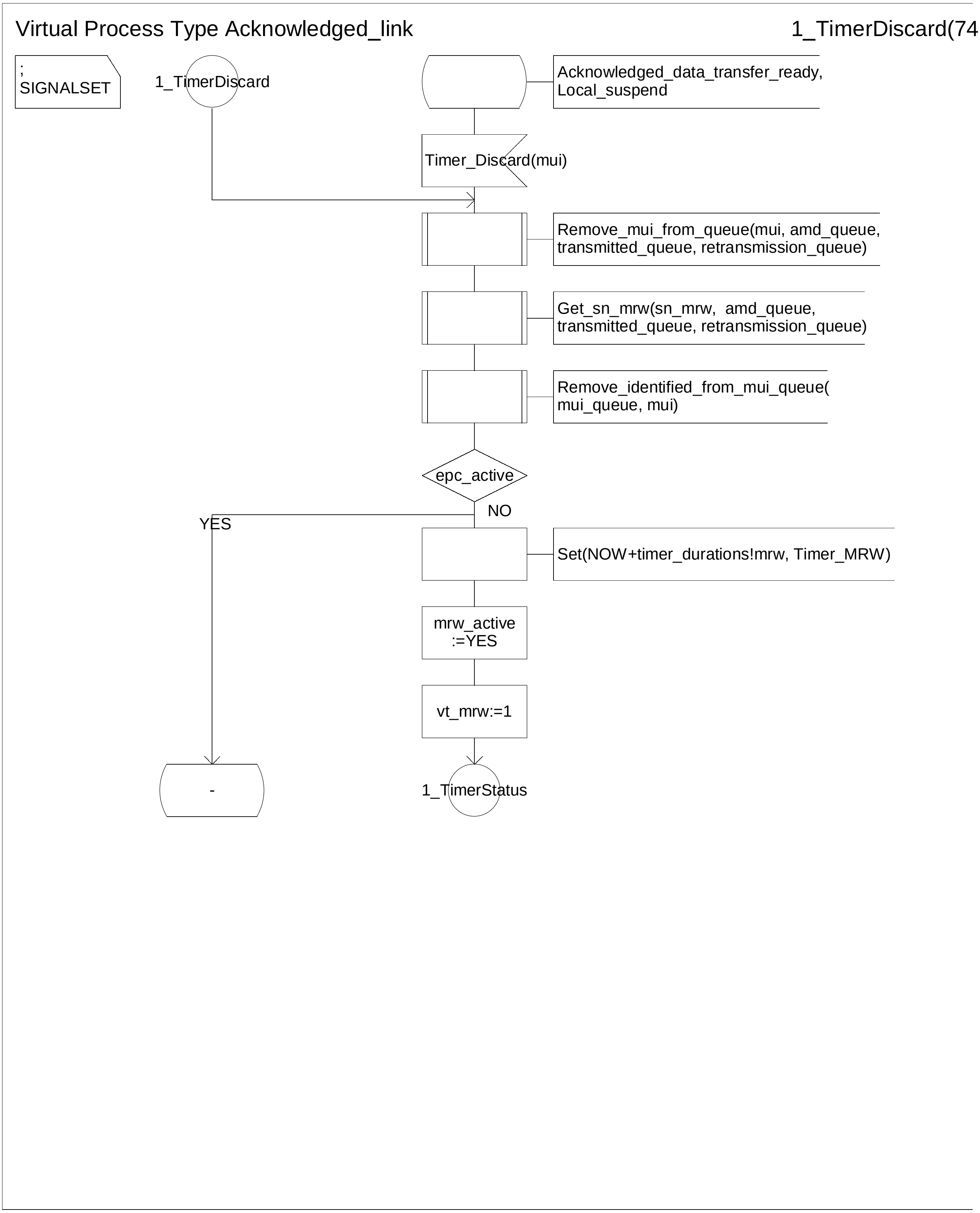

Virtual Process Type Acknowledged_link
1_TimerEpcCheck(74
;
SIGNALSET
2_TimerEPC
Acknowledged_data_transfer_ready,
Local_suspend
Timer_EPC_check
Estimate_number_of_pus(n_pu_per_tti)
vr_ep:=vr_ep-n_pu_per_tti
vr_ep<=0
YES
NO
status_triggered
Set(NOW+timer_durations!epc_check,
Timer_EPC_check)
NO
YES
Check_status_creation
(vr_r, vr_h,
receiver_queue,
create_status)
create_status
NO
YES
status_prohibit_active
NO
YES
status_triggered:=YES
-
1_TimerStatus
-


Virtual Process Type Acknowledged_link
1_TimerStatus(74
;
SIGNALSET
1A_TimerStatus
1_TimerStatus
Create_status(vr_r, vr_h, receive_window, pdu_size, receiver_queue, status_pdus, vr_ep, n_status, sn_mrw)
j:=1
tx_status_pdu:=status_pdus(j)
Place_piggyback_in_queue(amd_queue, retransmission_queue, tx_status_pdu, vt_ms, possible)
2_TimerStatus

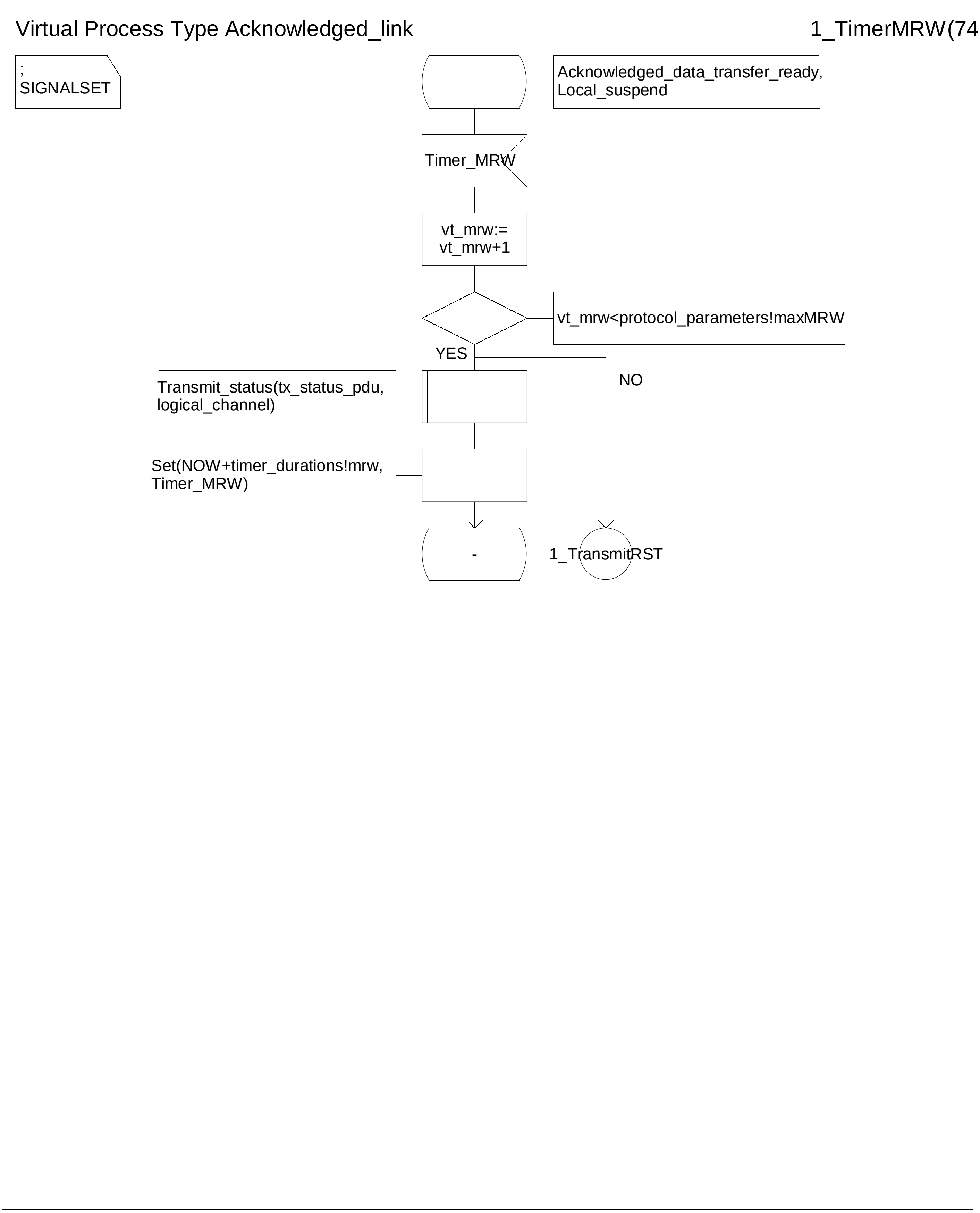


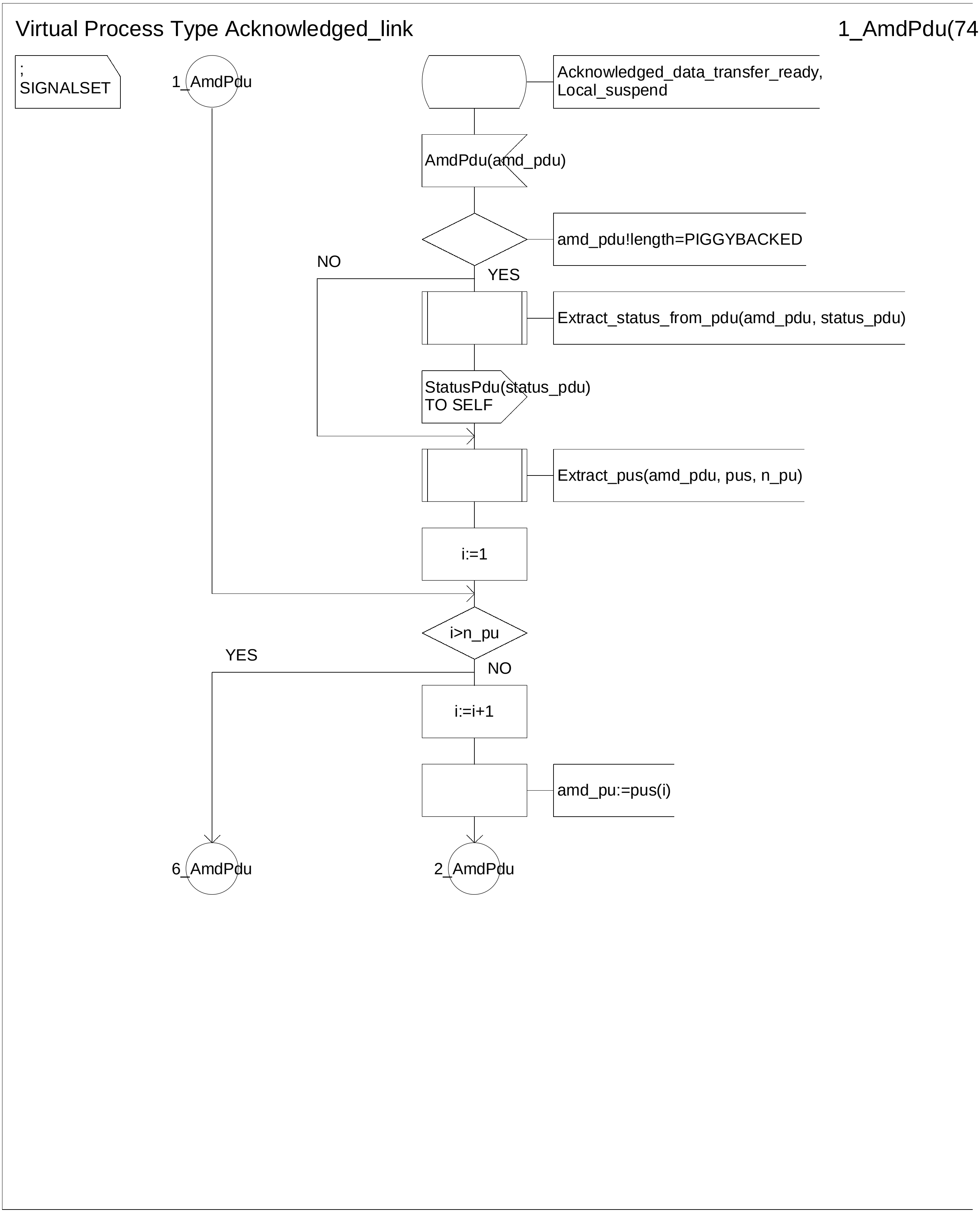
Virtual Process Type Acknowledged_link
2_TimerStatus(74
;
SIGNALSET
2_TimerStatus
possible
YES
NO
j=n_status
YES
NO
Transmit_status(tx_status_pdu, logical_channel)
j=n_status
NO
YES
j:=j+1
status_triggers(EPC)
NO
YES
Set(NOW+timer_durations!epc, Timer_EPC)
epc_active:=YES
status_triggers(STATUS_PROHIBIT)
NO
YES
Set(NOW+timer_durations!status_prohibit, Timer_Status_prohibit)
status_prohibit_active:=YES
-
1A_TimerStatus
-

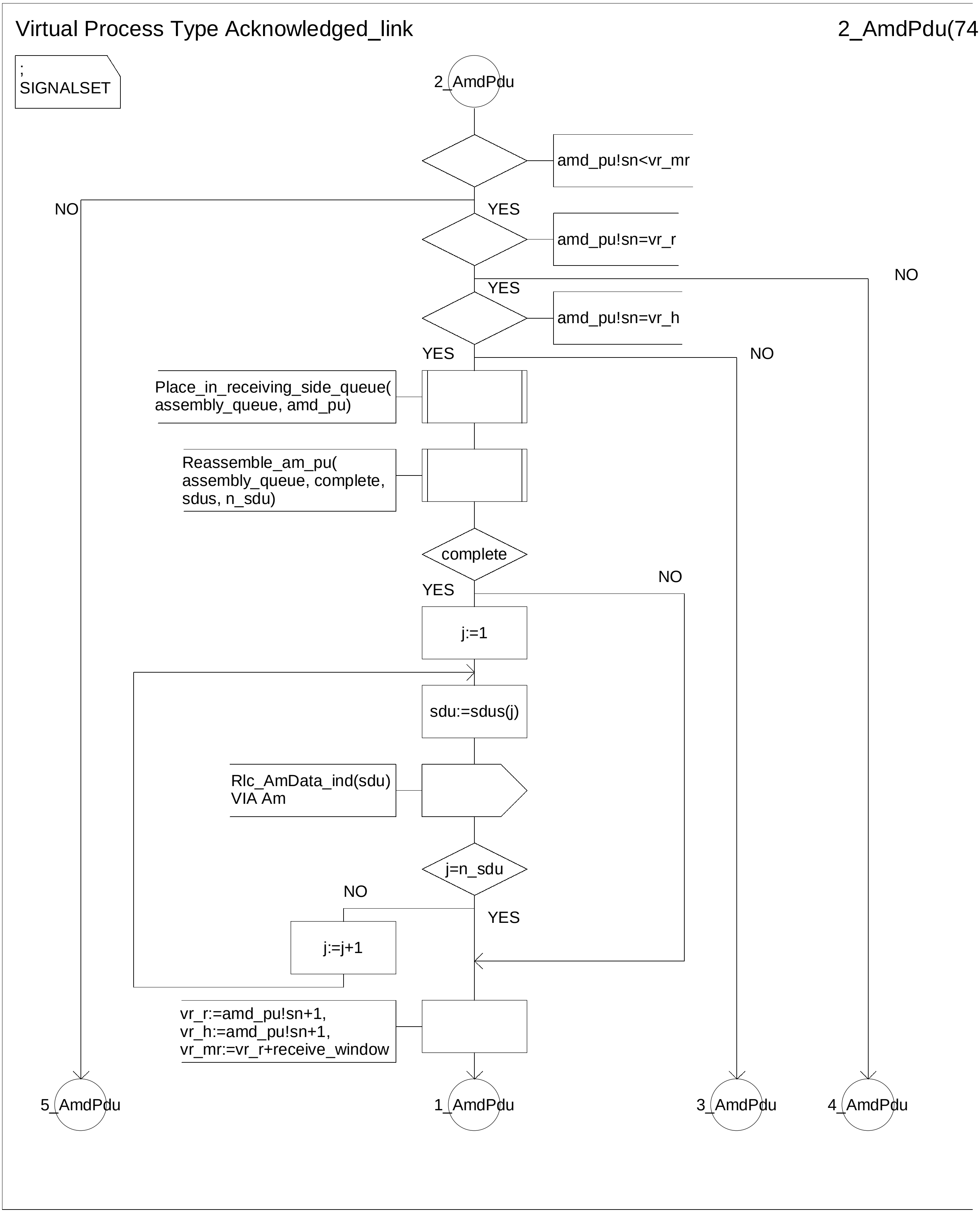

Virtual Process Type Acknowledged_link
1_TimerPollPeriodic(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_Poll_Periodic
poll_prohibit_active
YES
NO
Set(NOW+timer_durations!poll_periodic,
Timer_Poll_Periodic)
poll_triggered:=YES
poll_triggered:=NO
-
1_TimerPoll

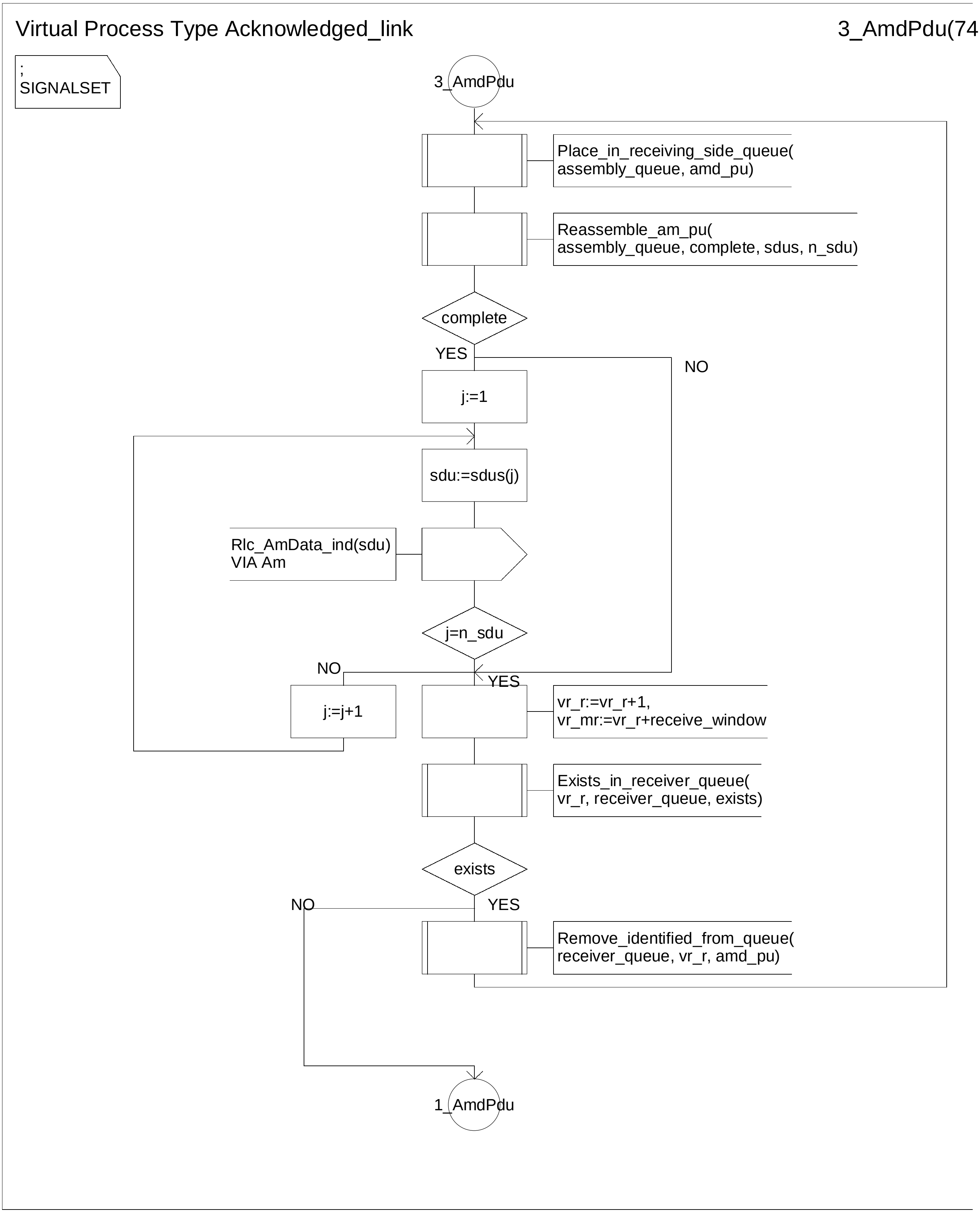


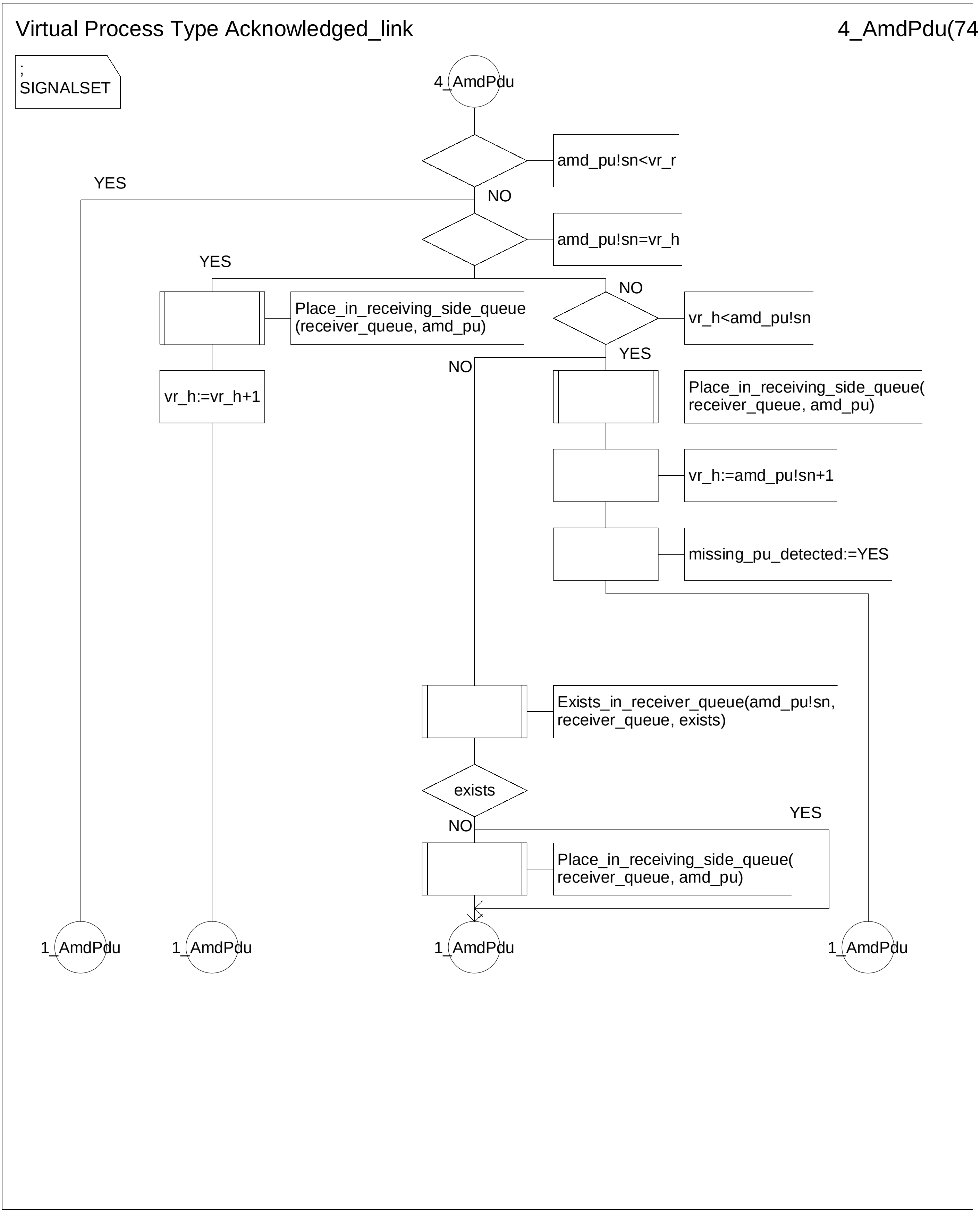
Virtual Process Type Acknowledged_link
1_TimerDiscard(74
;
SIGNALSET
1_TimerDiscard
Acknowledged_data_transfer_ready,
Local_suspend
Timer_Discard(mui)
Remove_mui_from_queue(mui, amd_queue,
transmitted_queue, retransmission_queue)
Get_sn_mrw(sn_mrw, amd_queue,
transmitted_queue, retransmission_queue)
Remove_identified_from_mui_queue(
mui_queue, mui)
epc_active
YES
NO
Set(NOW+timer_durations!mrw, Timer_MRW)
mrw_active
:=YES
vt_mrw:=1
-
1_TimerStatus

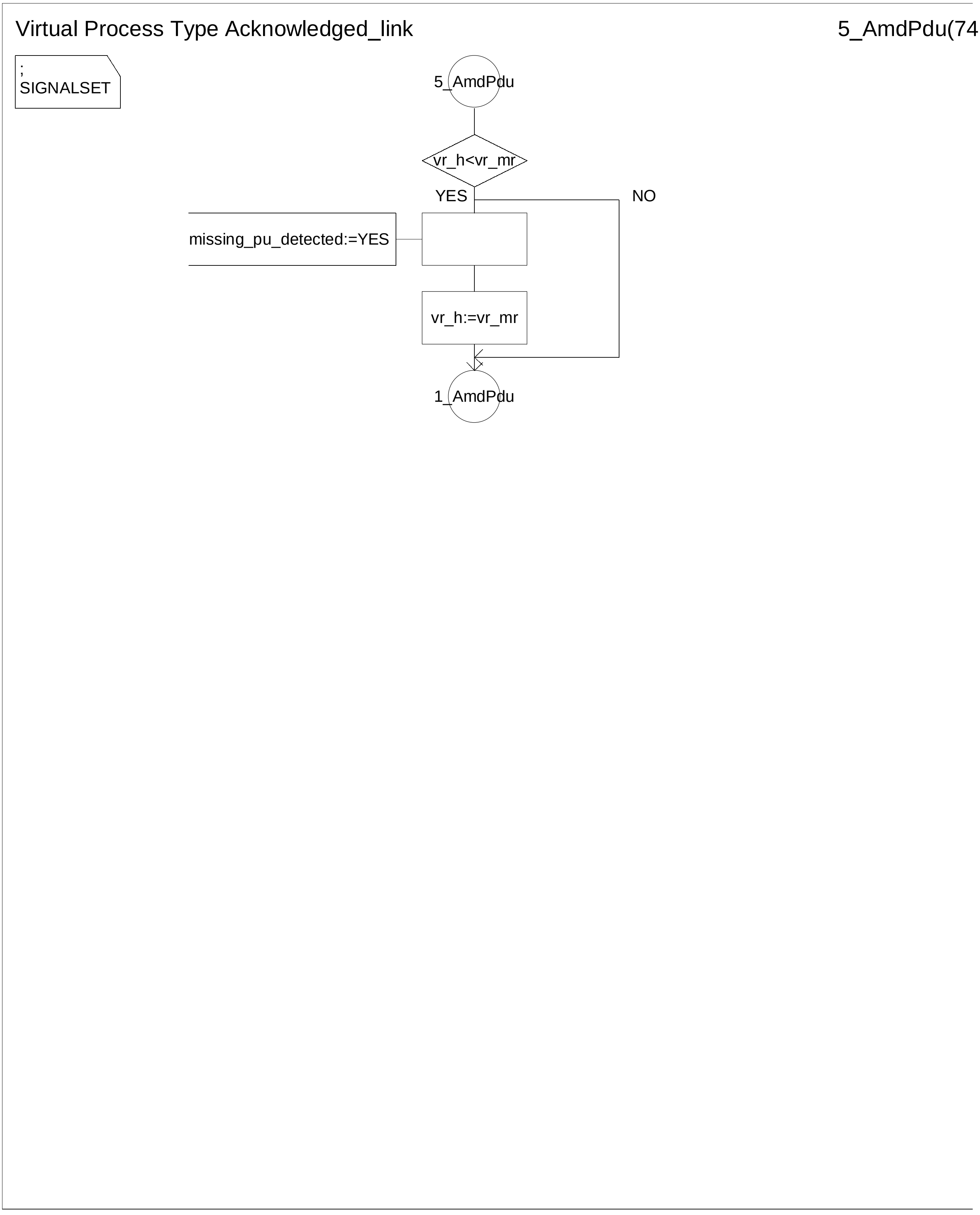

Virtual Process Type Acknowledged_link
1_TimerMRW(74
;
SIGNALSET
Acknowledged_data_transfer_ready,
Local_suspend
Timer_MRW
vt_mrw:=
vt_mrw+1
vt_mrw<protocol_parameters!maxMRW
YES
NO
Transmit_status(tx_status_pdu,
logical_channel)
Set(NOW+timer_durations!mrw,
Timer_MRW)
-
1_TransmitRST

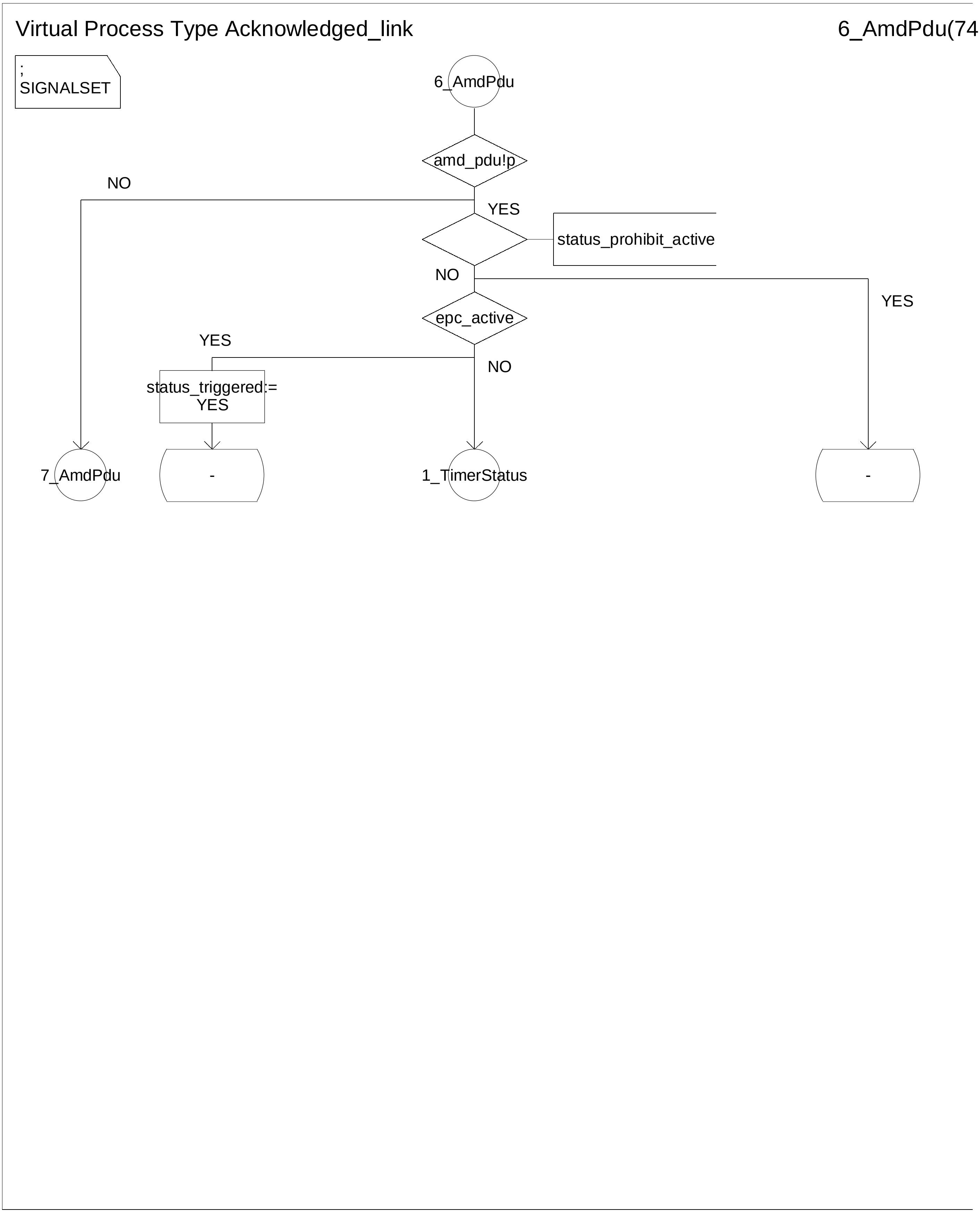

Virtual Process Type Acknowledged_link
1_AmdPdu(74
;
SIGNALSET
1_AmdPdu
Acknowledged_data_transfer_ready,
Local_suspend
AmdPdu(amd_pdu)
amd_pdu!length=PIGGYBACKED
NO
YES
Extract_status_from_pdu(amd_pdu, status_pdu)
StatusPdu(status_pdu)
TO SELF
Extract_pus(amd_pdu, pus, n_pu)
i:=1
i>n_pu
YES
NO
i:=i+1
amd_pu:=pus(i)
6_AmdPdu
2_AmdPdu

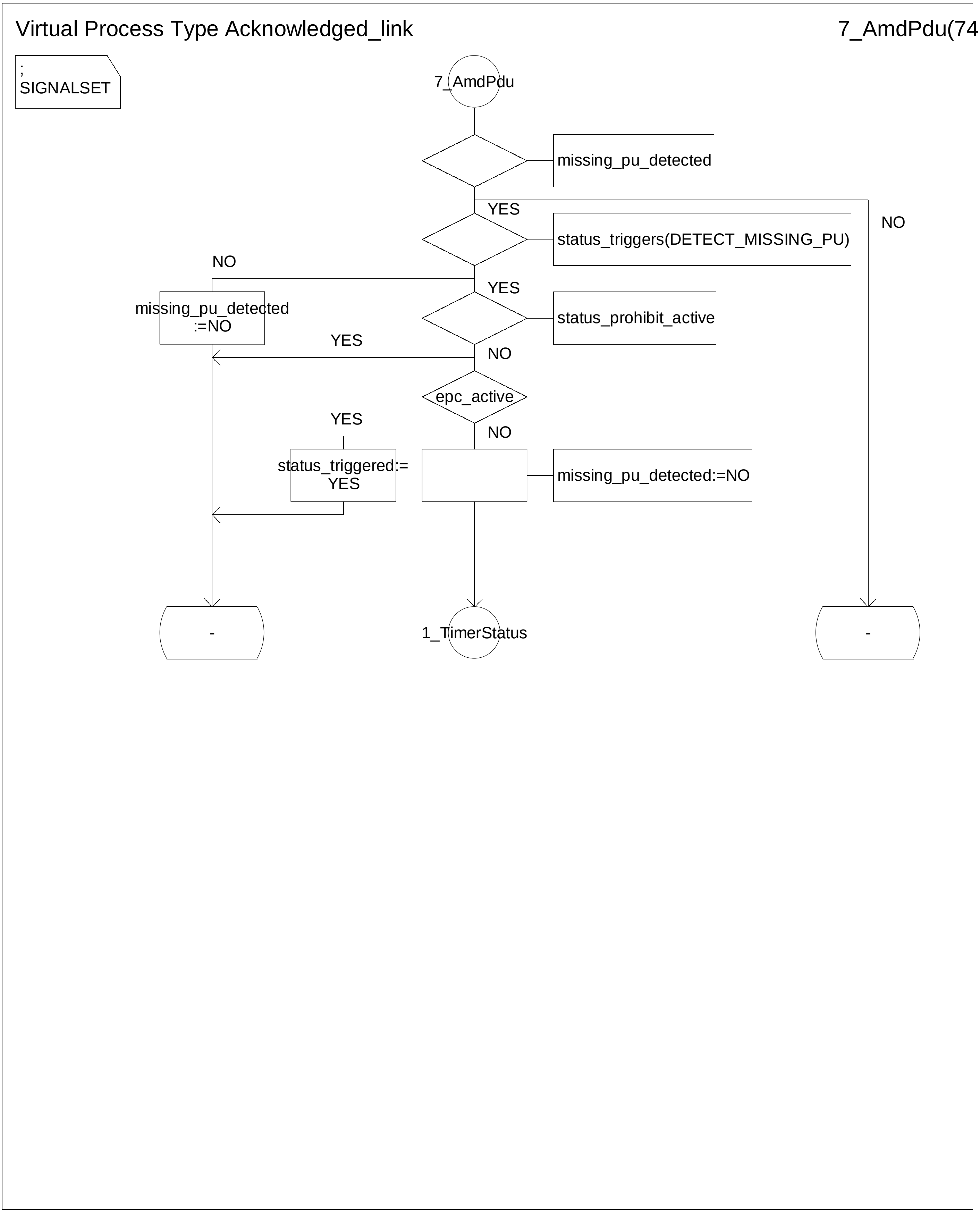

Virtual Process Type Acknowledged_link
2_AmdPdu(74
;
SIGNALSET
2_AmdPdu
amd_pu!sn<vr_mr
NO
YES
amd_pu!sn=vr_r
NO
YES
amd_pu!sn=vr_h
YES
NO
Place_in_receiving_side_queue(
assembly_queue, amd_pu)
Reassemble_am_pu(
assembly_queue, complete,
sdus, n_sdu)
complete
YES
NO
j:=1
sdu:=sdus(j)
Rlc_AmData_ind(sdu)
VIA Am
j=n_sdu
NO
YES
j:=j+1
vr_r:=amd_pu!sn+1,
vr_h:=amd_pu!sn+1,
vr_mr:=vr_r+receive_window
5_AmdPdu
1_AmdPdu
3_AmdPdu
4_AmdPdu

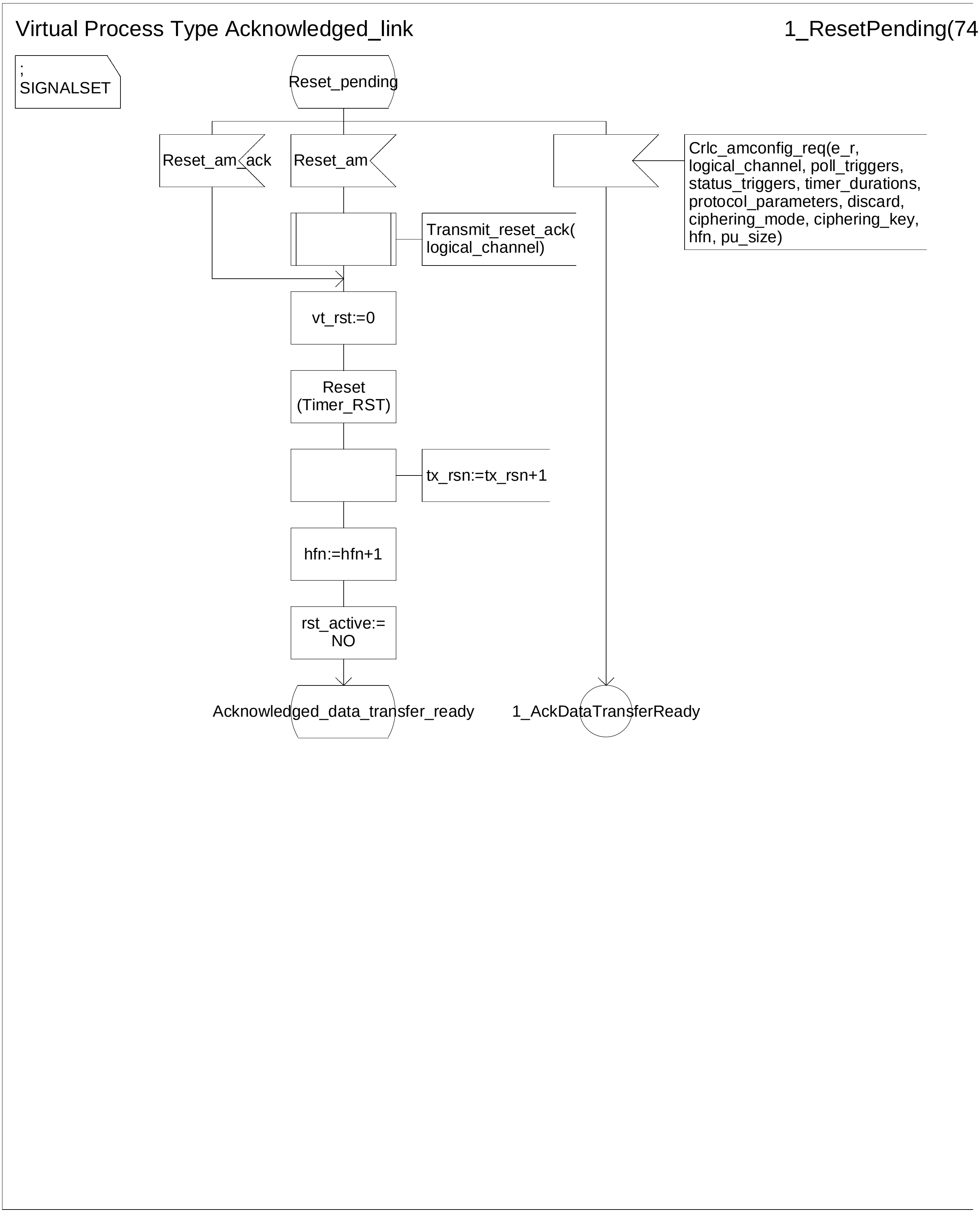

Virtual Process Type Acknowledged_link
3_AmdPdu(74
;
SIGNALSET
3_AmdPdu
Place_in_receiving_side_queue(
assembly_queue, amd_pu)
Reassemble_am_pu(
assembly_queue, complete, sdus, n_sdu)
complete
YES
NO
j:=1
sdu:=sdus(j)
Rlc_AmData_ind(sdu)
VIA Am
j=n_sdu
NO
YES
j:=j+1
vr_r:=vr_r+1,
vr_mr:=vr_r+receive_window
Exists_in_receiver_queue(
vr_r, receiver_queue, exists)
exists
NO
YES
Remove_identified_from_queue(
receiver_queue, vr_r, amd_pu)
1_AmdPdu

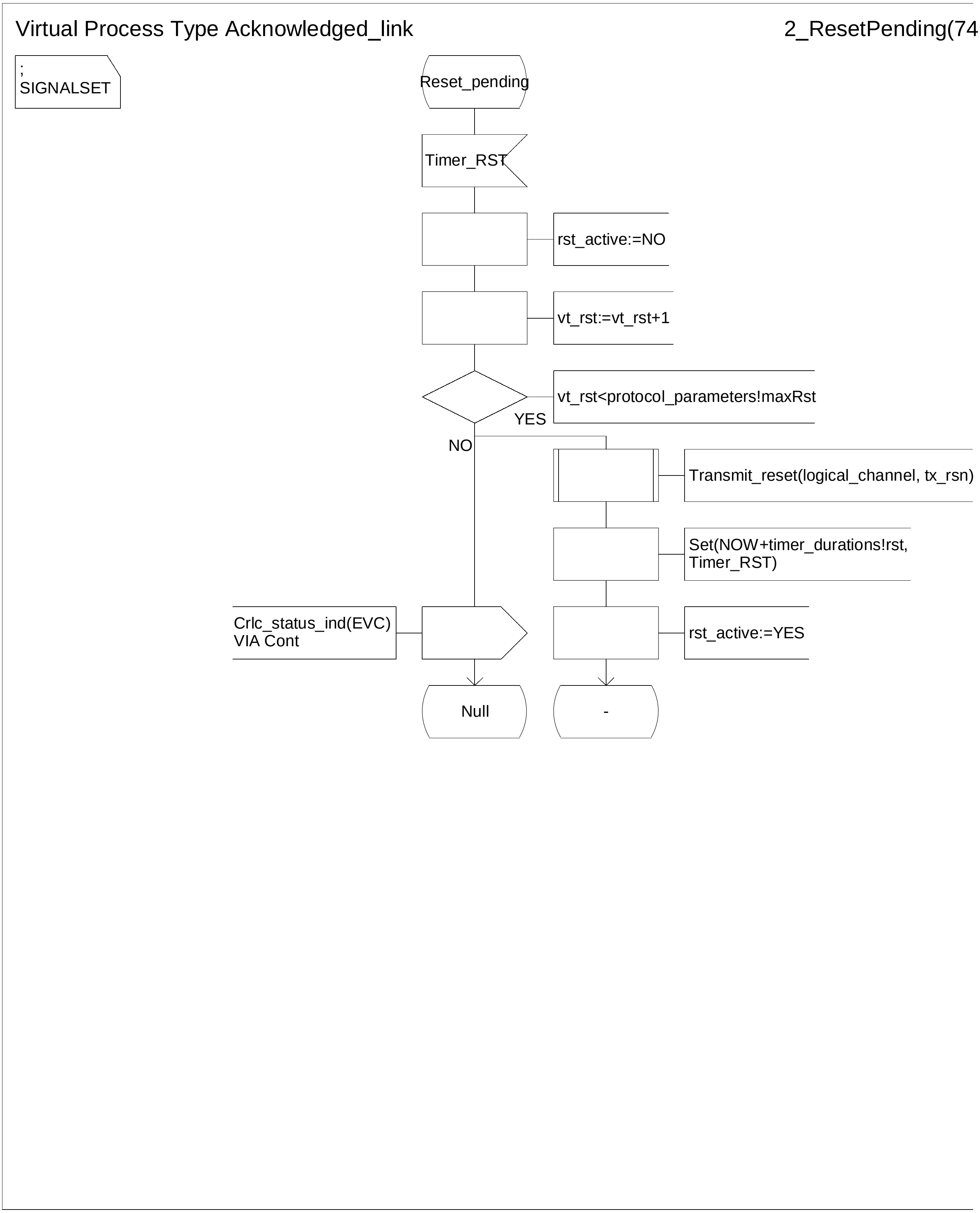

Virtual Process Type Acknowledged_link
4_AmdPdu(74
;
SIGNALSET
4_AmdPdu
amd_pu!sn<vr_r
YES
NO
amd_pu!sn=vr_h
YES
NO
Place_in_receiving_side_queue
(receiver_queue, amd_pu)
vr_h<amd_pu!sn
NO
YES
vr_h:=vr_h+1
Place_in_receiving_side_queue(
receiver_queue, amd_pu)
vr_h:=amd_pu!sn+1
missing_pu_detected:=YES
Exists_in_receiver_queue(amd_pu!sn,
receiver_queue, exists)
exists
NO
YES
Place_in_receiving_side_queue(
receiver_queue, amd_pu)
1_AmdPdu
1_AmdPdu
1_AmdPdu
1_AmdPdu


Virtual Process Type Acknowledged_link
5_AmdPdu(74
;
SIGNALSET
5_AmdPdu
vr_h<vr_mr
YES
NO
missing_pu_detected:=YES
vr_h:=vr_mr
1_AmdPdu


Virtual Process Type Acknowledged_link
6_AmdPdu(74
;
SIGNALSET
6_AmdPdu
amd_pdu!p
NO
YES
status_prohibit_active
NO
YES
epc_active
YES
NO
status_triggered:=
YES
7_AmdPdu
-
1_TimerStatus
-


Virtual Process Type Acknowledged_link
7_AmdPdu(74
;
SIGNALSET
7_AmdPdu
missing_pu_detected
YES
NO
status_triggers(DETECT_MISSING_PU)
NO
YES
missing_pu_detected
:=NO
status_prohibit_active
YES
NO
epc_active
YES
NO
status_triggered:=
YES
missing_pu_detected:=NO
-
1_TimerStatus
-


Virtual Process Type Acknowledged_link
1_ResetPending(74
;
SIGNALSET
Reset_pending
Reset_am_ack
Reset_am
Crlc_amconfig_req(e_r, logical_channel, poll_triggers, status_triggers, timer_durations, protocol_parameters, discard, ciphering_mode, ciphering_key, hfn, pu_size)
Transmit_reset_ack(logical_channel)
vt_rst:=0
Reset (Timer_RST)
tx_rsn:=tx_rsn+1
hfn:=hfn+1
rst_active:= NO
Acknowledged_data_transfer_ready
1_AckDataTransferReady


Virtual Process Type Acknowledged_link
2_ResetPending(74
;
SIGNALSET
Reset_pending
Timer_RST
rst_active:=NO
vt_rst:=vt_rst+1
vt_rst<protocol_parameters!maxRst
YES
NO
Transmit_reset(logical_channel, tx_rsn)
Set(NOW+timer_durations!rst,
Timer_RST)
Crlc_status_ind(EVC)
VIA Cont
rst_active:=YES
Null
-

Annex B (informative): Pseudo code describing AMD PDU header Compression

The following Pseudo-Code is an example of algorithm to describe the exact Header Compression Operation that takes place when several PUs are packed into one RLC PDU.

```
/* Prior to calling this procedure it must be checked that <pus_in_pdu> consecutive PU:s
 are to be transmitted (or there is padding in the end)*/

Compress_PDU (pus_in_pdu, pu_size) {

 li_addition = 0; // reset the variable that counts data in full pu:s

 Loop through pus_in_pdu {

 d_e_flag = E-flag for this PU;

 If (d_e_flag == FALSE) {

 Append PU data to PDU data; // complete PU is SDU-data
 li_addition += pu_size; // to be added to the next LI

 } else { // E-flag is TRUE, so LI-field(s) exist

 Previous E-flag in PDU = TRUE; // Either in PDU header or pdu_li_vector;

 j = 0; // reset LI-counter for this PU
 pu_data_size = 0; // reset data size counter for this PU

 Loop until (d_e_flag == FALSE) {

 d_li = next LI; // in octet j of PU;
 d_e_flag = next E_FLAG; // in octet j of PU;

 if (d_li is not PADDING) {

 pu_data_size += d_li; // to keep track of data segment size in this PU);
 d_li += li_addition; // to add data from previous PU:s to LI-value);
 li_addition = 0; // reset li_addition;

 }

 Append (d_li + d_e_flag) to pdu_li_vector;

 j++; // go to next li_octet, if d_e_flag is TRUE);

 } /* end-of-loop (exit when d_e_flag is TRUE) */

 Append pu_data_size segments starting from j to RLC-PDU data;

 } /* end-of e-flag == TRUE */

 } /* end-of loop through PU:s in PDU */

} /* end-of Compress_PDU */
```

Annex C (informative): Change history

Change history					
TSG-RAN#	Version	CR	Tdoc RAN	New Version	Subject/Comment
RAN_05	-	-	RP-99465	3.0.0	(10/99) Approved at TSG-RAN #5 and placed under Change Control
RAN_06	3.0.0	001	RP-99641	3.1.0	(12/99) RLC: Editorial corrections
RAN_06	3.0.0	002	RP-99641	3.1.0	Editorial changes on RLC protocol specification
RAN_06	3.0.0	003	RP-99643	3.1.0	MRW procedure
RAN_06	3.0.0	004	RP-99643	3.1.0	SDU Discard Functionality
RAN_06	3.0.0	005	RP-99643	3.1.0	Change in RLC control PDU format
RAN_06	3.0.0	006	RP-99642	3.1.0	Editorial corrections regarding CTCH
RAN_06	3.0.0	007	RP-99641	3.1.0	Updated RLC SDL
RAN_06	3.0.0	011	RP-99642	3.1.0	RLC Editorial Changes
RAN_06	3.0.0	013	RP-99642	3.1.0	Editorial Modification on RLC specification
RAN_06	3.0.0	014	RP-99641	3.1.0	Editorial changes
RAN_06	3.0.0	015	RP-99642	3.1.0	Change to one PU in a AMD PDU
RAN_06	3.0.0	016	RP-99643	3.1.0	Introduction of RLC suspend state
RAN_06	3.0.0	017	RP-99641	3.1.0	RLC editorial corrections
-	3.1.0	-	-	3.1.1	(01/00) Editorial corrections in title and Annex A (SDL)
-	3.1.1	-	-	3.1.2	(01/00) Correction of persistent error regarding SDL in Table of Contents
RAN_07	3.1.2	018	RP-000040	3.2.0	(03/00) RLC editorial changes
RAN_07	3.1.2	021	RP-000040	3.2.0	Corrections to RLC
RAN_07	3.1.2	025	RP-000040	3.2.0	Corrections to RLC
RAN_07	3.1.2	026	RP-000040	3.2.0	STATUS PDUs
RAN_07	3.1.2	027	RP-000040	3.2.0	Clarification of RLC AMD Model
RAN_07	3.1.2	028	RP-000040	3.2.0	Corrections to Timer_discard procedures
RAN_07	3.1.2	029	RP-000040	3.2.0	Segmentation of RLC SDUs
RAN_07	3.1.2	030	RP-000040	3.2.0	Modification of SDU discard to support virtual PDCP sequence numbers
RAN_07	3.1.2	031	RP-000040	3.2.0	Removal of SCCH
RAN_07	3.1.2	032	RP-000040	3.2.0	Updated RLC SDL
RAN_07	3.1.2	033	RP-000040	3.2.0	RLC Editorial Changes
RAN_07	3.1.2	034	RP-000040	3.2.0	Order of bit transmission for RLC PDUs
RAN_08	3.2.0	038	RP-000220	3.3.0	(06/00) Corrections to RLC
RAN_08	3.2.0	039	RP-000220	3.3.0	Correction to the description of the MRW SUFI fields
RAN_08	3.2.0	040	RP-000220	3.3.0	Editorial corrections to length indicators and local suspend rate
RAN_08	3.2.0	041	RP-000220	3.3.0	Clarification of the RESET PDU
RAN_08	3.2.0	043	RP-000220	3.3.0	Clarification of RLC/MAC interaction
RAN_08	3.2.0	044	RP-000220	3.3.0	General RLC corrections
RAN_08	3.2.0	045	RP-000220	3.3.0	Clarification of RLC Transparent Mode operation
RAN_08	3.2.0	048	RP-000220	3.3.0	Editorial corrections to abbreviations, SCCH, BCCH
RAN_08	3.2.0	052	RP-000220	3.3.0	Updated RLC SDL
RAN_08	3.2.0	053	RP-000220	3.3.0	Correction to RLC
RAN_08	3.2.0	055	RP-000220	3.3.0	RLC Logical Channel mapping
RAN_08	3.2.0	057	RP-000220	3.3.0	Correction of EPC timer mechanism
RAN_09	3.3.0	059	RP-000358	3.4.0	(09/00) State variables after window change
RAN_09	3.3.0	060	RP-000358	3.4.0	SDU discard
RAN_09	3.3.0	061	RP-000358	3.4.0	General RLC corrections
RAN_09	3.3.0	066	RP-000358	3.4.0	Editorial changes to RLC
RAN_09	3.3.0	067	RP-000358	3.4.0	Correction to RLC window size range
RAN_09	3.3.0	068	RP-000358	3.4.0	Window based polling
RAN_09	3.3.0	070	RP-000358	3.4.0	General corrections to RLC
RAN_09	3.3.0	071	RP-000358	3.4.0	State Transition in RLC Acknowledged Mode
RAN_09	3.3.0	073	RP-000358	3.4.0	Clarification of the Length Indicators
RAN_09	3.3.0	076	RP-000358	3.4.0	RLC corrections
RAN_09	3.3.0	077	RP-000358	3.4.0	Corrections to reset procedure and length indicator definitions
RAN_09	3.3.0	078	RP-000358	3.4.0	RLC Modes for SHCCH
RAN_09	3.3.0	079	RP-000358	3.4.0	CCCH in UM RLC

History

Document history		
V3.1.2	January 2000	Publication
V3.2.0	March 2000	Publication
V3.3.0	June 2000	Publication
V3.4.0	September 2000	Publication