TSG-RAN Meeting #6
Nice, France, 13 – 15 December 1999

TSGRP#6(99)641

Title: **Agreed CRs of category "D" (Editorial) to TS 25.322**

Source: **TSG-RAN WG2**

Agenda item: **5.2.3**

Doc #	Status-	Spec	CR	Rev	Subject	Cat	Versio	Versio
R2-99f04	agreed	25.322	001		RLC: Editorial corrections	D	3.0.0	3.1.0
R2-99i00	agreed	25.322	002	1	Editorial changes on RLC protocol	D	3.0.0	3.1.0
R2-99i01	agreed	25.322	007		Updated RLC SDL	D	3.0.0	3.1.0
R2-99k23	agreed	25.322	014		Editorial changes	D	3.0.0	3.1.0
R2-99k71	agreed	25.322	017	1	RLC editorial corrections	D	3.0.0	3.1.0

3GPP TSG-RAN Meeting #6
Nice, France, 13-15 December 1999

Document (R2-99f04)
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.322	CR	001	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN#6	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 28/11/1999

Subject: RLC: Editorial corrections

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction	[ ]	**Release:**	Phase 2	[ ]
A	Corresponds to a correction in an earlier release	[ ]		Release 96	[ ]
B	Addition of feature	[ ]		Release 97	[ ]
C	Functional modification of feature	[ ]		Release 98	[ ]
D	Editorial modification	[X]		Release 99	[X]
				Release 00	[ ]

Reason for change: obvious editorial mistakes are removed and obsolete parts are removed

Clauses affected: 4.2.1.2, 7, 9.2.2.3, 9.2.2.7, 9.2.2.8, 9.2.2.9, 9.3.3.3, 9.7.2, 9.7.3.2, 9.7.4, 11.2.2, 11.5.2,

Other specs affected:

Other 3G core specifications	[ ]	→ List of CRs:
Other GSM core specifications	[ ]	→ List of CRs:
MS test specifications	[ ]	→ List of CRs:
BSS test specifications	[ ]	→ List of CRs:
O&M specifications	[ ]	→ List of CRs:

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

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. ~~If the SDU is very large it is segmented into~~ 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, a SHCCH (downlink only) or a DTCH. The CCCH 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 (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. ~~After that t~~The RLC SDUs are delivered to the higher layer.

7 Services expected from MAC

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

- Data transfer~~;~~

9.2.2.3 Sequence Number (SN)

This field indicates the sequence number of the payload unit. ~~In a normal AMD PDU~~ If header compression is applied the sequence number of the first PU in the PDU is indicated. ~~If the PUs are not in sequence,~~Otherwise a sequence number is indicated separately for each PU in the extended header.

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

9.2.2.7 Header Extension Type (HE)

Length: 2 bits

This two-bit field indicates the format of the extended header.

Value	Description
00	The succeeding octets contains data
01	The succeeding octet contains a 7bit length indicator and E bit
10	The succeeding ~~octet~~field contains an extended header field
11	The succeeding two octets contain~~s~~ a 15bit length indicator and E bit

9.2.2.8 Length Indicator (LI)

This field is optional and is used if concatenation, padding or a piggybacked STATUS PDU takes place in a PU. It indicates the end of the last segment of a SDU. It points out the end of a segment by giving the number of octets between the end of the header fields (including the length indicator fields) and the end of the segment. The size of the Length Indicator may be either 7bits or 15bits. If the last segment of a SDU do not completely fill a PU either padding or a piggybacked STATUS PDU can be added. Predefined values of the length indicator are used to indicate this. The padding/piggybacked STATUS PDU predefined length indicators shall be added after the length indicator that indicates the end of the last SDU segment in the PU. The values that are reserved for special purposes are listed in the tables below depending on the size of the Length Indicator.

If a length indicator that indicates padding/piggybacked STATUS PDU refers to the last PU in the PDU it implicitly means that the rest of the PDU contains padding/piggybacked STATUS PDU. If the last PU in a PDU does not include padding or piggybacked STATUS PDU, but the PDU includes padding or a piggybacked STATUS PDU, an extra length indicator field shall be added as a normal length indicator to the last PU. This extra length indicator shall indicate either padding or a piggybacked STATUS PDU and shall be placed as the last length indicator in the PDU. The space needed for this length indicator shall not be taken from the data part in the PU, but from the padding or piggybacked STATUS PDU in the PDU. The receiving entity shall discard this length indicator.

If RLC PDUs always carry only one PU, 7bit indicators are used in a particular RLC PDU if the address space is sufficient to indicate all SDU segment borders. Otherwise 15bit Length Indicators are applied.

If RLC PDUs may carry more than one PUs the length of the Length Indicator only depends on the size of the largest RLC PDU and the size of the Length Indicator is always the same for all PUs.

Only one size of Length Indicators is used in one RLC PDU.

Length: 7bit

Bit	Description
0000000	The previous RLC PU was exactly filled with the last segment of a RLC SDU.
1111110	The rest part of the RLC PU or RLC PDU includes a piggybacked STATUS PDU.
1111111	The rest part of the RLC PU or RLC PDU is padding.

Length: 15bit

Bit	Description
000000000000000	The previous RLC PU was exactly filled with the last segment of a RLC SDU.
111111111111110	The rest part of the RLC PU or RLC PDU includes a piggybacked STATUS PDU.
111111111111111	The rest part of the RLC PU or RLC PDU is padding.

9.2.2.9 Data

RLC SDUs are mapped to this field. ~~If a~~ RLC SDU~~s is too large to fit into the data field it is~~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.

9.3.3.3 *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, the RLC entity resets the protocol and enters the acknowledged data transfer ready state.

Upon reception of a RESET PDU, the RLC entity resets the protocol, send a RESET ACK PDU and enters the acknowledged data transfer ready state.

Figure 9-18: The state model for the acknowledged mode entities

9.7.2 STATUS PDU transmission for acknowledged mode

The receiver of AMD PDUs transmits 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 PDU. 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 PDU 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 PDU to the sender.

2) Timer based STATUS PDU transfer

The receiver transmits a STATUS PDU periodically to the sender. The time period is controlled by the timer Timer_Status_Periodic.

3) The EPC mechanism

The EPC is started when a STATUS PDU is transmitted to the peer entity. If not all PUs requested for retransmission have been received before the EPC has expired a new STATUS PDU is transmitted to the peer entity. A more detailed description of the EPC mechanism is given in section 9.7.4~~2~~.

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 section 9.7.3.1. The difference is that this discard method does not use any peer-to-peer signalling. For unacknowledged mode RLC, peer-to-peer signalling is never needed. The SDUs are simply discarded in the transmitter, once the transmission time is exceeded. For acknowledged mode RLC, peer-to-peer signalling can be avoided as long as SDU discard is always performed in the transmitter before it is performed in the receiver. As long as the corresponding SDU is eventually discarded in the receiver too, possible retransmission requests of PDU of discarded SDUs can be ignored by the transmitter. ~~The bigger the time difference is between the triggering of the discard condition at the transmitter and the receiver, the bigger the unnecessary buffering need is at the receiver and the more bandwidth is lost on the reverse link due to unnecessary retransmission requests. On the other hand, forward link bandwidth is saved, as no explicit SDU discard signalling is needed.~~

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 PDU 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 PDU 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 (Status PDU). 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 PDU will be transmitted and the EPC will be reset accordingly. The EPC timer will be started once more.

~~The EPC is based on the estimation of the number of PUs that should have been received during a transmission time interval. To estimate this number is easiest done by means of the TFI bits. However, if these bits are lost due to some reason or another, this estimation must be based on something else. A straightforward solution is to base the estimation on the number done in the previous transmission time interval. Only if the rate has changed this estimation is incorrect. Another method of estimating the number of PUs is based on the maximum allowable rate. The consequence of this is that if the estimation is incorrect, the Status PDU is sent too early. Alternatively, the estimation can be based on the lowest possible transmission rate. In this case, if the estimation is incorrect, the Status PDU will most likely be transmitted too late.~~

11.5.2 Initiation

This procedure is initiated by the receiver in any of following cases:

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 PDU transfer is used and the timer Timer_Status_Periodic has expired.

The receiver shall transmit a STATUS PDU 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 PDU has higher priority than data PDUs.

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

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

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

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

If the EPC meachanism 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.

3GPP TSG-RAN Meeting #6
Nice, France, 13-15 December 1999

***Document* (R2-99i00)**
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.322	CR	002r1	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN#6	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)*	(U)SIM		ME	X	UTRAN / Radio	X	Core Network	

Source: TSG-RAN WG2 **Date:** 1999-11-24

Subject: Editorial changes on RLC protocol specification

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction		**Release:**	Phase 2	
A	Corresponds to a correction in an earlier release			Release 96	
B	Addition of feature			Release 97	
C	Functional modification of feature			Release 98	
D	Editorial modification	X		Release 99	X
				Release 00	

Reason for change: The purpose of these changes is to make some of the contents in the current RLC specification clearer.

Clauses affected: 8.1, 9.2.2.7.1, 9.5

Other specs affected:

Other 3G core specifications		→ List of CRs:	
Other GSM core specifications		→ List of CRs:	
MS test specifications		→ List of CRs:	
BSS test specifications		→ List of CRs:	
O&M specifications		→ List of CRs:	

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

-

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, CFN, MUI	Data	Not Defined	MUI
RLC-UM-DATA	Data,	Data	Not Defined	Not Defined
RLC-TR-DATA	Data	Data	Not Defined	Not Defined
CRLC-CONFIG	E/R, Ciphering Elements (UM/AM only), AM_parameters (AM only)	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.
- 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, release or reconfigure the RLC. Ciphering elements are included for UM and AM operation.

CRLC-STATUS-Ind

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

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 alligned.
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 whether RLC should enter or exit the data transfer ready state.

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) ~~6)~~ The parameter ciphering elements are only applicable for UM and AM operation. These parameters are Ciphering Mode, Ciphering Key and Ciphering Sequence Number.

7) The AM_parameters is only applicable for AM operation. It contains PU size, Timer values (see section 9.5), Protocol parameter values (see section 9.6), Polling triggers (see section 9.7.1), Status triggers (see section 9.7.2), SDU discard mode (see section 9.7.3),.

9.2.2.7.1 AMD PDU Extended Header

The Extended Header is used when additional sequence numbers are needed to indicate PUs that are not sequential within a PDU or when the rest of a PDU, which is not filled by PUs, is equal or larger than the size of a PU. A PDU that includes more than one sequence number shall include sequence numbers for all PUs in the PDU. The n^{th} sequence number in the PDU indicates the sequence number of the n^{th} PU in the PDU. The decision to use Extended Header is made by the transmitting RLC.

First all the Extended Headers are listed. Then all Length Indicators are listed. Finally the PUs follow.

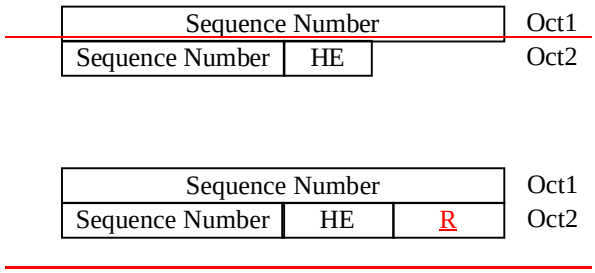


Figure 9-7: Format of the extended header

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 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.

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 has been sent. The timer is started when a STATUS report is transmitted and when it expires EPC can start decrease (see section 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. If the SDU has not been acknowledged when the timer expires, the SDU is discarded and a Move Receiving Window request is sent to the receiver. If the SDU discard function does not use the Move Receiving Window request, the timer is also used in the receiver, where it is activated once a PDU is detected as outstanding, i.e. there is a gap between sequence numbers of received PDUs. 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 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 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.

g) 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.

h) 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.

3GPP TSG-RAN Meeting #6
Nice, France, 13-15 December 1999

***Document* (R2-99i01)**
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.322	CR	007	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to: TSG-RAN#6 *list expected approval meeting # here ↑*

for approval	X	strategic		*(for SMG use only)*
for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 1999-11-29

Subject: Updated RLC SDL

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction	
A	Corresponds to a correction in an earlier release	
B	Addition of feature	
C	Functional modification of feature	
D	Editorial modification	X

Release:

Phase 2	
Release 96	
Release 97	
Release 98	
Release 99	X
Release 00	

Reason for change: Correction of the editorial mistakes in the current RLC SDL and updating it in accordance with the latest RLC specification.

Clauses affected: Annex A

Other specs affected:

Other 3G core specifications		→ List of CRs:
Other GSM core specifications		→ List of CRs:
MS test specifications		→ List of CRs:
BSS test specifications		→ List of CRs:
O&M specifications		→ List of CRs:

Other comments:


help.doc

<--------- double-click here for help and instructions on how to create a CR.

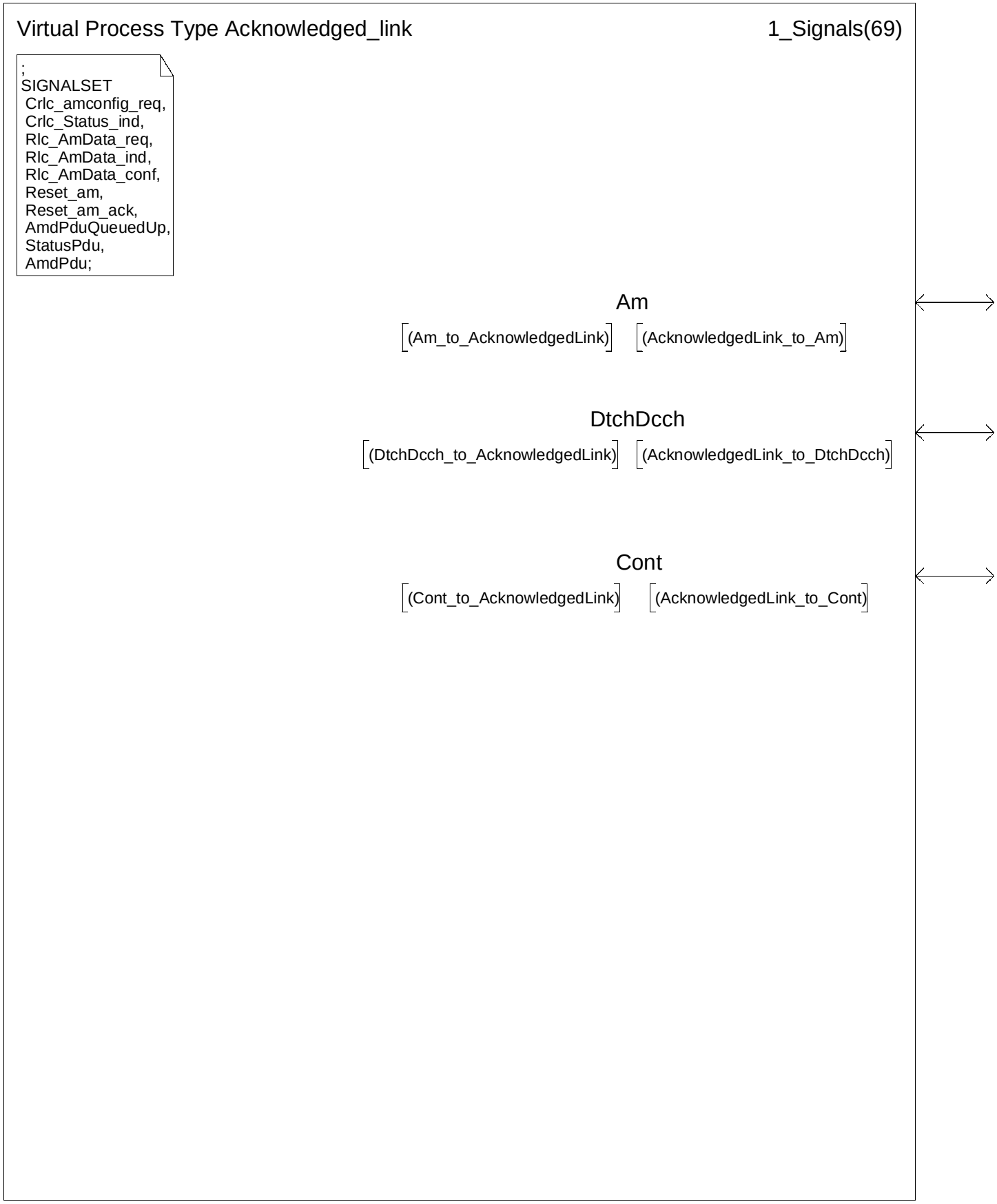
-

Annex A (informative): SDL diagrams

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

[All the section shall be reviewed when the protocol is defined;

all the SDL diagrams presented are [FFS]]


Virtual Process Type Acknowledged_link
1_Signals(69)
;
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(69)

;
SIGNALSET

```
DCL


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

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

amd_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 an StatPdu.*/


/*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(69)

;
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.*/

 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.*/

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

Virtual Process Type Acknowledged_link 3_Declarations(69)

;
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.*/

 poll_answer                                   IndicatorType,
 /*This indicator stores whether a status report is sent as an answer to a poll
    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(69)

;
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.*/

 ciphering_sequence_number                    CipheringSequenceNumberType,
/*The ciphering sequence number.*/

 pdu_size                                     OctetType,
/*The size in octets of an AMD PDU.*/

 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.*/

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

Virtual Process Type Acknowledged_link 5_Declarations(69)

;
SIGNALSET

```
DCL

/*Local variables declarations:_____*/

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

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

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

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

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.*/
```

Virtual Process Type Acknowledged_link 6_Declarations(69)

;
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(69)

;
SIGNALSET
Crlc_amconfig_req

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

 vt_dat                                          SequenceNumberType,
 /*This is a local variable that stores the highest value associated with any
    PU within the PDU formed from the retransmission queue.*/

 vt_mrw                                          SequenceNumberType;
 /*A variable used to keep track of the number of transmissions of MRW that has
   occurred.*/
```

Virtual Process Type Acknowledged_link 8_Declarations(69)

```
;
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 acknowled-
     gement 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 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. 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 (see section 9.7.3). 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 and a
     Move Receiving Window request is sent to the receiver. If the SDU discard function does not use the Move
     Receiving Window request, the timer is also used in the receiver, where it is activated once a PDU is detected
     as outstanding, i.e. there is a gap between sequence numbers of received PDUs. 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. The value of the timer
     is signalled by RRC.*/

  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 to keep track of the response to the MRW sufi type.*/

 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(69)

;
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(69)

;
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 places a piggybacked STATUS PDU onto the first AMD PDU within a queue.

FPAR

IN/OUT qu Queue,

IN/OUT re_qu Queue,

IN/OUT stat_pdu StatPdu,

IN pa IndicatorType,

IN/OUT pos IndicatorType;

Place_in_mui_queue

This procedure places a message identifier in the sdu 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(69)

;
SIGNALSET
Crlc_amconfig_req

Place_in_receive_side_queue

This procedure places a PU in one of the receive 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_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(69)

;
SIGNALSET

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_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_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 5_LocalProcedures(69)

;
SIGNALSET

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 transmission 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;

Remove_bitmap_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 in accordance with a bitmap from the transmission queue and retranmission queue.

FPAR

IN/OUT	qu	Queue,
IN/OUT	re_qu	Queue,
IN	sq	SequenceNumberType,
IN/OUT	no	PduIndexType,
IN/OUT	bitmap	IndicatorArrayType,
IN/OUT	tot	PduIndexType,
IN/OUT	pus	AmPuArrayStructType;

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;

Virtual Process Type Acknowledged_link 6_LocalProcedures(69)

;
SIGNALSET

Remove_rlist_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 in accordance with a codewords from the transmission queue and retranmission queue.

FPAR

IN/OUT	qu	Queue,
IN/OUT	re_qu	Queue,
IN	sq	SequenceNumberType,
IN/OUT	no	PduIndexType,
IN/OUT	codewords	IndicatorArrayType,
IN/OUT	tot	PduIndexType,
IN/OUT	pus	AmPuArrayType,
IN/OUT	poss	IndicatorType;

Remove_all_below_mrw_from_queue

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

FPAR

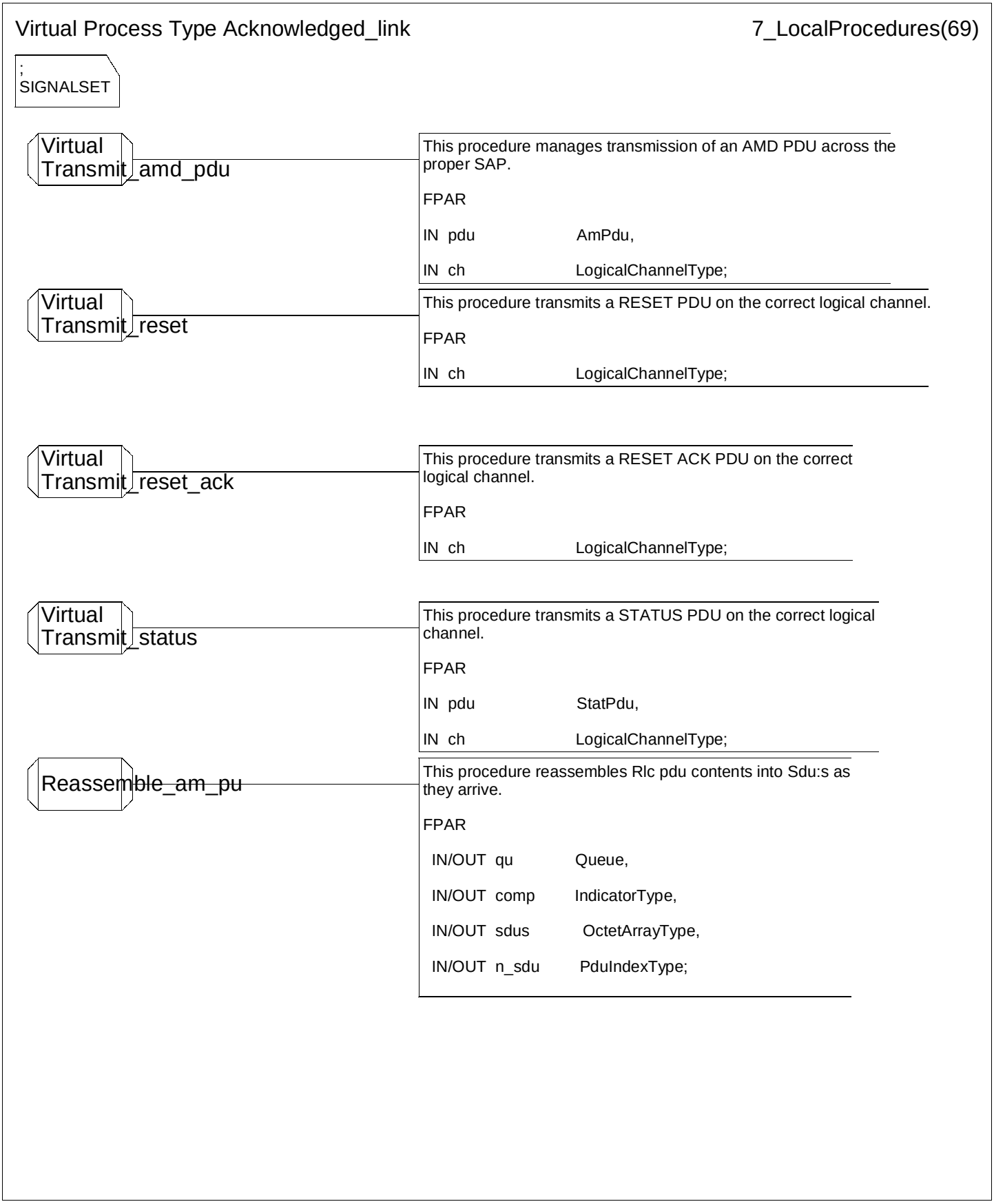
IN/OUT	r_qu	Queue,
IN/OUT	a_qu	Queue,
IN/OUT	sn	SequenceNumberType;

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;


Virtual Process Type Acknowledged_link
7_LocalProcedures(69)
;
SIGNALSET
Virtual Transmit_amd_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;
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(69)

;
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 ssets 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(69)

;
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;

Virtual Process Type Acknowledged_link 10_LocalProcedures(69)

;
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(69)

;
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.

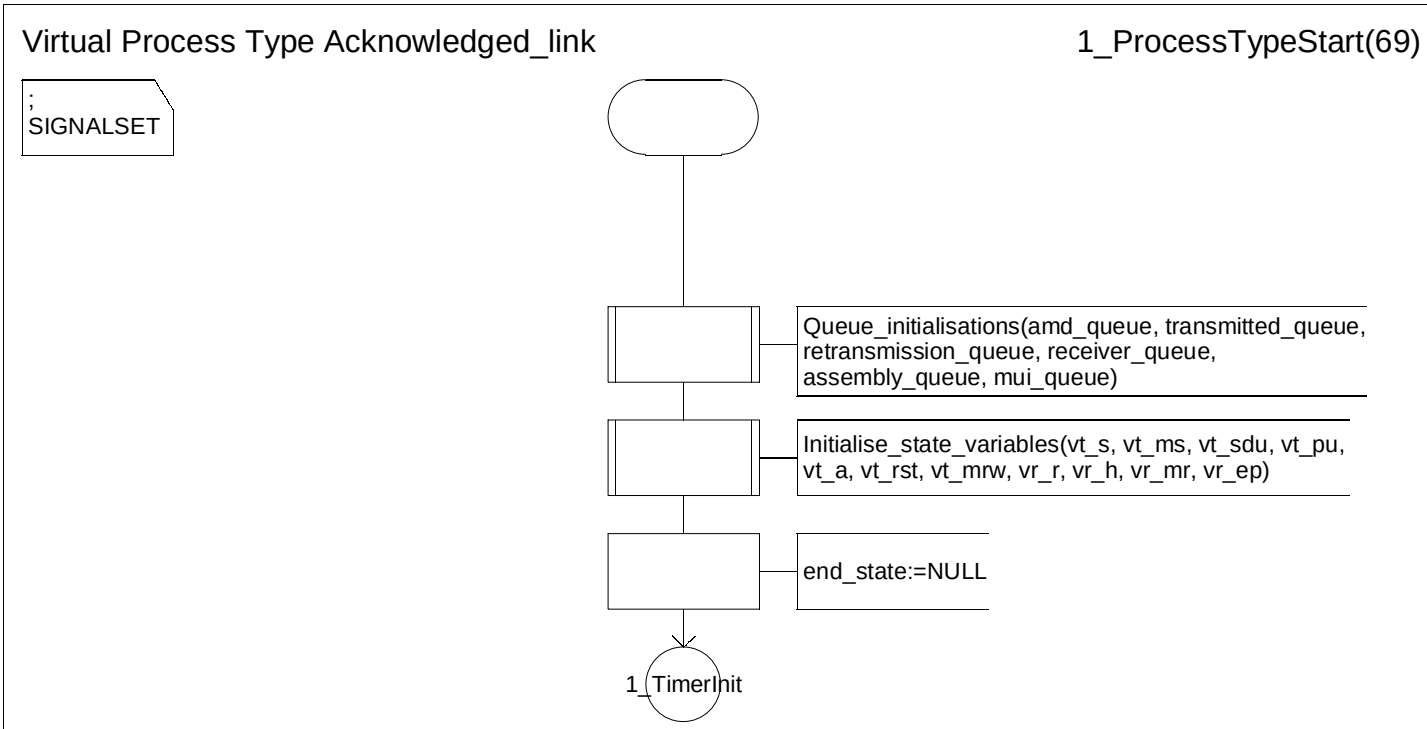
FPAR

IN/OUT pdu AmPdu,

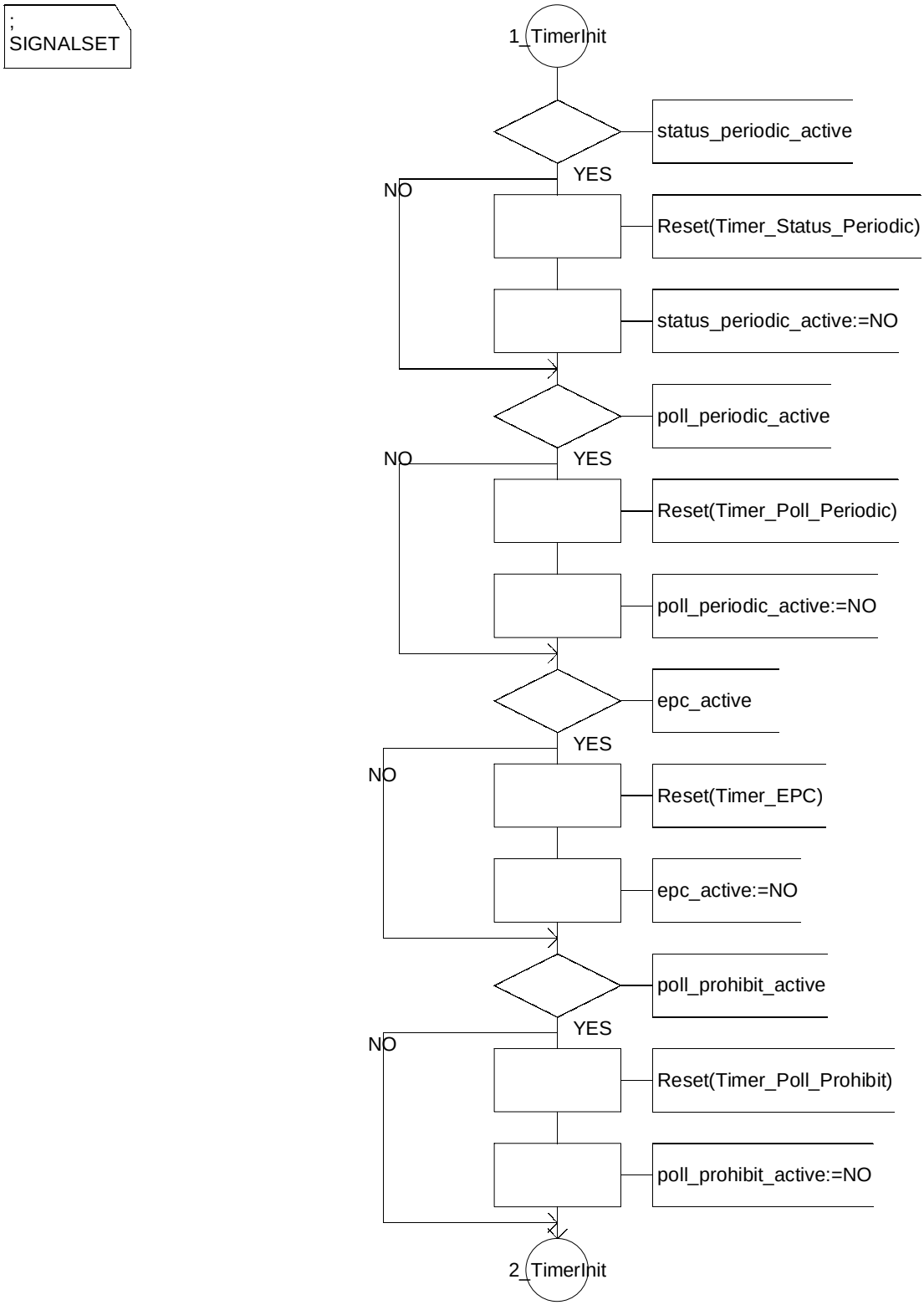
IN/OUT poll_win Real,

IN vt_ms SequenceNumberType,

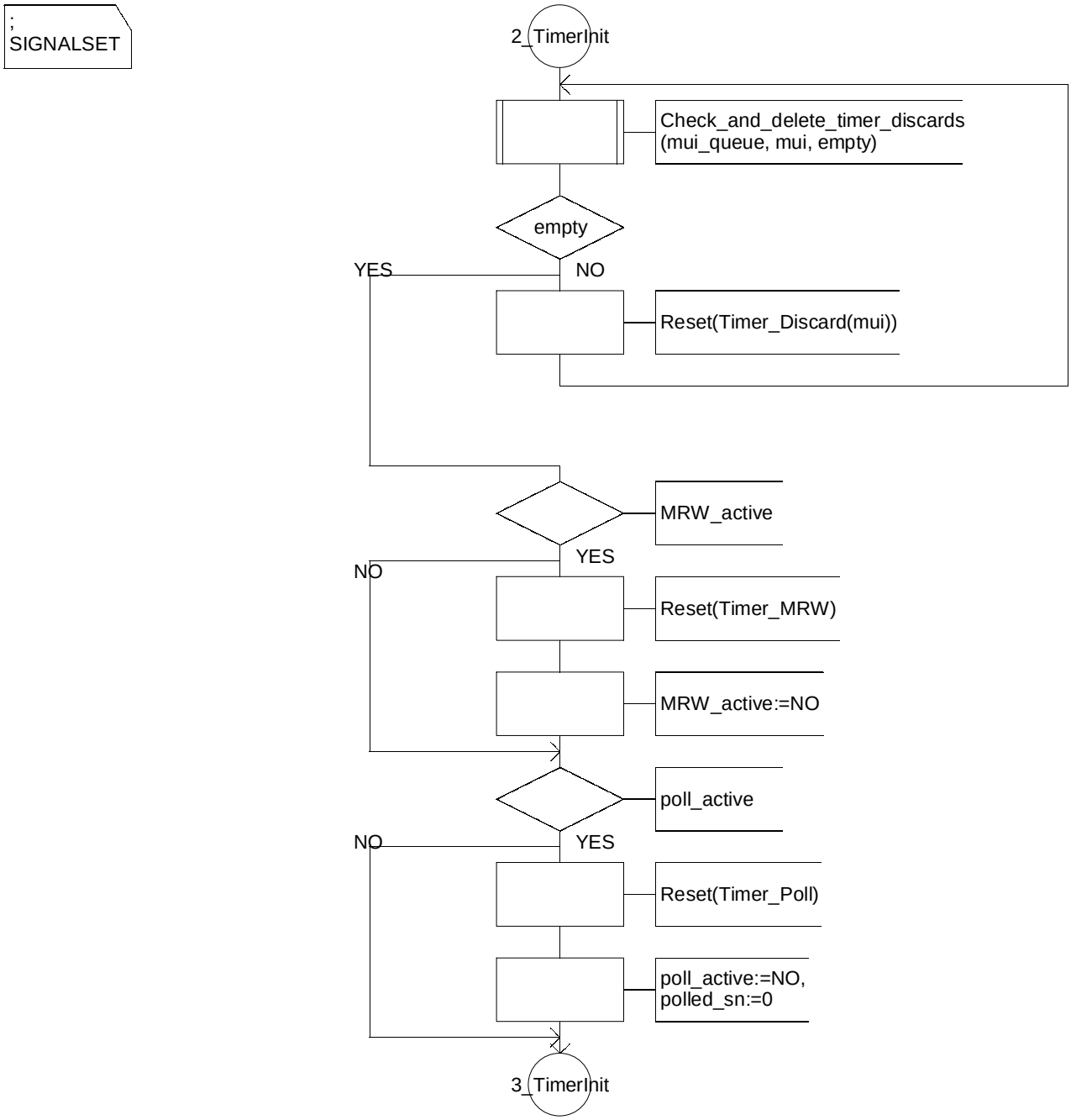
IN tx_win SequenceNumberType;


Virtual Process Type Acknowledged_link
1_ProcessTypeStart(69)
;
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)
end_state:=NULL
1_TimerInit

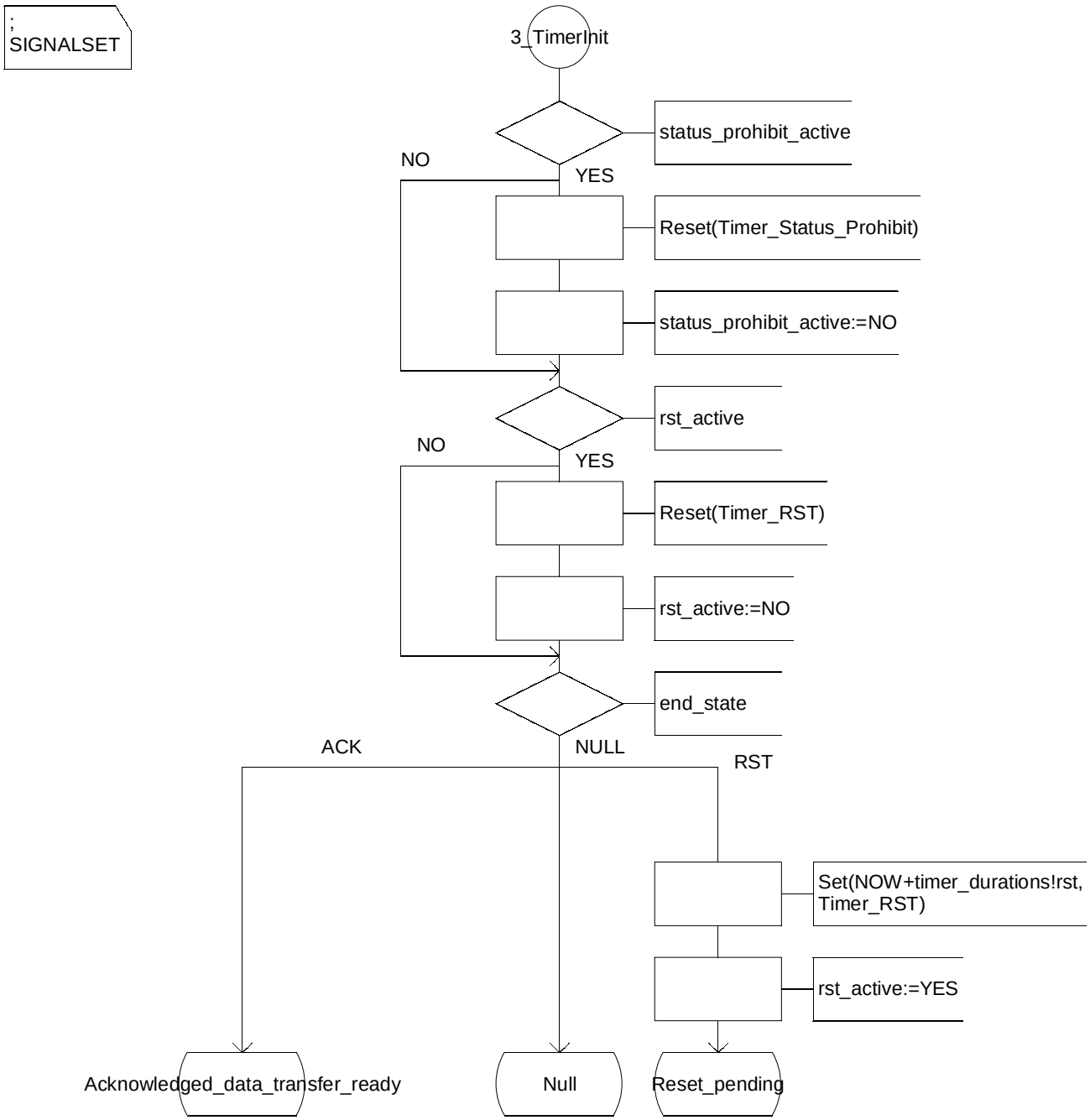
Virtual Process Type Acknowledged_link 1_TimerInit(69)

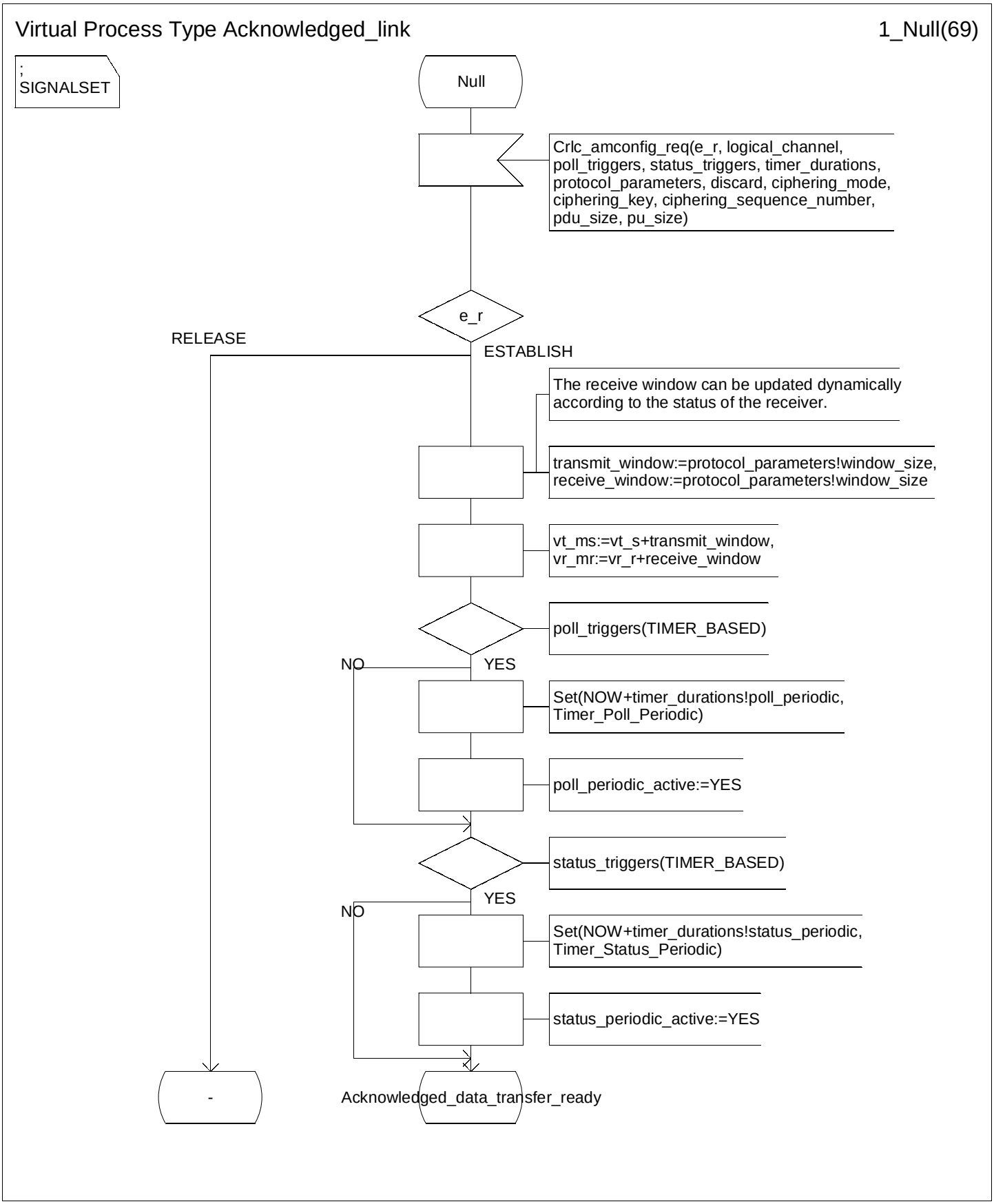


Virtual Process Type Acknowledged_link 2_TimerInit(69)

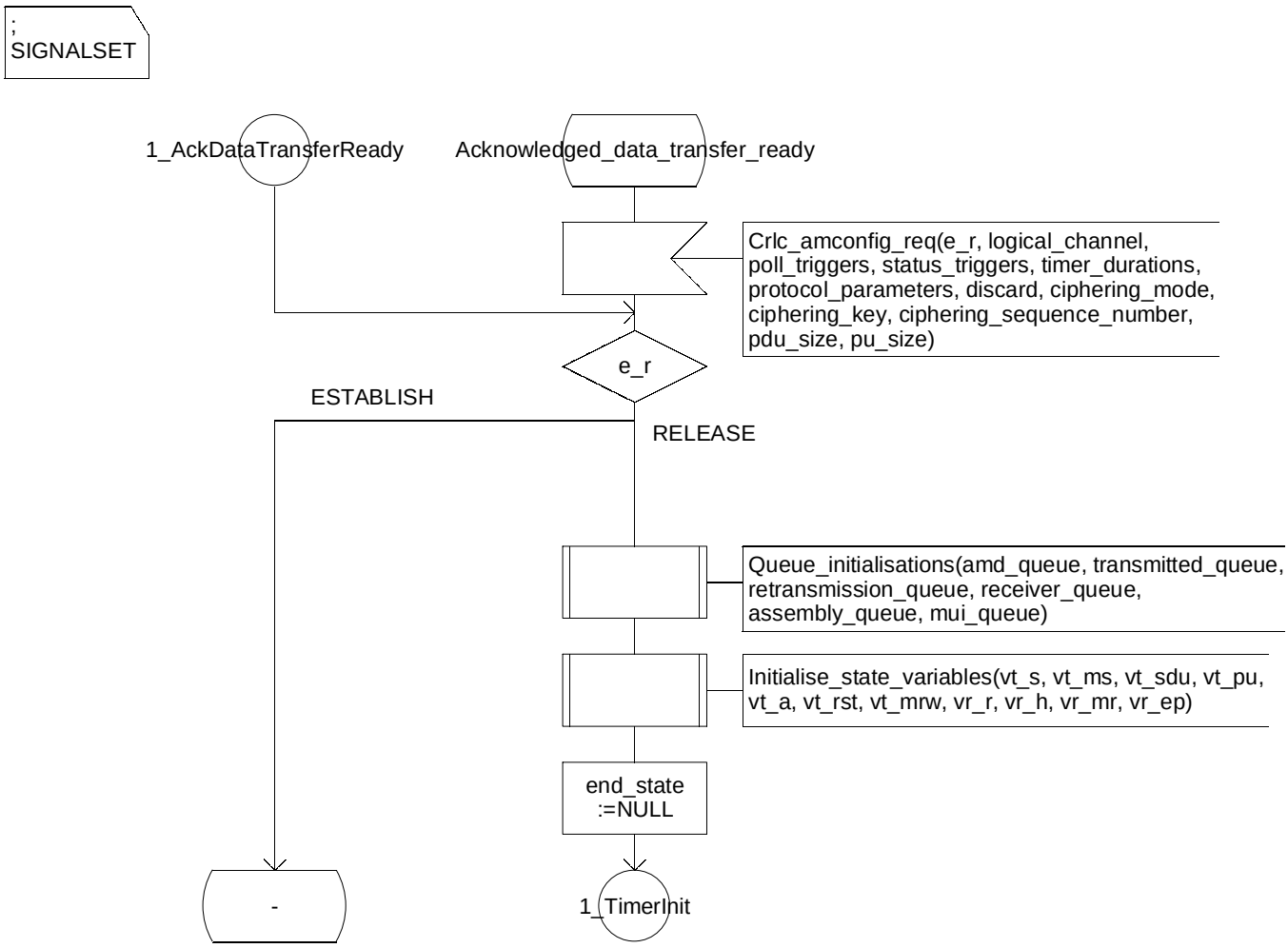


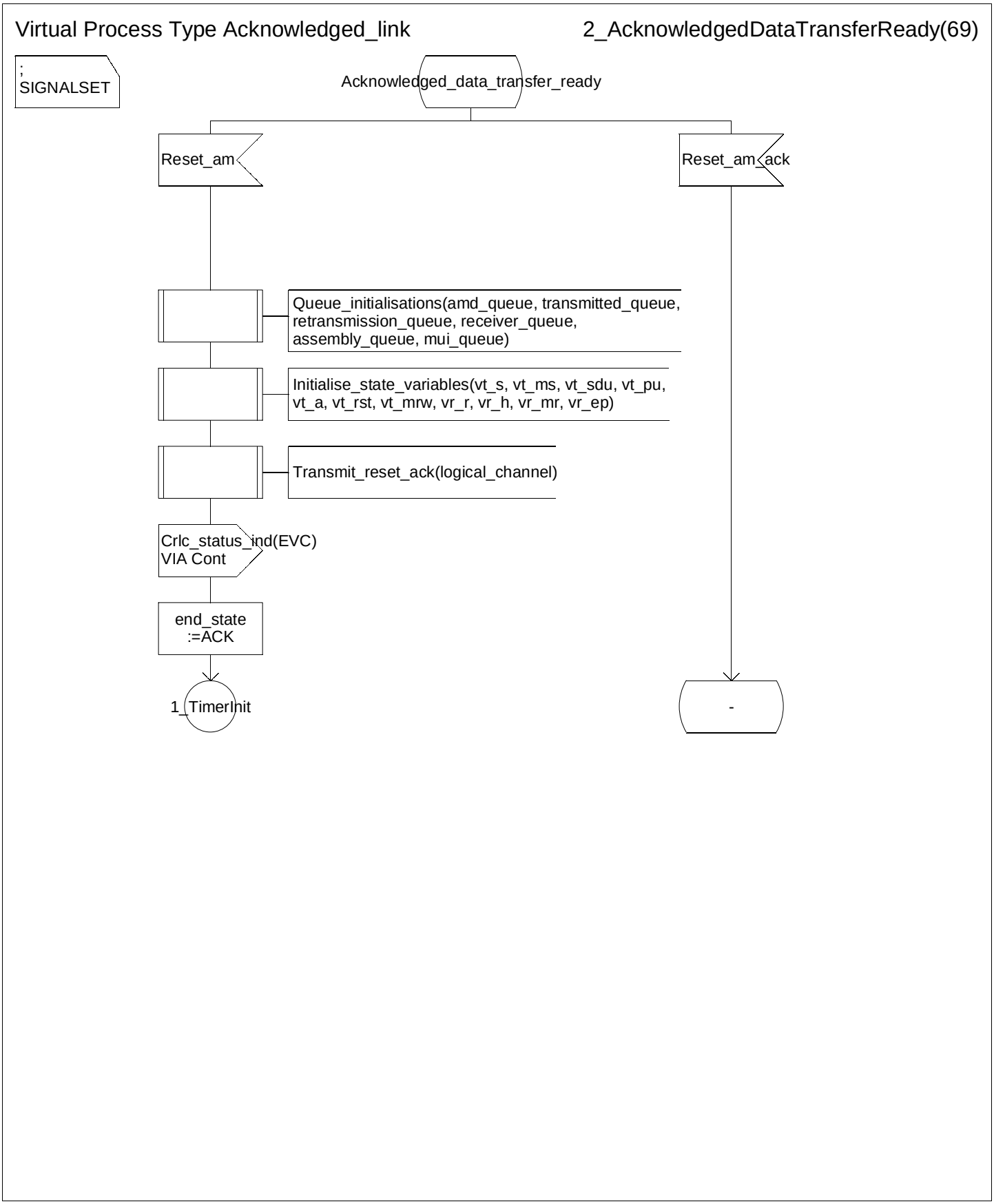
Virtual Process Type Acknowledged_link 3_TimerInit(69)



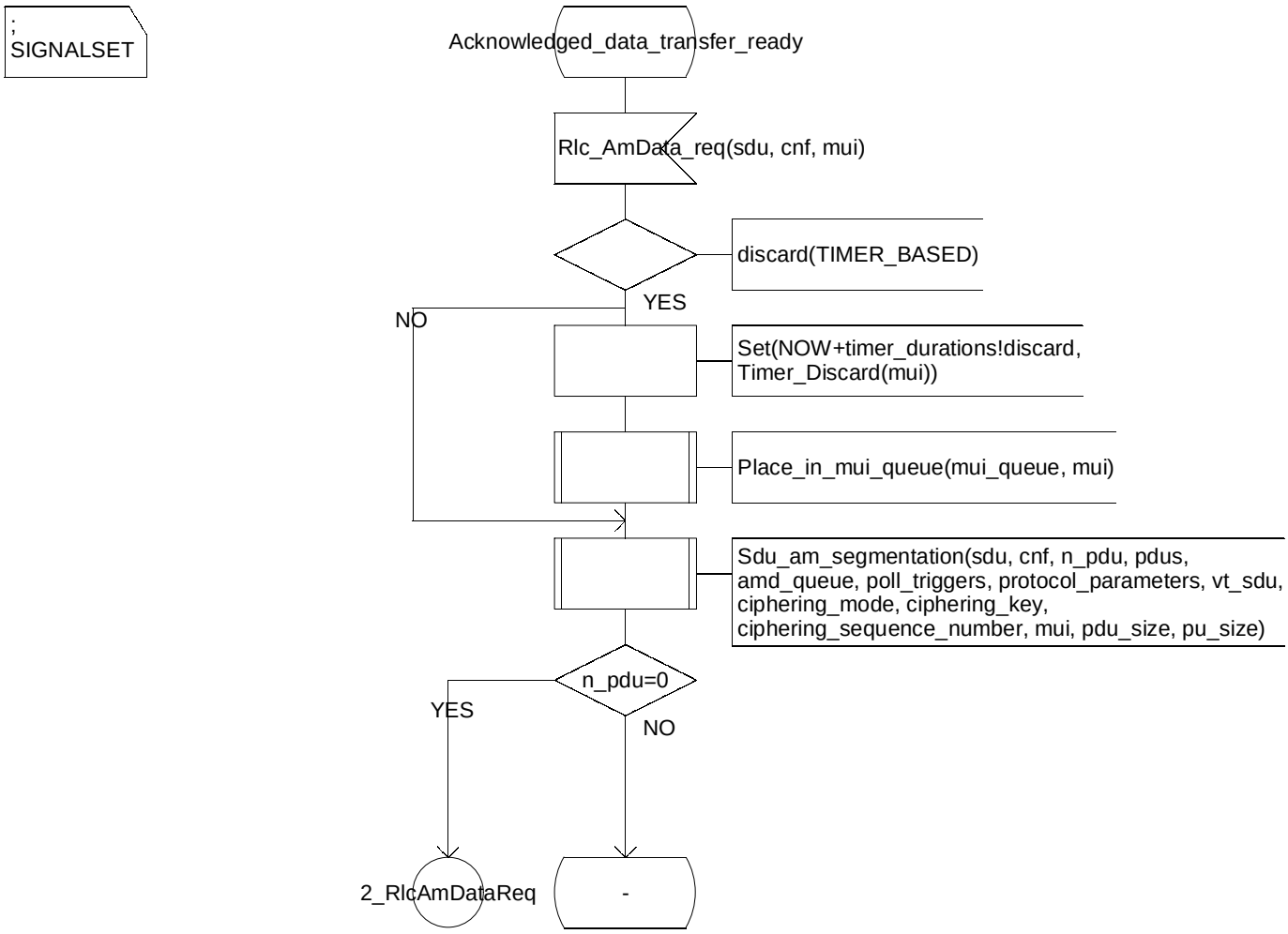

Virtual Process Type Acknowledged_link
1_Null(69)
;
SIGNALSET
Null
Crlc_amconfig_req(e_r, logical_channel,
poll_triggers, status_triggers, timer_durations,
protocol_parameters, discard, ciphering_mode,
ciphering_key, ciphering_sequence_number,
pdu_size, pu_size)
e_r
RELEASE
ESTABLISH
The receive window 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)
YES
NO
Set(NOW+timer_durations!status_periodic,
Timer_Status_Periodic)
status_periodic_active:=YES
-
Acknowledged_data_transfer_ready

Virtual Process Type Acknowledged_link 1_AcknowledgedDataTransferReady(69)

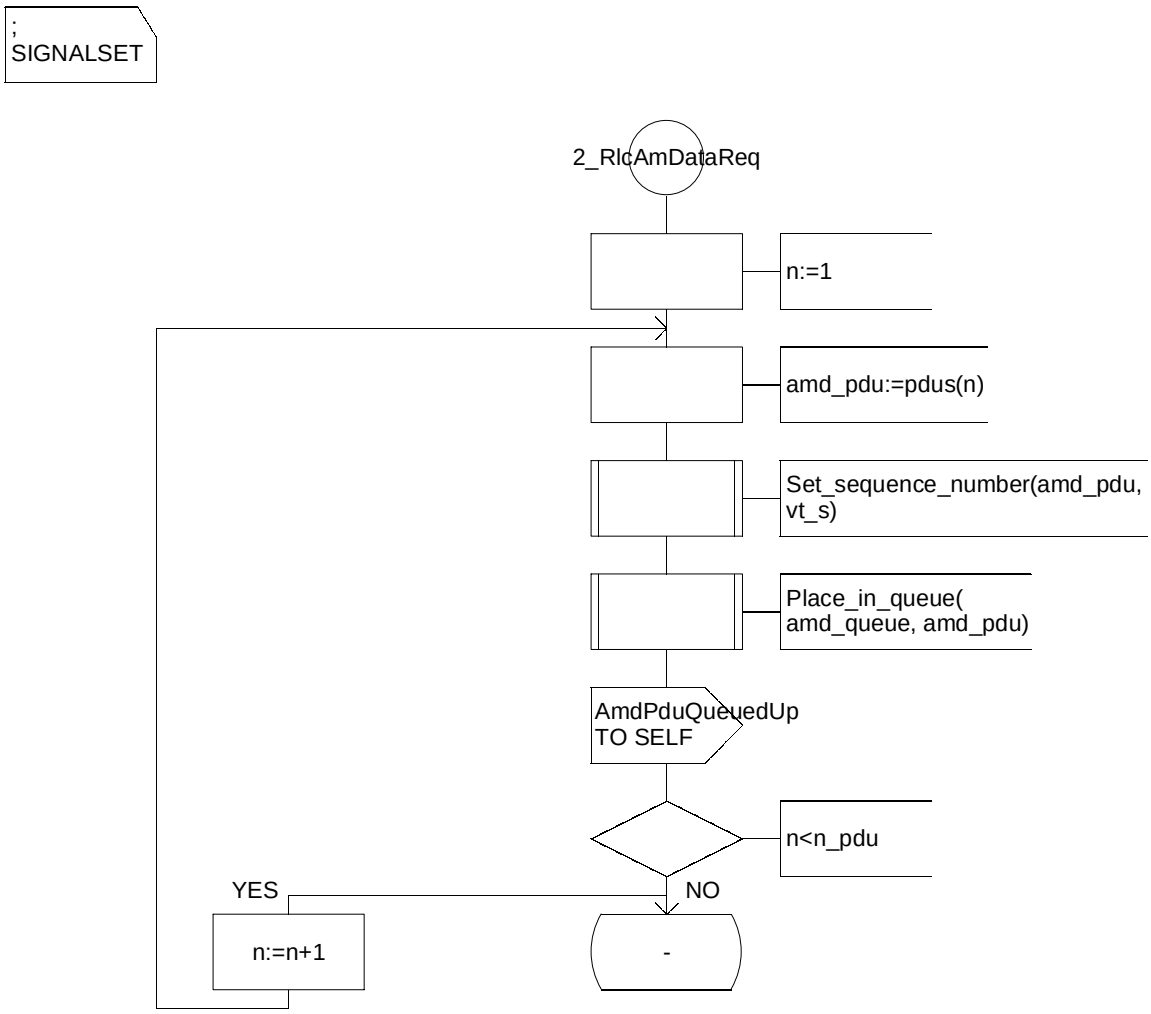


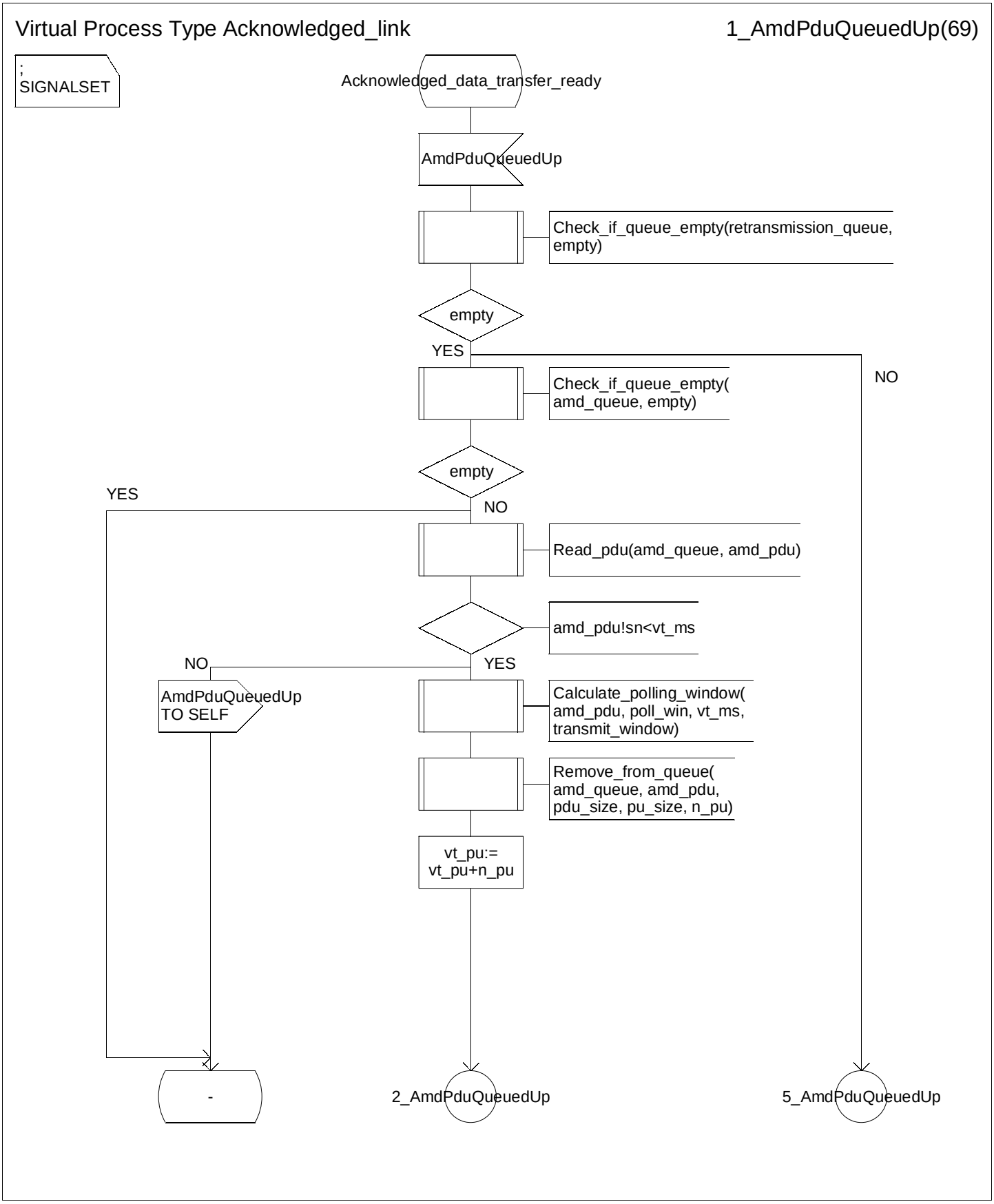

Virtual Process Type Acknowledged_link
2_AcknowledgedDataTransferReady(69)
;
SIGNALSET
Acknowledged_data_transfer_ready
Reset_am
Reset_am_ack
Queue_initialisations(amd_queue, transmitted_queue, retransmission_queue, receiver_queue, assembly_queue, mui_queue)
Initialise_state_variables(vt_s, vt_ms, vt_sdu, vt_pu, vt_a, vt_rst, vt_mrw, vr_r, vr_h, vr_mr, vr_ep)
Transmit_reset_ack(logical_channel)
Crlc_status_ind(EVC)
VIA Cont
end_state
:=ACK
1_TimerInit
-

Virtual Process Type Acknowledged_link 1_RlcAmDataReq(69)

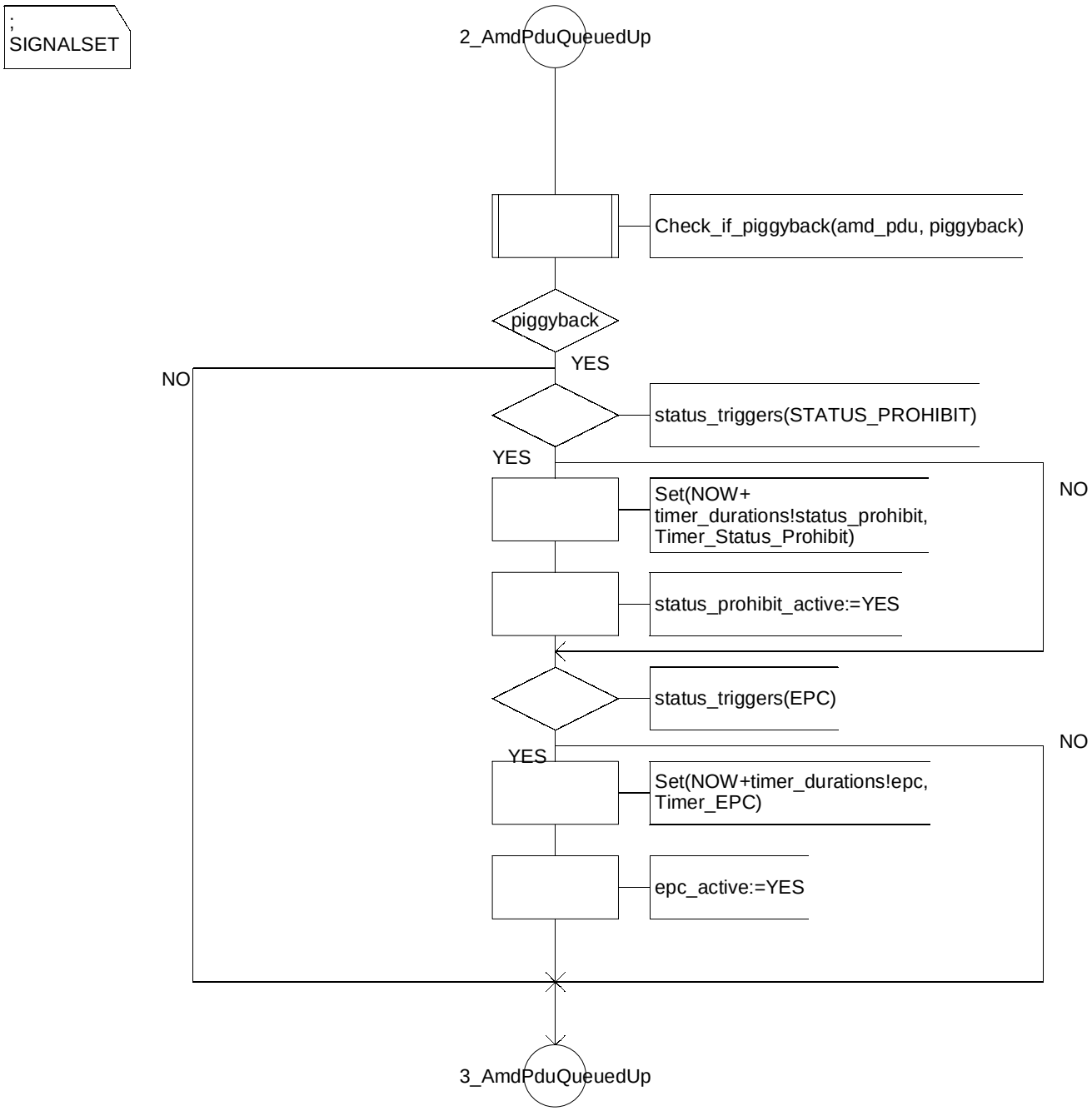


Virtual Process Type Acknowledged_link 2_RlcAmDataReq(69)

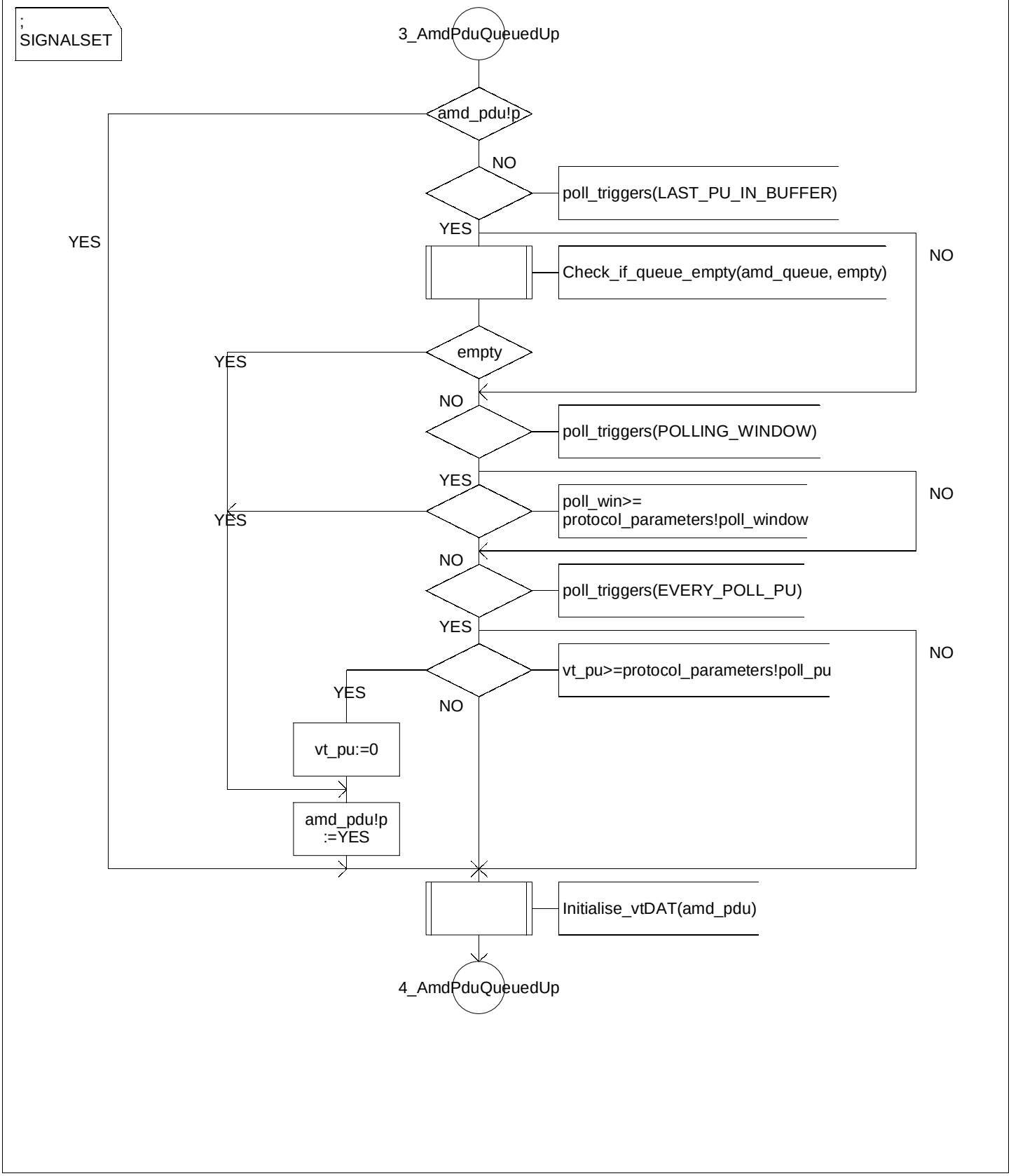



Virtual Process Type Acknowledged_link
1_AmdPduQueuedUp(69)
;
SIGNALSET
Acknowledged_data_transfer_ready
AmdPduQueuedUp
Check_if_queue_empty(retransmission_queue, empty)
empty
YES
NO
Check_if_queue_empty(amd_queue, empty)
empty
YES
NO
Read_pdu(amd_queue, amd_pdu)
amd_pdu!sn<vt_ms
NO
YES
AmdPduQueuedUp TO SELF
Calculate_polling_window(amd_pdu, poll_win, vt_ms, transmit_window)
Remove_from_queue(amd_queue, amd_pdu, pdu_size, pu_size, n_pu)
vt_pu:= vt_pu+n_pu
-
2_AmdPduQueuedUp
5_AmdPduQueuedUp

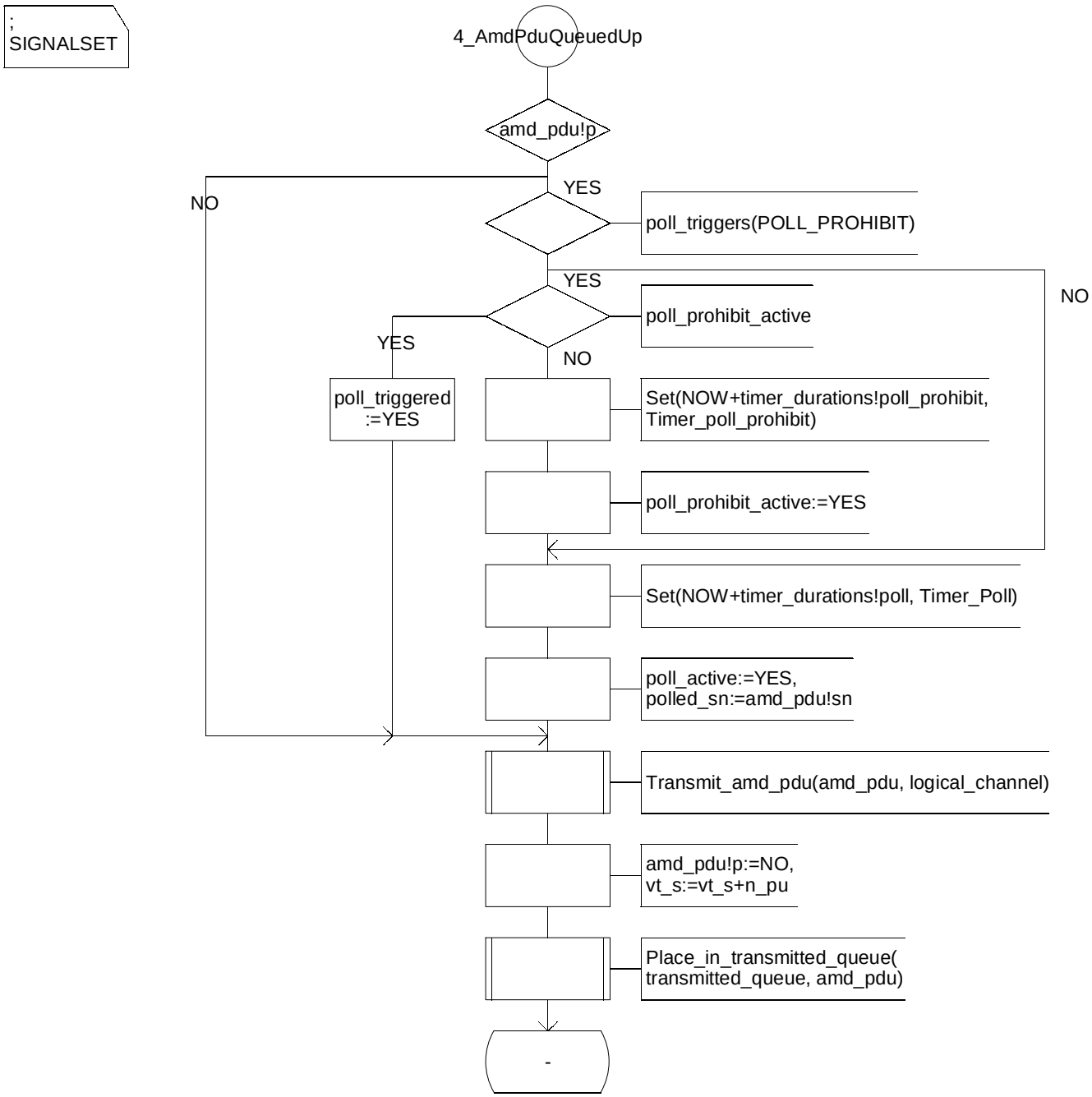
Virtual Process Type Acknowledged_link 2_AmdPduQueuedUp(69)



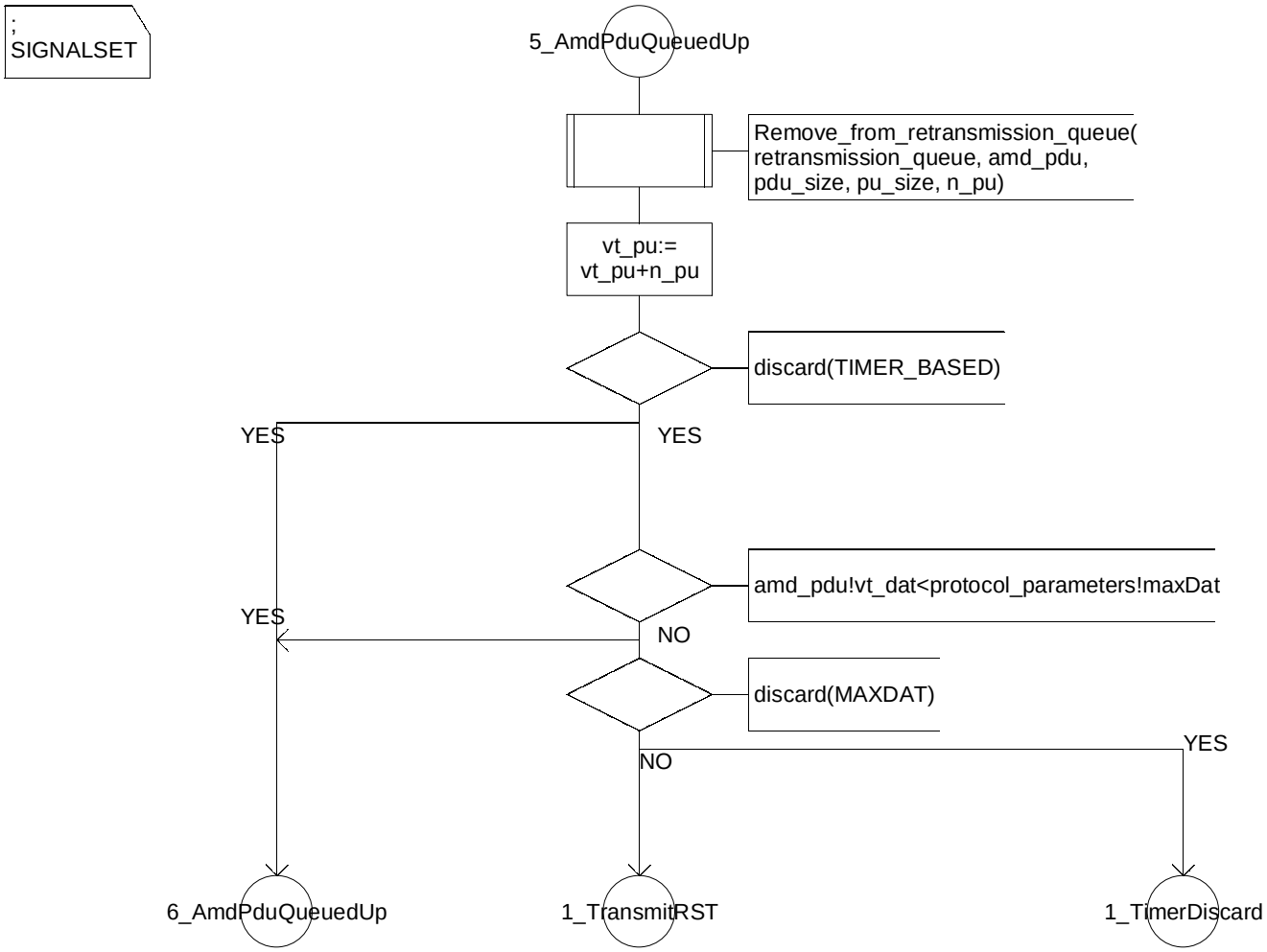
Virtual Process Type Acknowledged_link 3_AmdPduQueuedUp(69)



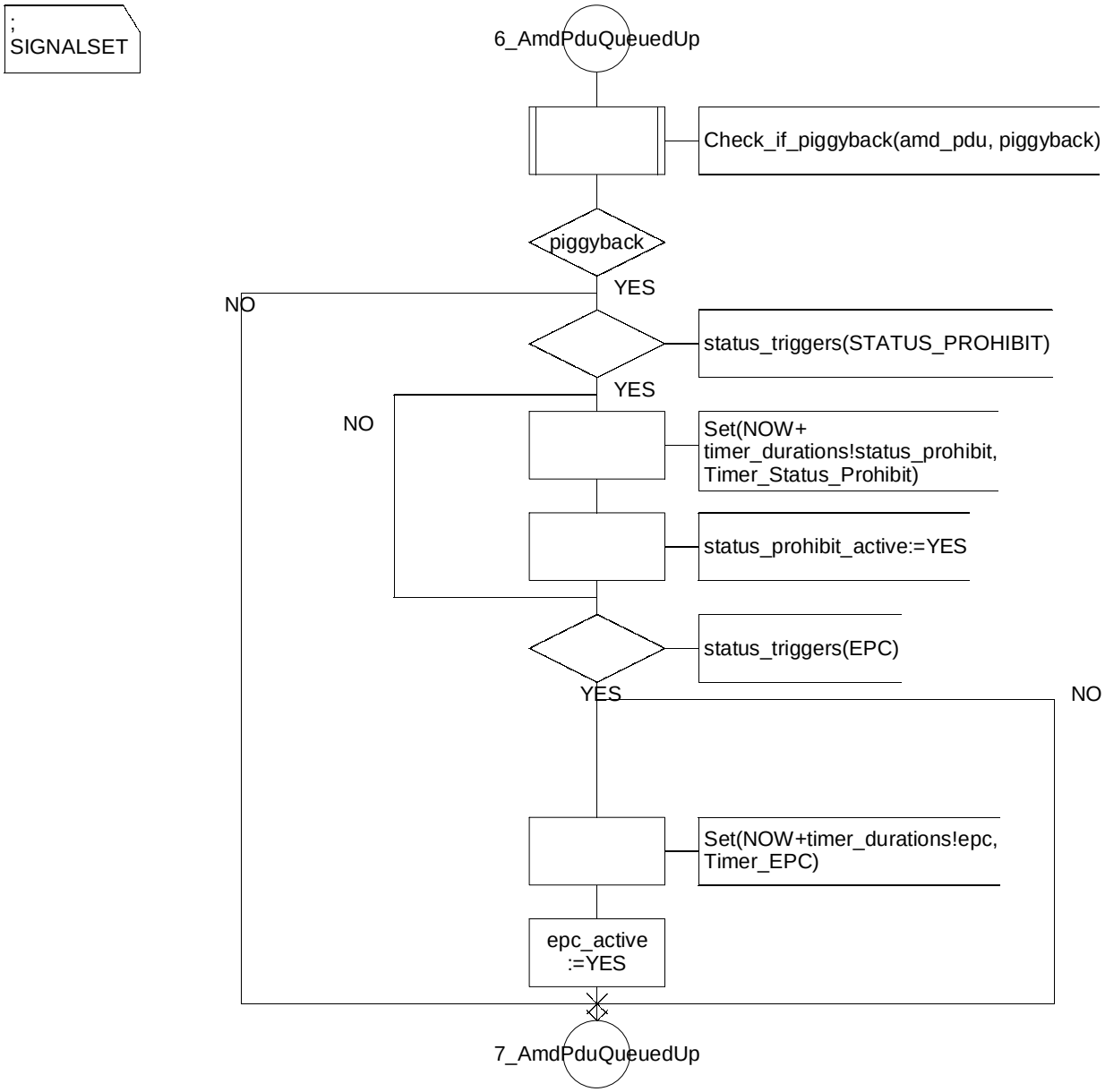
Virtual Process Type Acknowledged_link 4_AmdPduQueuedUp(69)



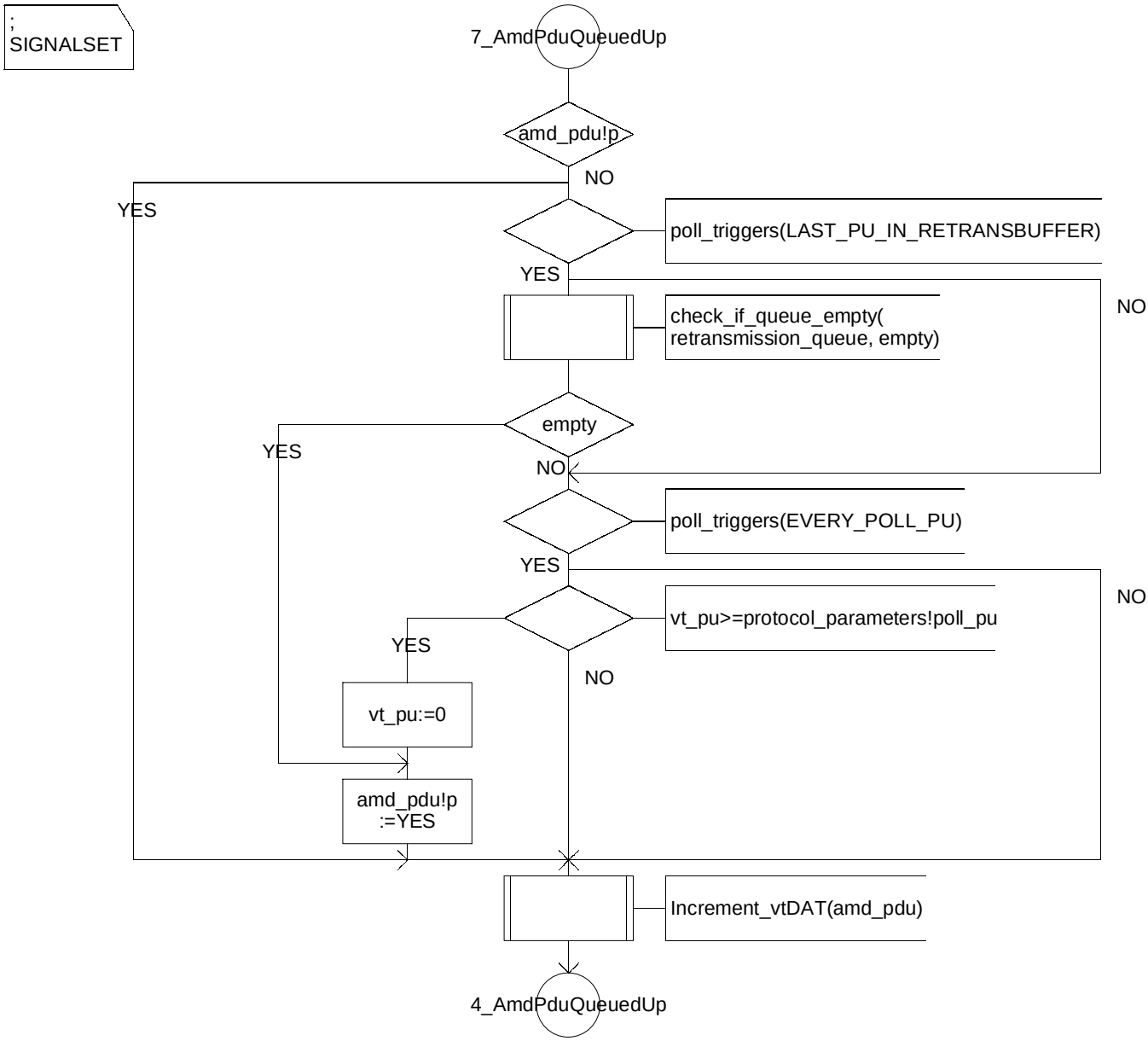
Virtual Process Type Acknowledged_link 5_AmdPduQueuedUp(69)



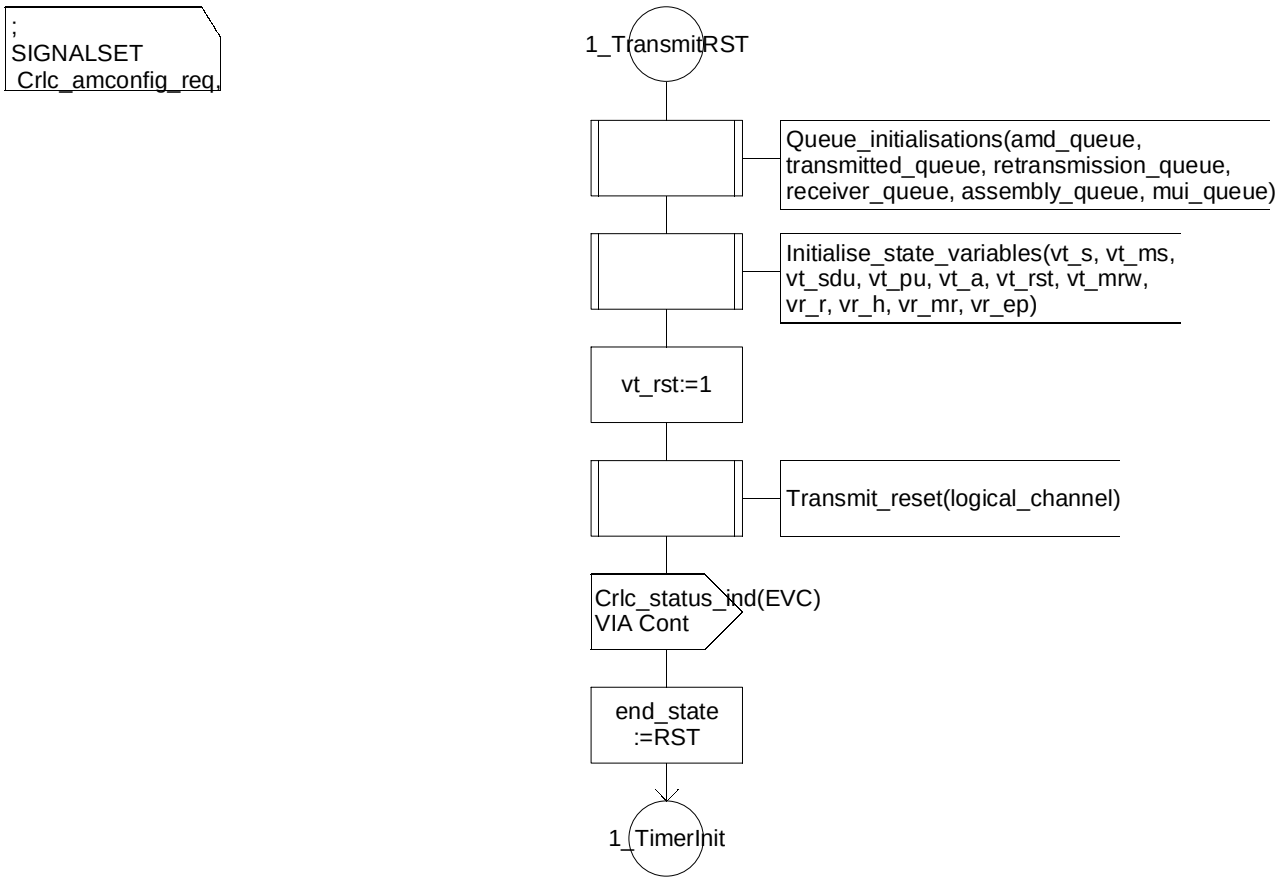
Virtual Process Type Acknowledged_link 6_AmdPduQueuedUp(69)



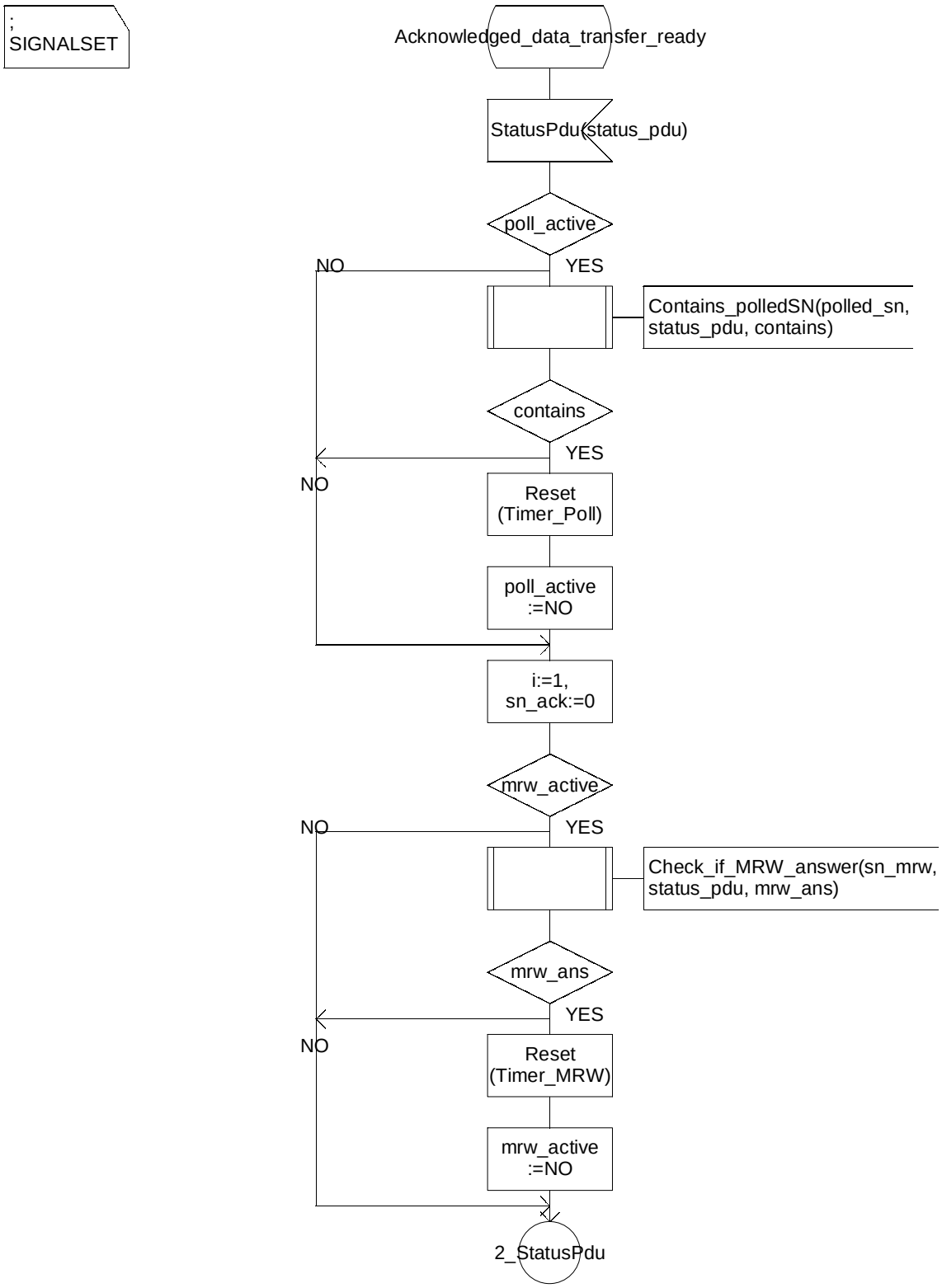
Virtual Process Type Acknowledged_link 7_AmdPduQueuedUp(69)

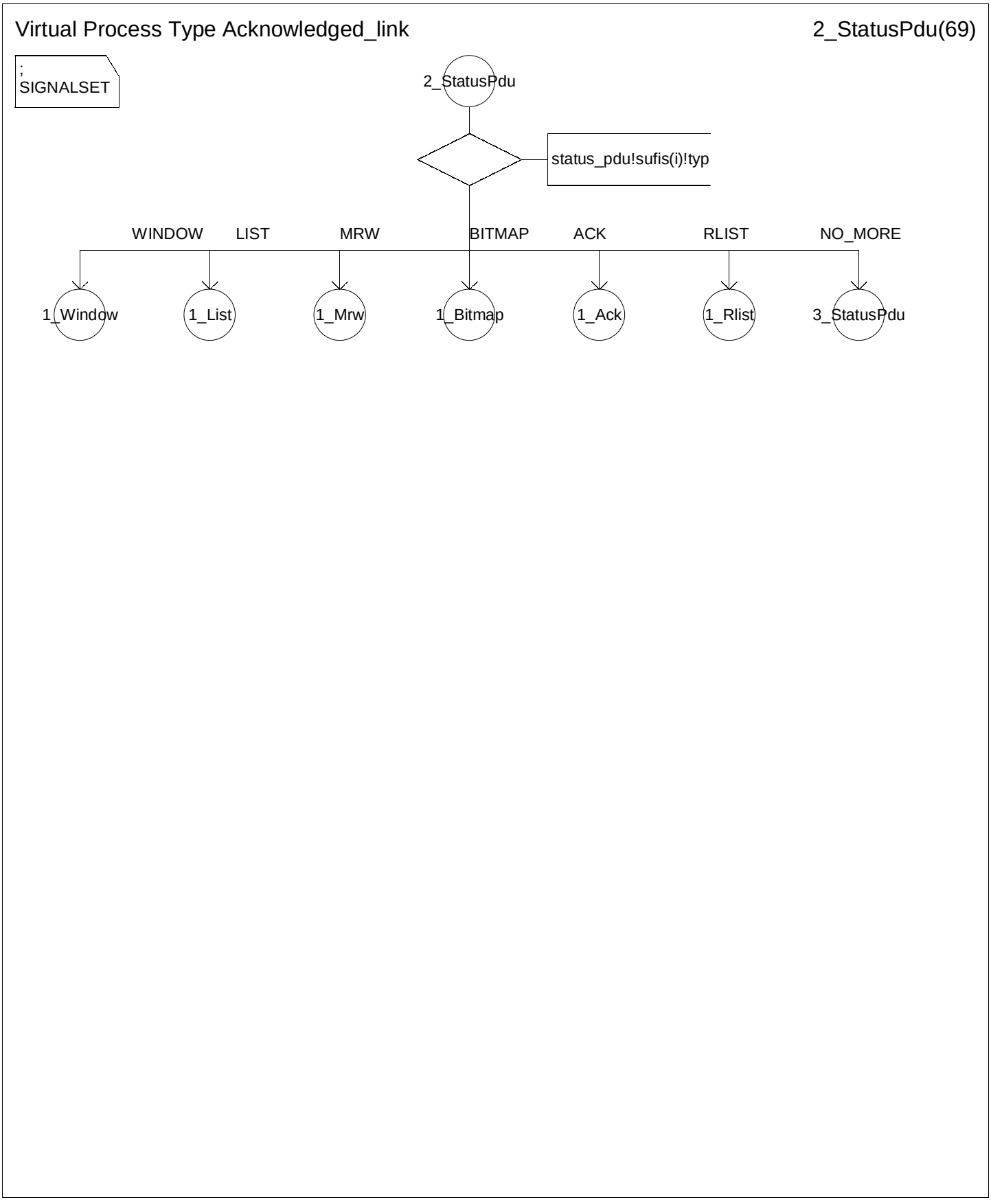


Virtual Process Type Acknowledged_link 1_TransmitRST(69)

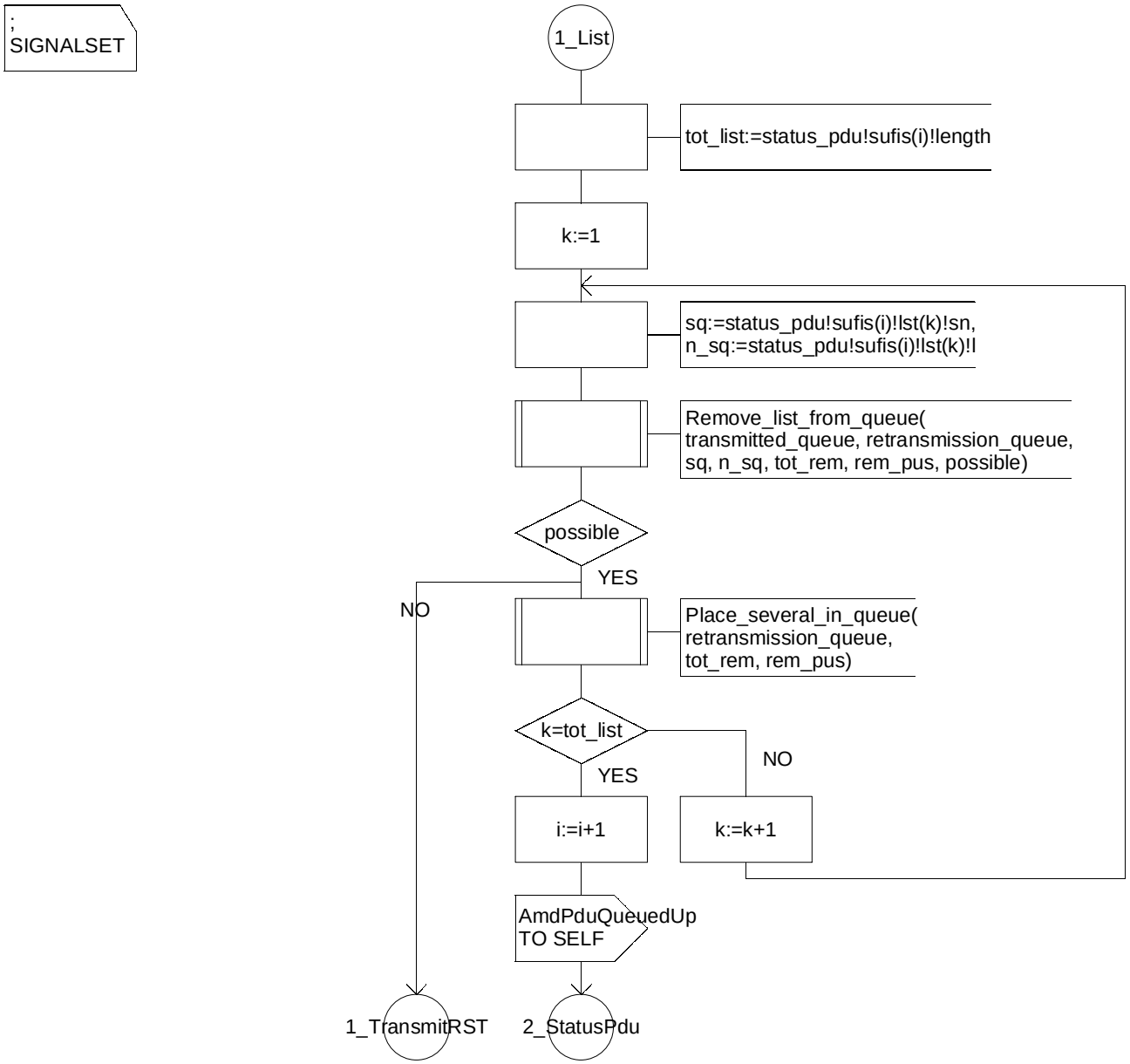


Virtual Process Type Acknowledged_link 1_StatusPdu(69)

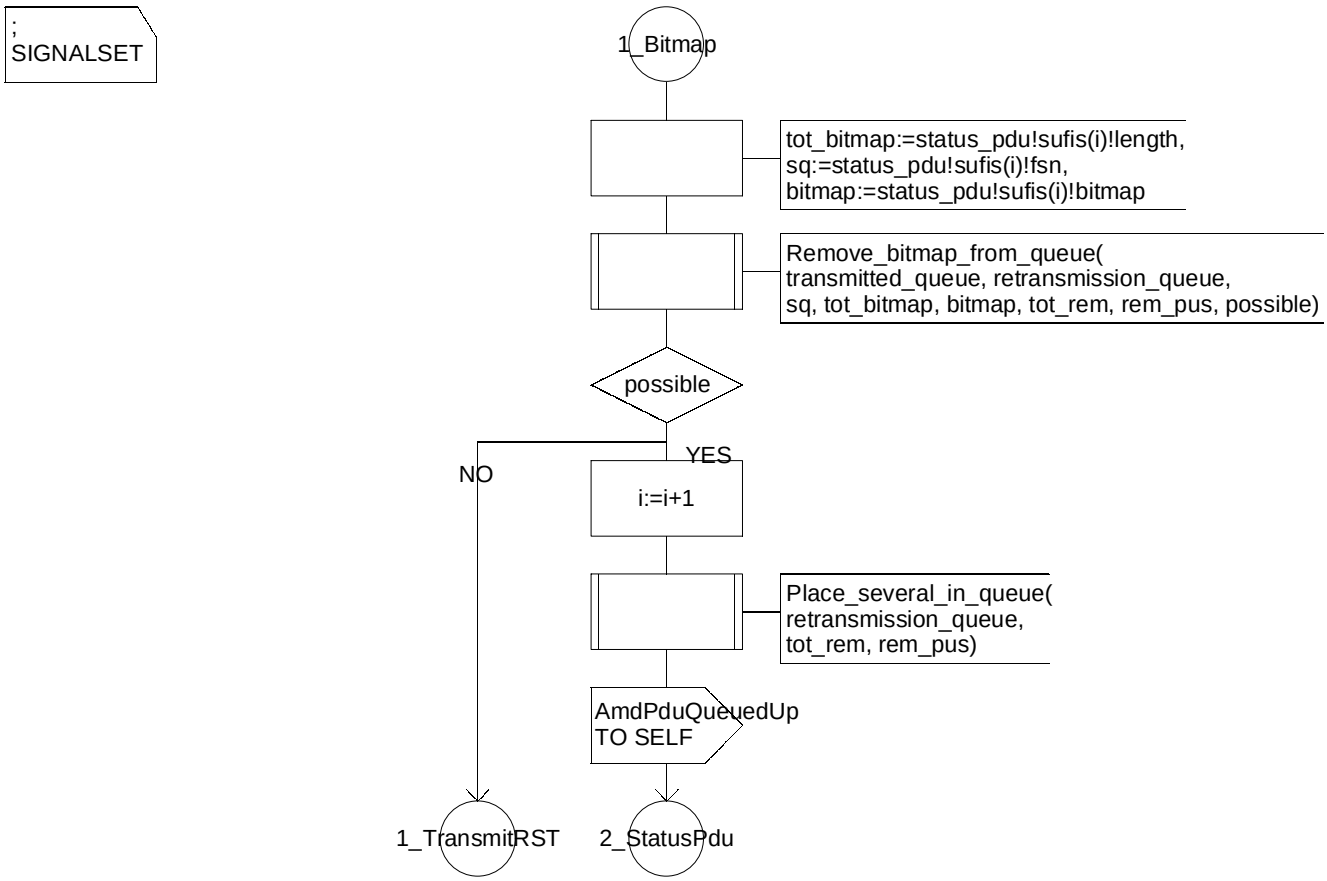



Virtual Process Type Acknowledged_link
2_StatusPdu(69)
;
SIGNALSET
2_StatusPdu
status_pdu!sufis(i)!typ
WINDOW
LIST
MRW
BITMAP
ACK
RLIST
NO_MORE
1_Window
1_List
1_Mrw
1_Bitmap
1_Ack
1_Rlist
3_StatusPdu

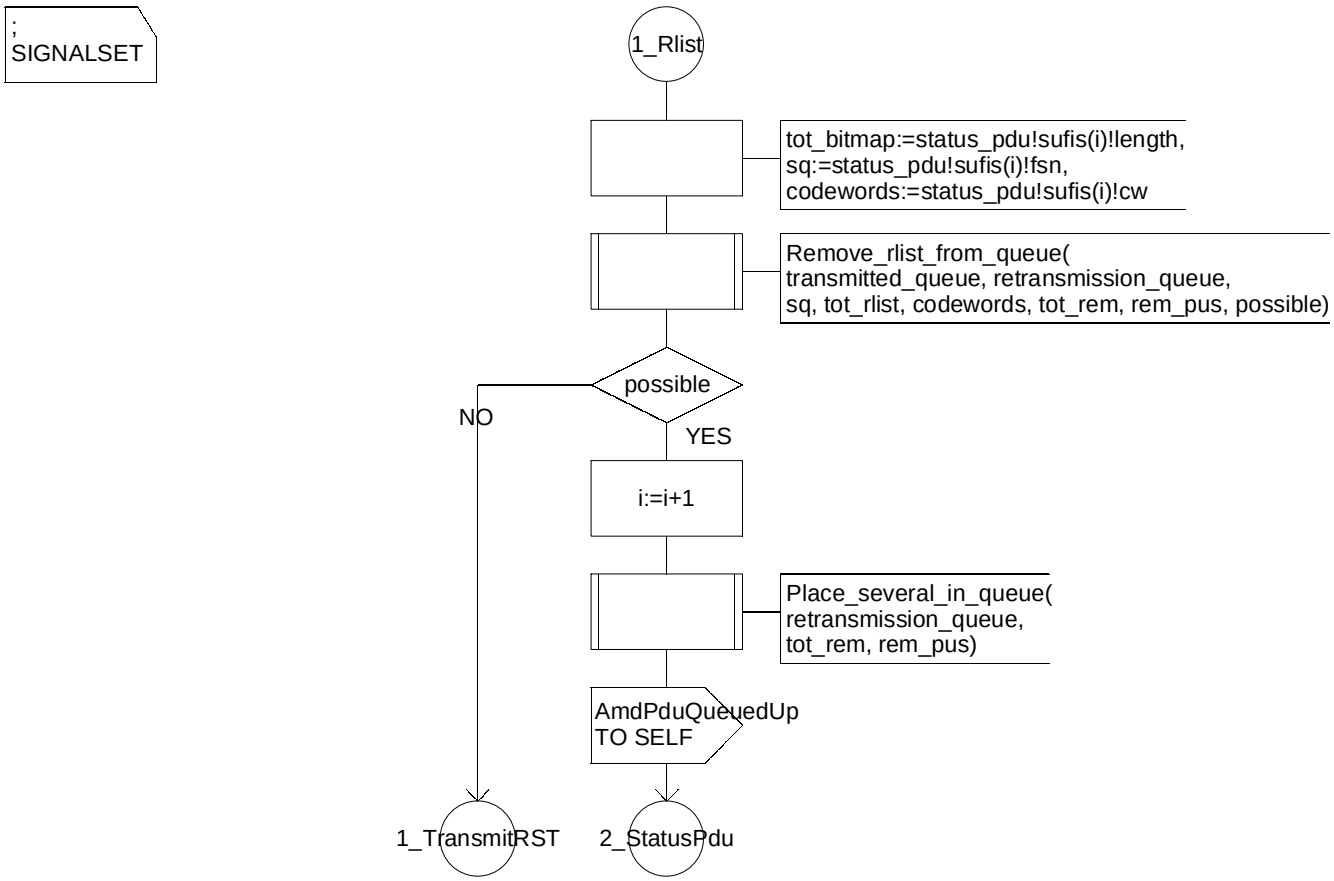
Virtual Process Type Acknowledged_link 1_StatusPduList(69)



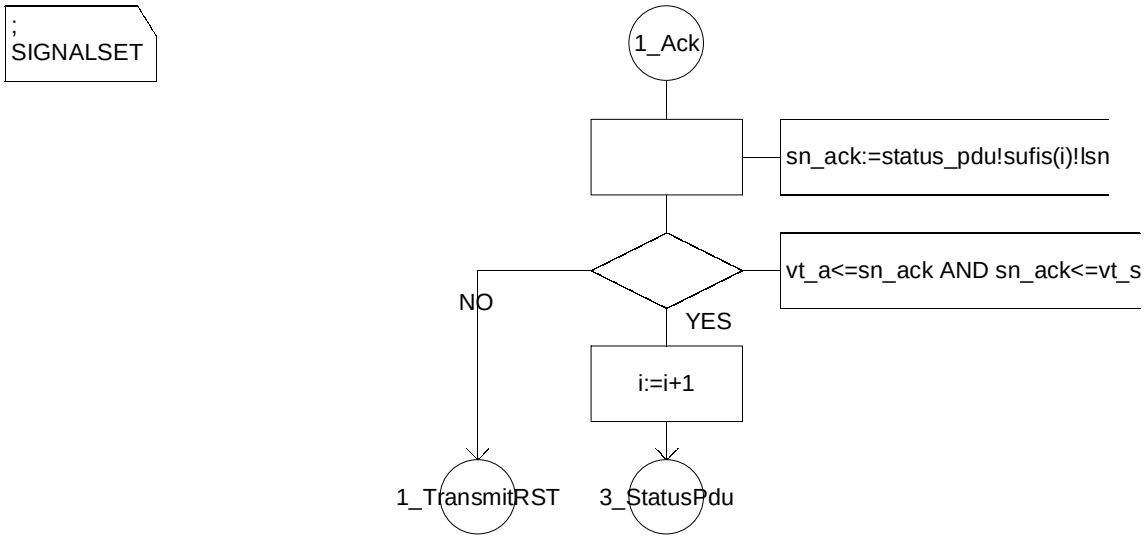
Virtual Process Type Acknowledged_link 1_StatusPduBitmap(69)



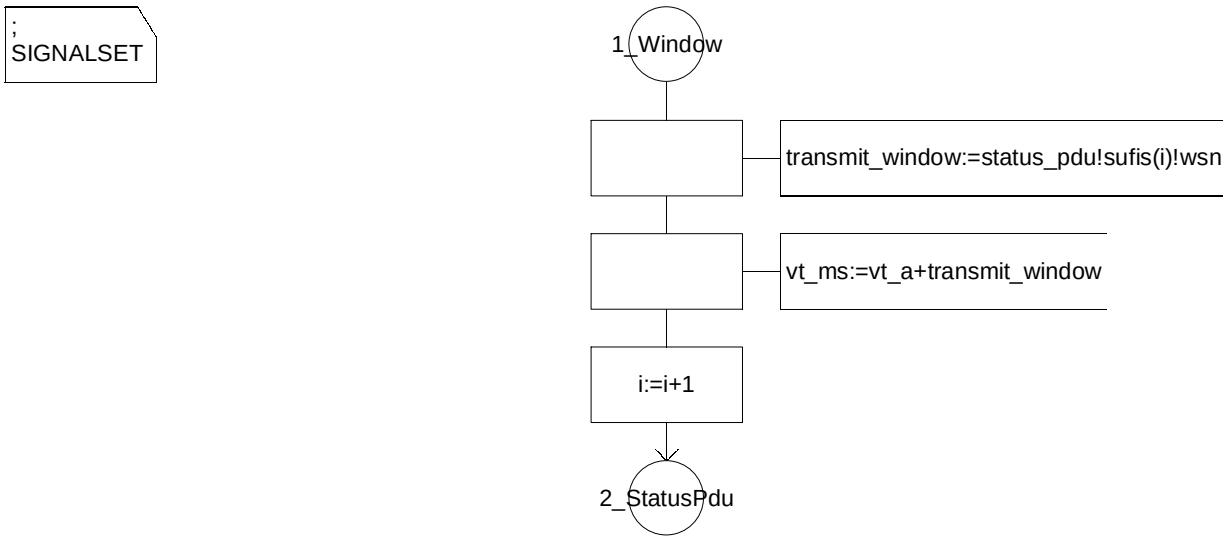
Virtual Process Type Acknowledged_link 1_StatusPduRlist(69)



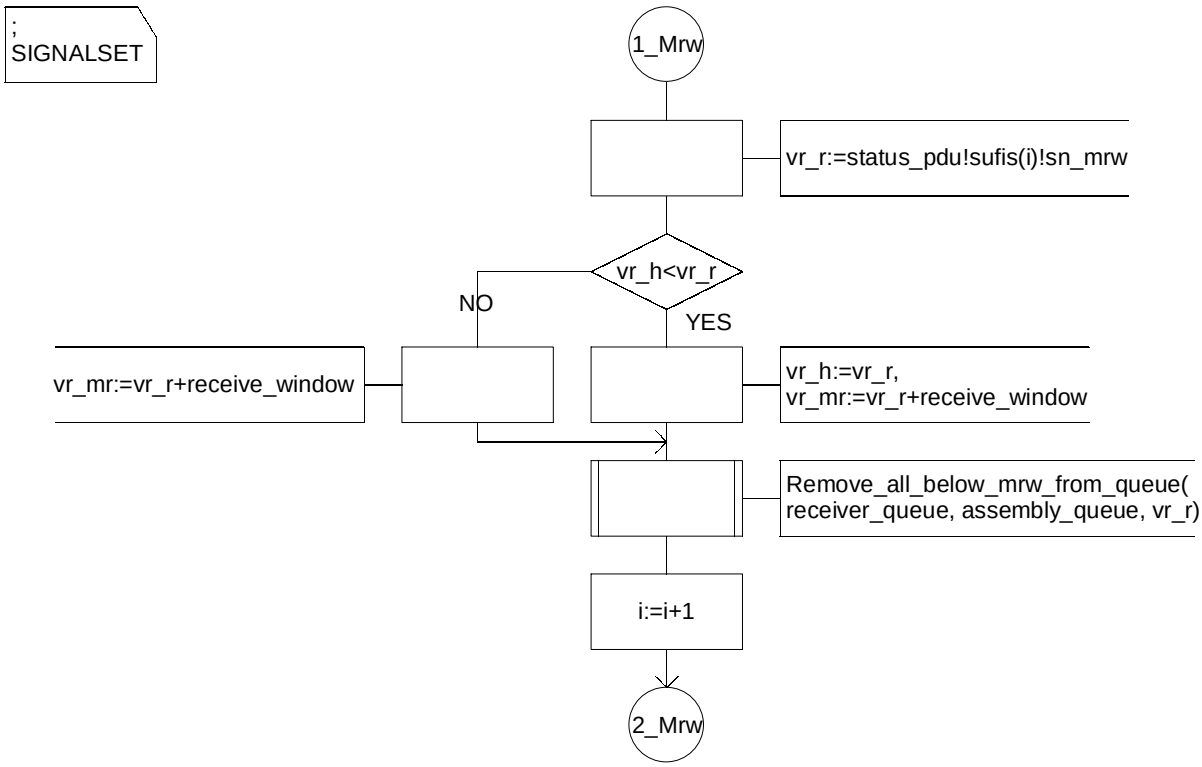
Virtual Process Type Acknowledged_link 1_StatusPduAck(69)

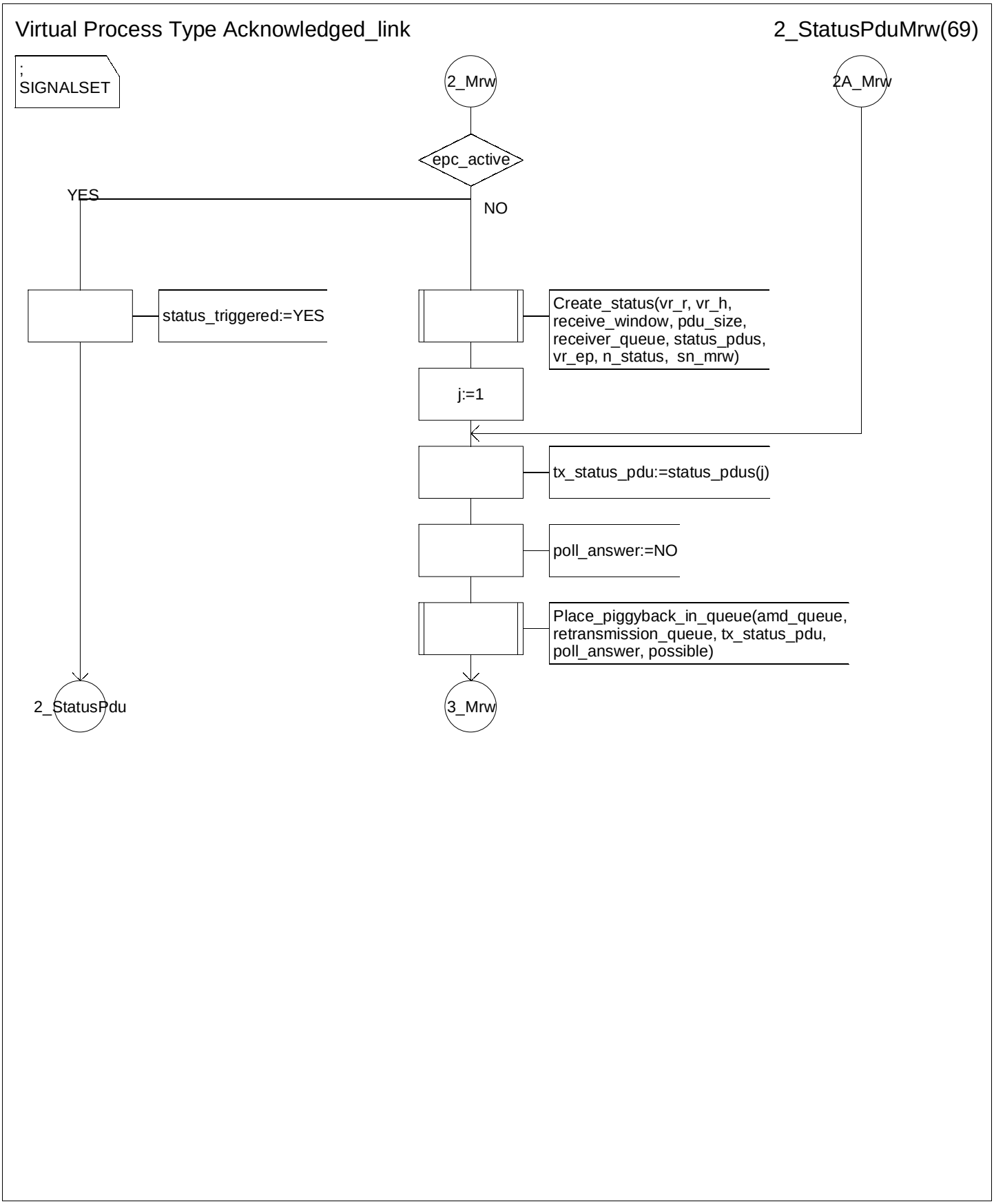


Virtual Process Type Acknowledged_link 1_StatusPduWindow(69)

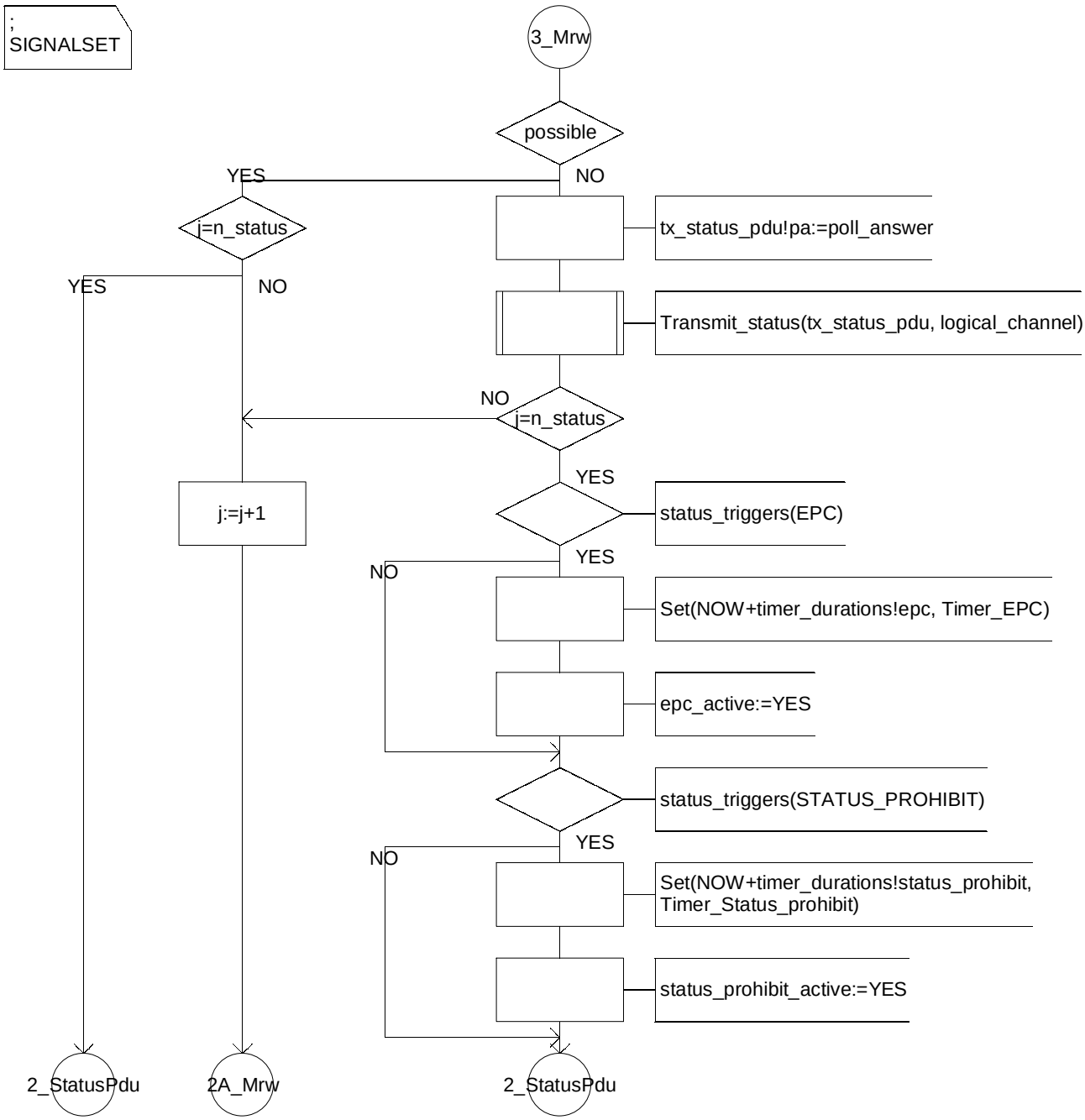


Virtual Process Type Acknowledged_link 1_StatusPduMrw(69)

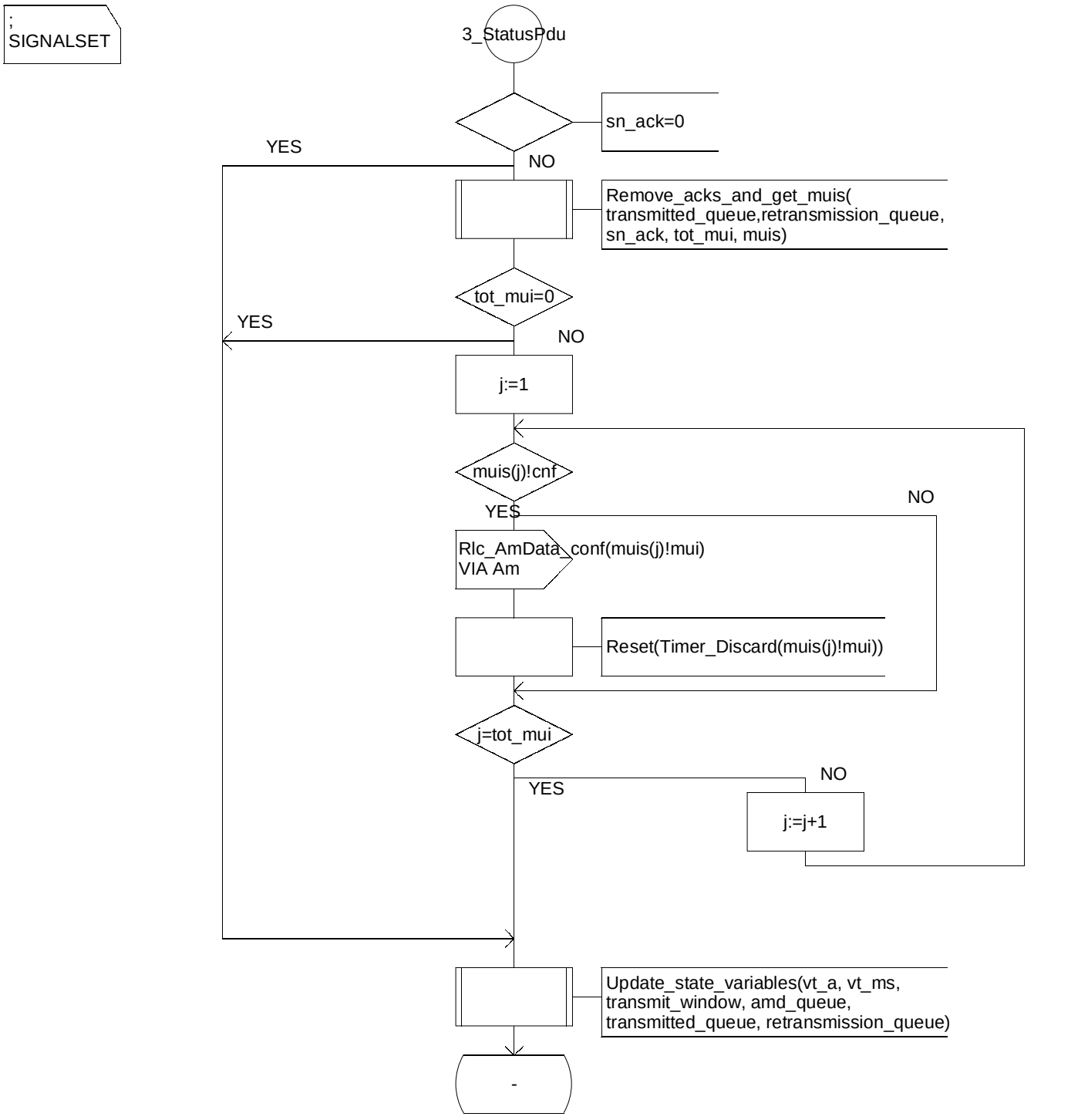


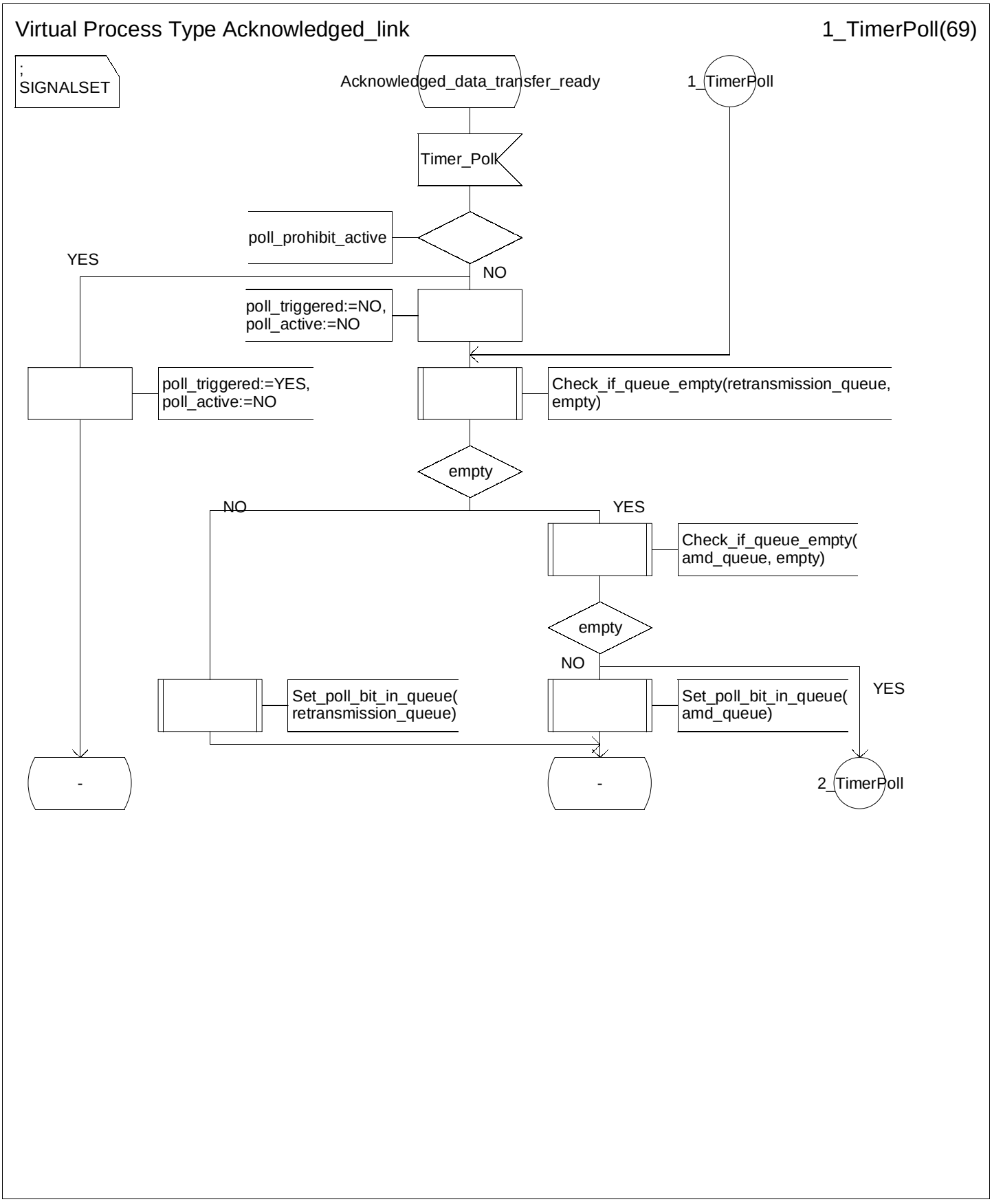

Virtual Process Type Acknowledged_link
2_StatusPduMrw(69)
;
SIGNALSET
2_Mrw
2A_Mrw
epc_active
YES
NO
status_triggered:=YES
Create_status(vr_r, vr_h,
receive_window, pdu_size,
receiver_queue, status_pdus,
vr_ep, n_status, sn_mrw)
j:=1
tx_status_pdu:=status_pdus(j)
poll_answer:=NO
Place_piggyback_in_queue(amd_queue,
retransmission_queue, tx_status_pdu,
poll_answer, possible)
2_StatusPdu
3_Mrw

Virtual Process Type Acknowledged_link 3_StatusPduMrw(69)

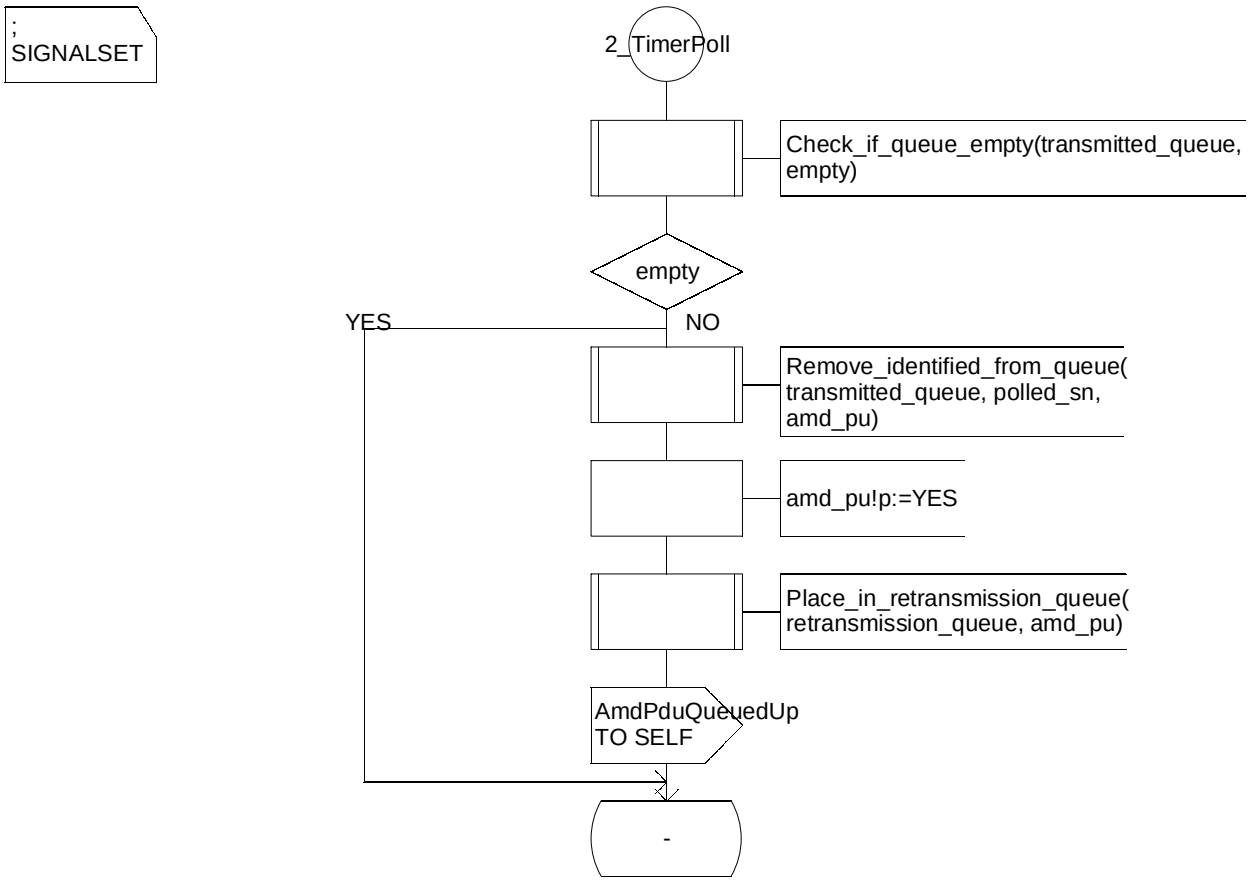


Virtual Process Type Acknowledged_link 3_StatusPdu(69)

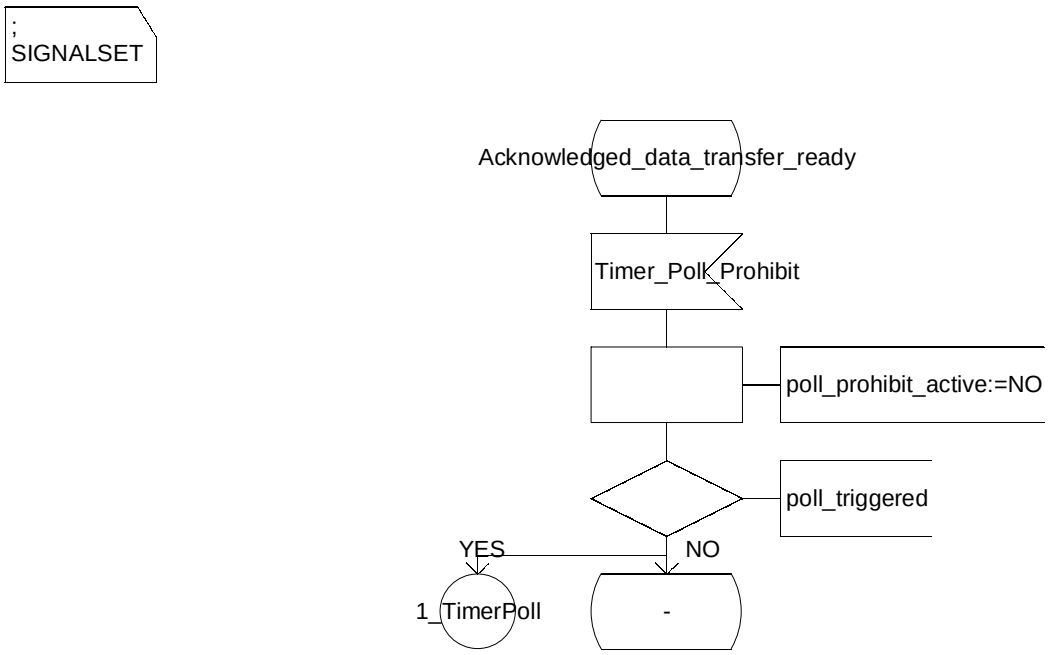



Virtual Process Type Acknowledged_link
1_TimerPoll(69)
;
SIGNALSET
Acknowledged_data_transfer_ready
1_TimerPoll
Timer_Poll
poll_prohibit_active
YES
NO
poll_triggered:=NO,
poll_active:=NO
poll_triggered:=YES,
poll_active:=NO
Check_if_queue_empty(retransmission_queue,
empty)
empty
NO
YES
Check_if_queue_empty(
amd_queue, empty)
empty
NO
YES
Set_poll_bit_in_queue(
retransmission_queue)
Set_poll_bit_in_queue(
amd_queue)
-
-
2_TimerPoll

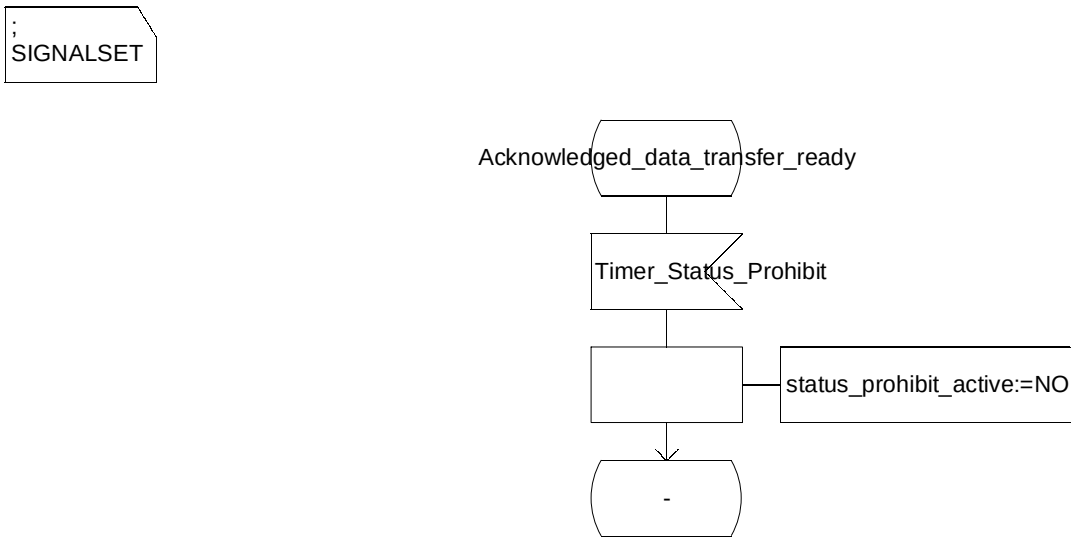
Virtual Process Type Acknowledged_link 2_TimerPoll(69)



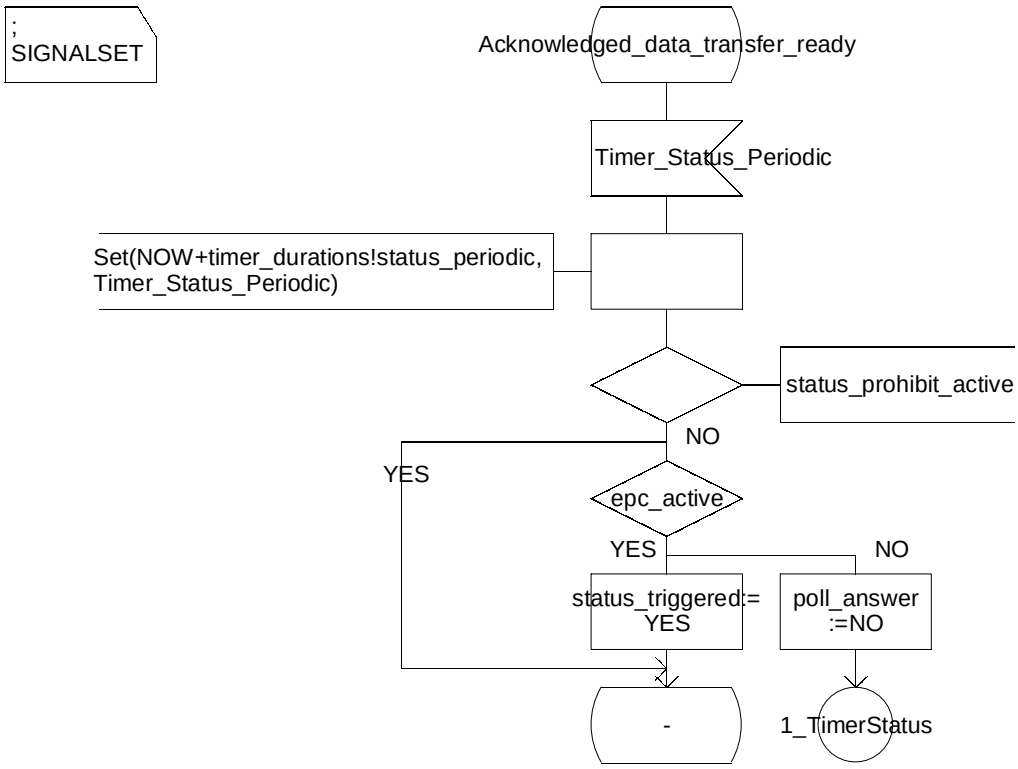
Virtual Process Type Acknowledged_link 1_TimerPollProhibit(69)



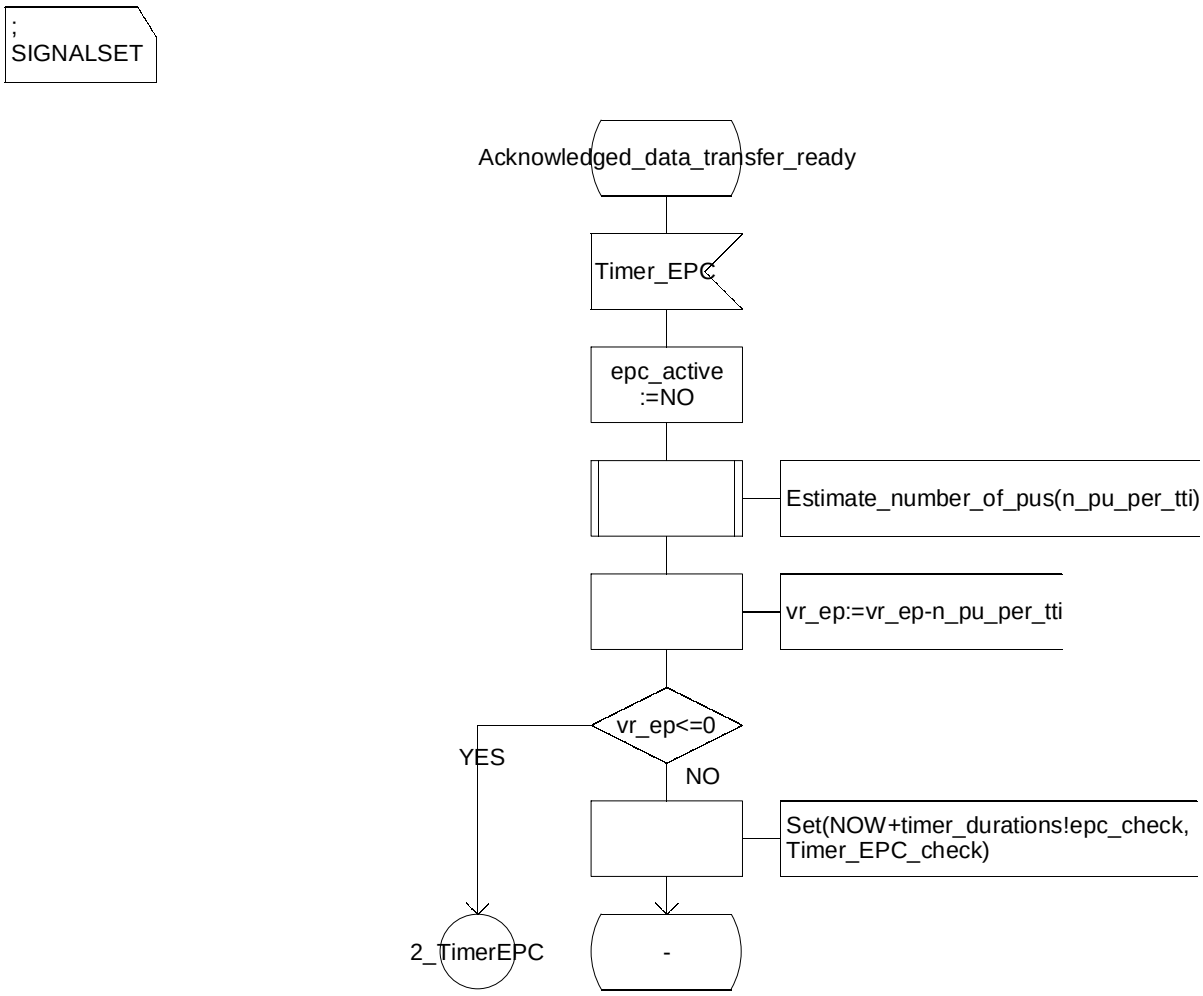
Virtual Process Type Acknowledged_link 1_TimerStatusProhibit(69)

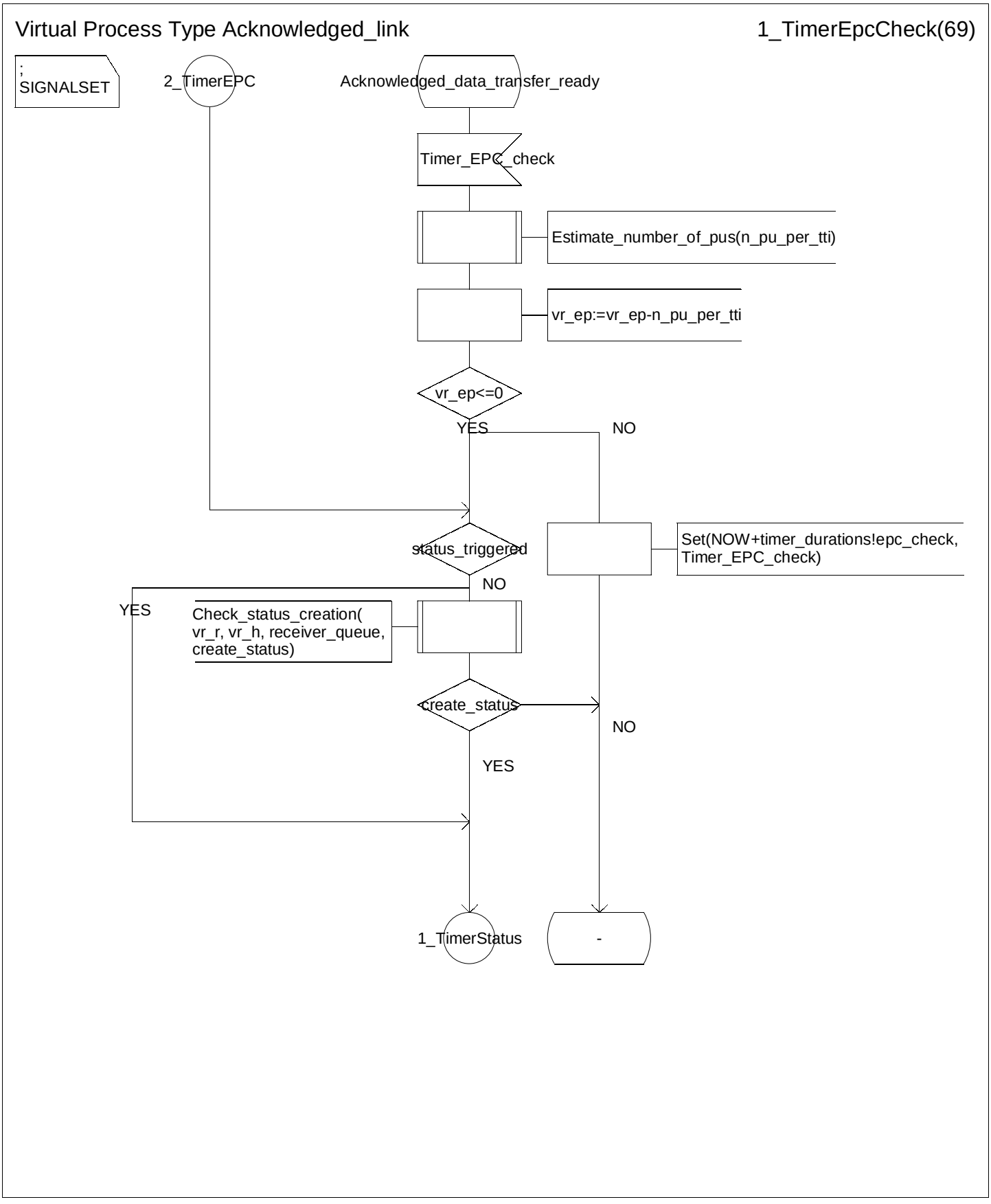


Virtual Process Type Acknowledged_link 1_TimerStatusPeriodic(69)

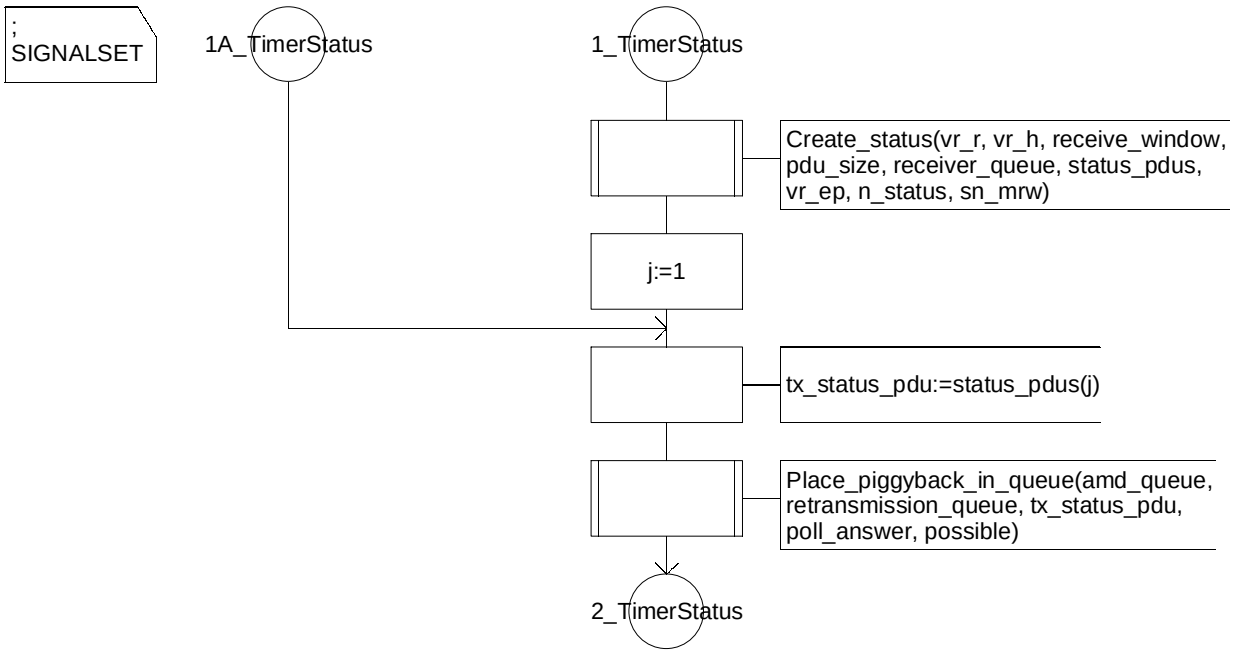


Virtual Process Type Acknowledged_link 1_TimerEpc(69)

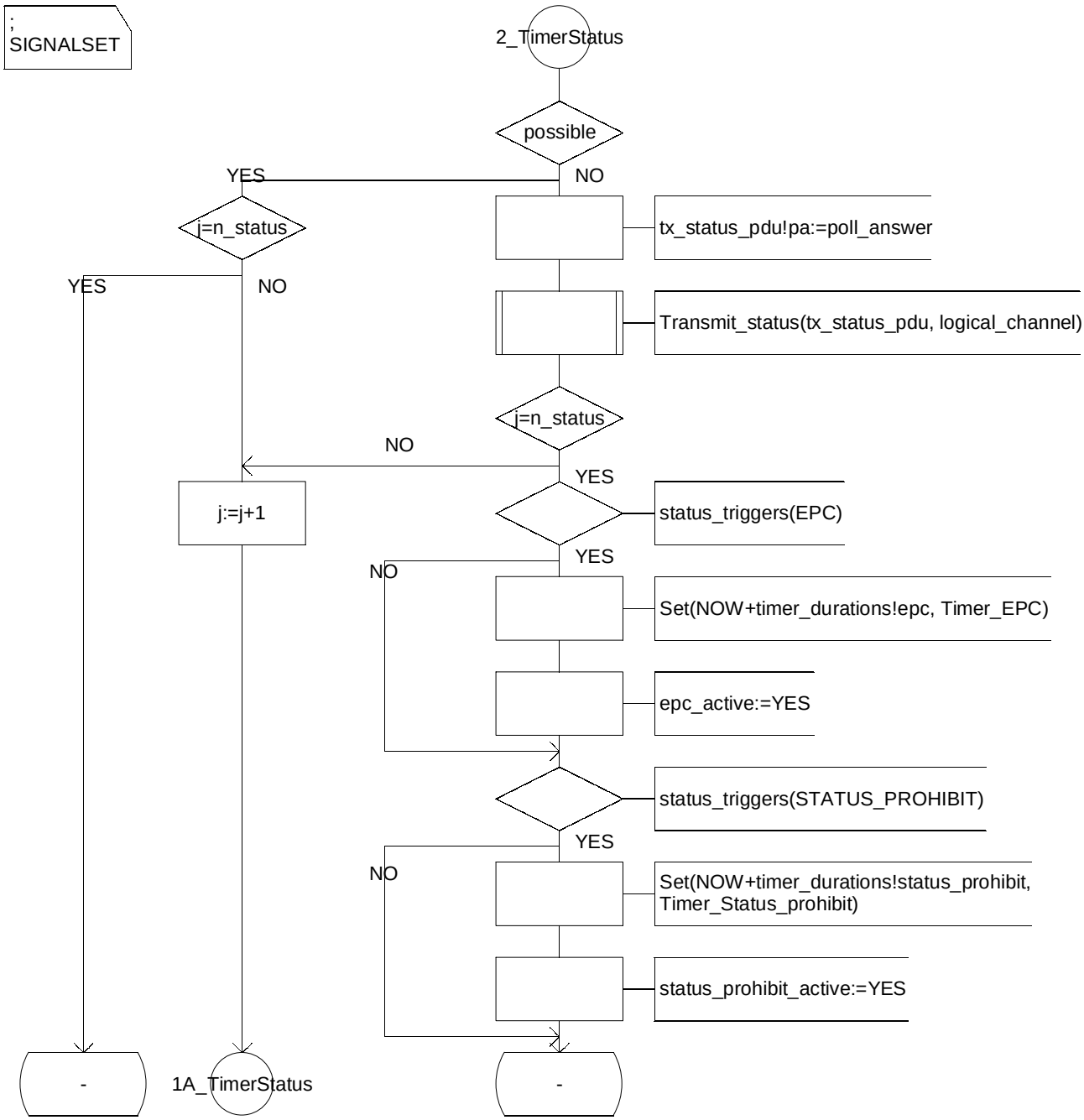



Virtual Process Type Acknowledged_link
1_TimerEpcCheck(69)
;
SIGNALSET
2_TimerEPC
Acknowledged_data_transfer_ready
Timer_EPC_check
Estimate_number_of_pus(n_pu_per_tti)
vr_ep:=vr_ep-n_pu_per_tti
vr_ep<=0
YES
NO
status_triggered
Set(NOW+timer_durations!epc_check,
Timer_EPC_check)
NO
YES
Check_status_creation(
vr_r, vr_h, receiver_queue,
create_status)
create_status
NO
YES
1_TimerStatus
-

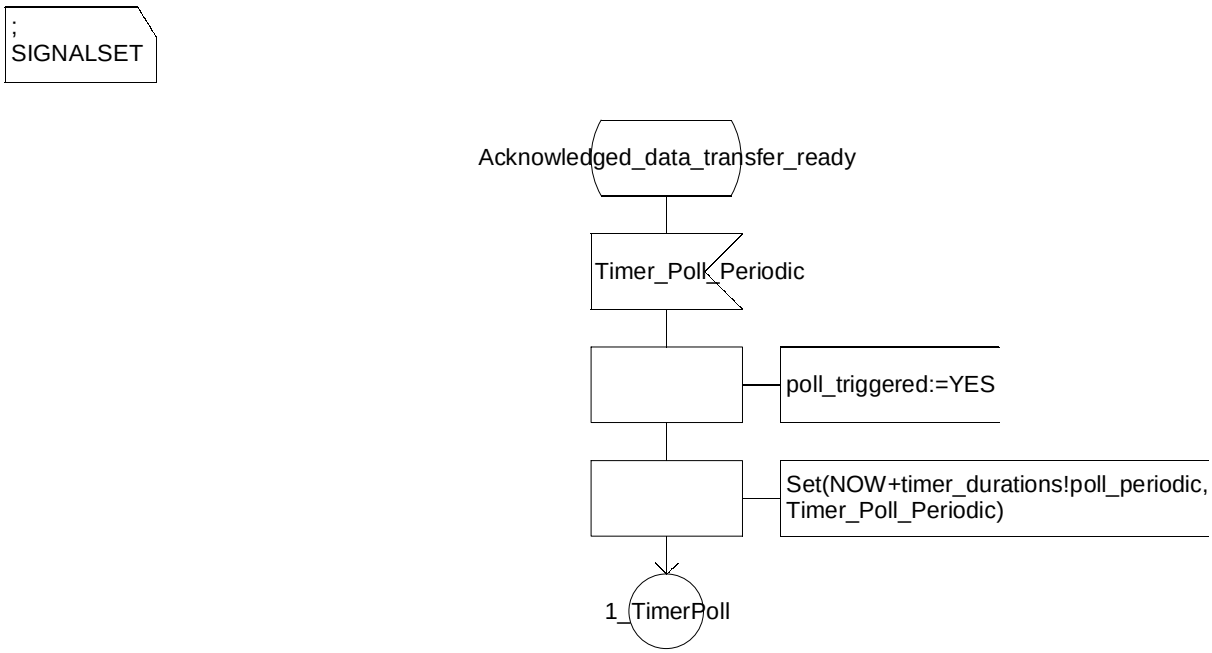
Virtual Process Type Acknowledged_link 1_TimerStatus(69)

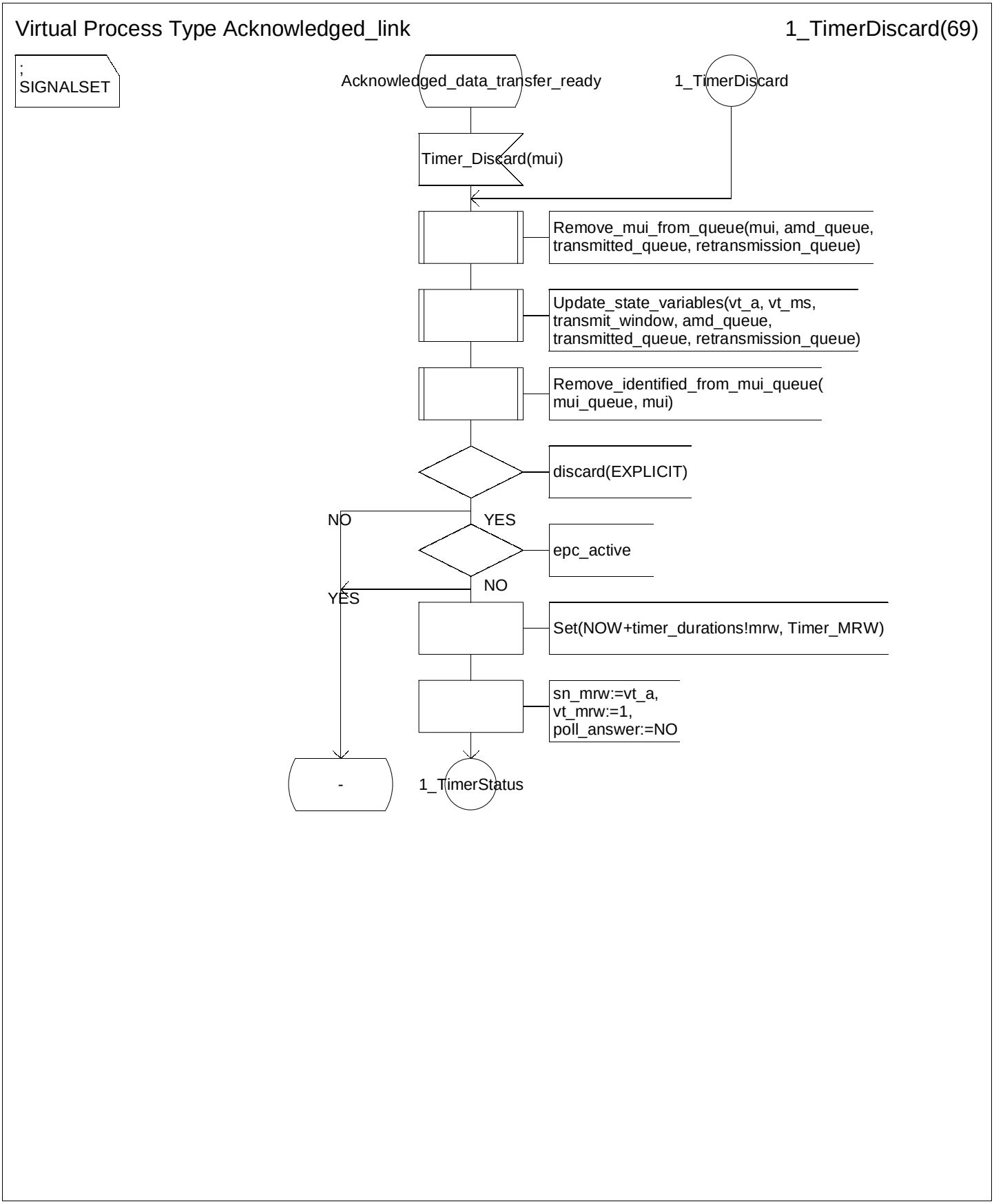


Virtual Process Type Acknowledged_link 2_TimerStatus(69)

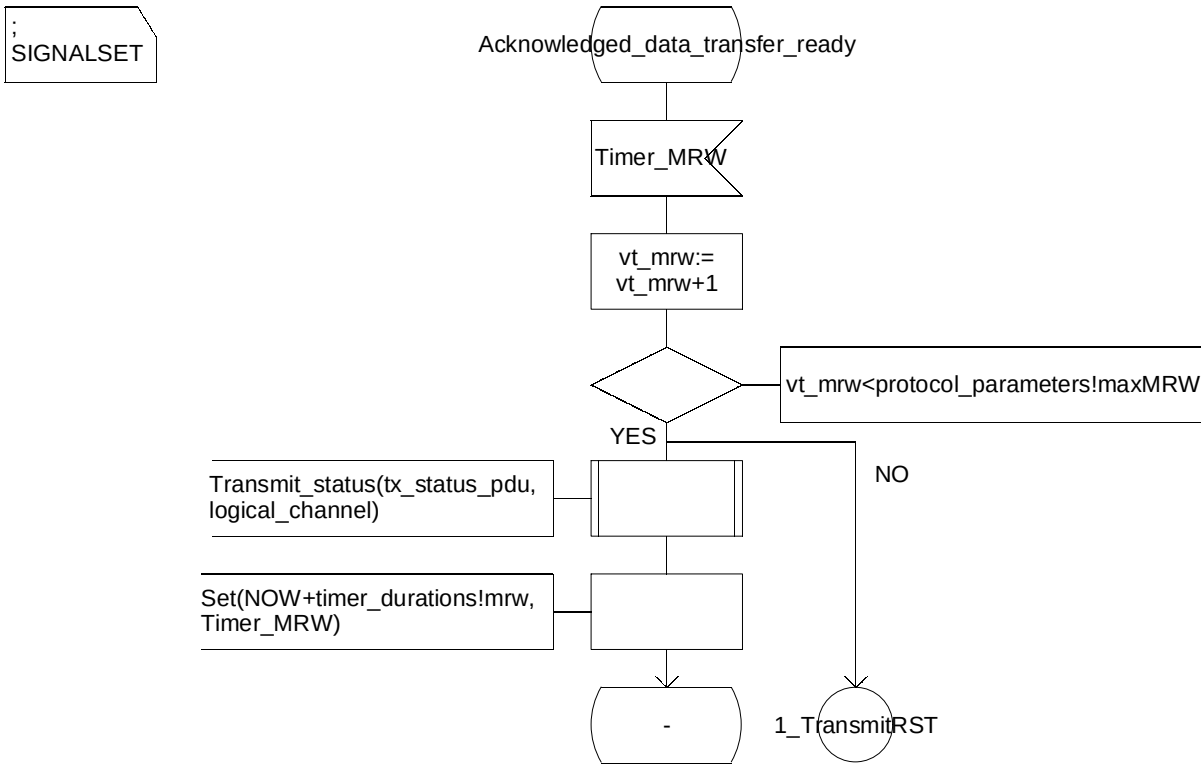


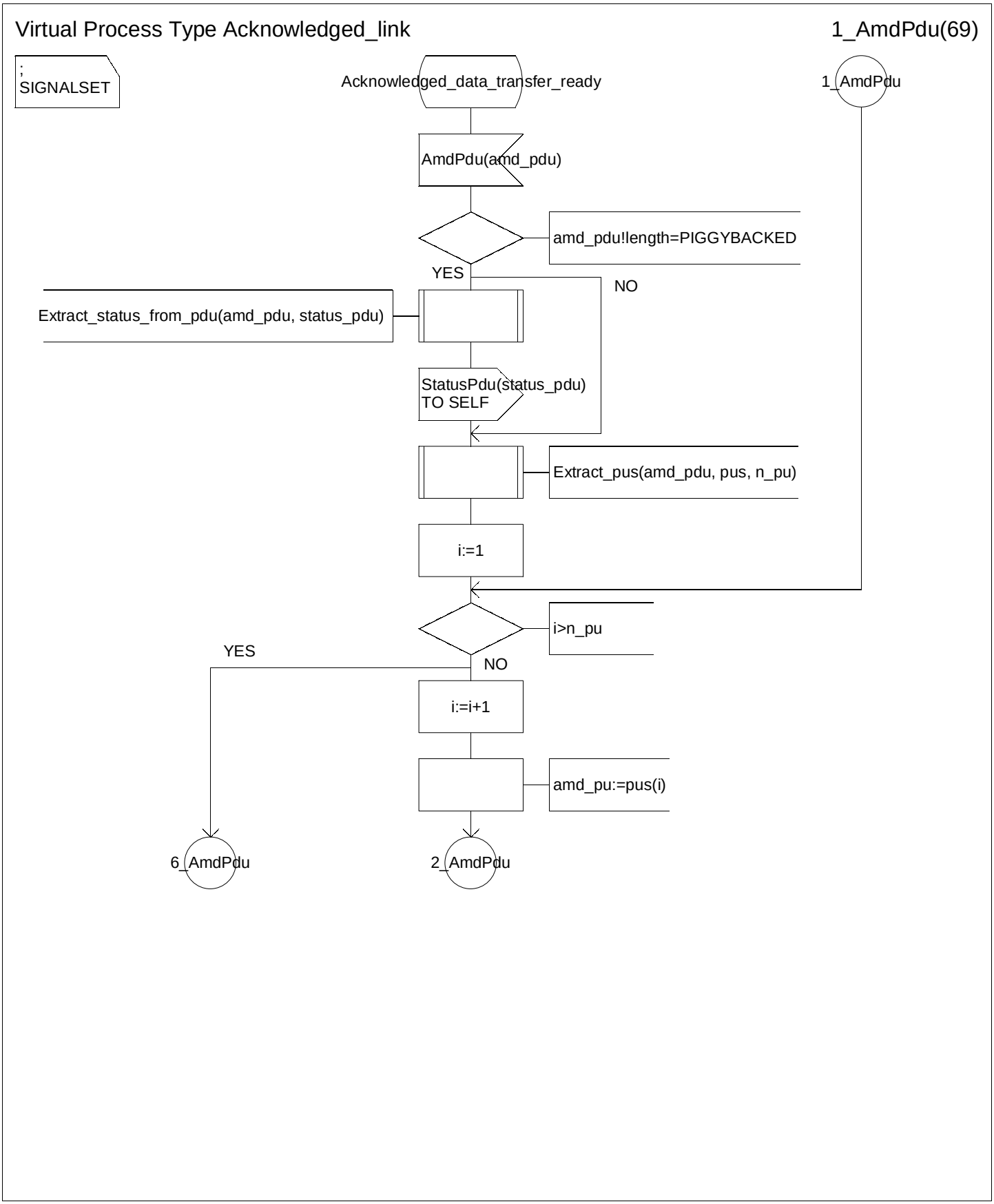
Virtual Process Type Acknowledged_link 1_TimerPollPeriodic(69)

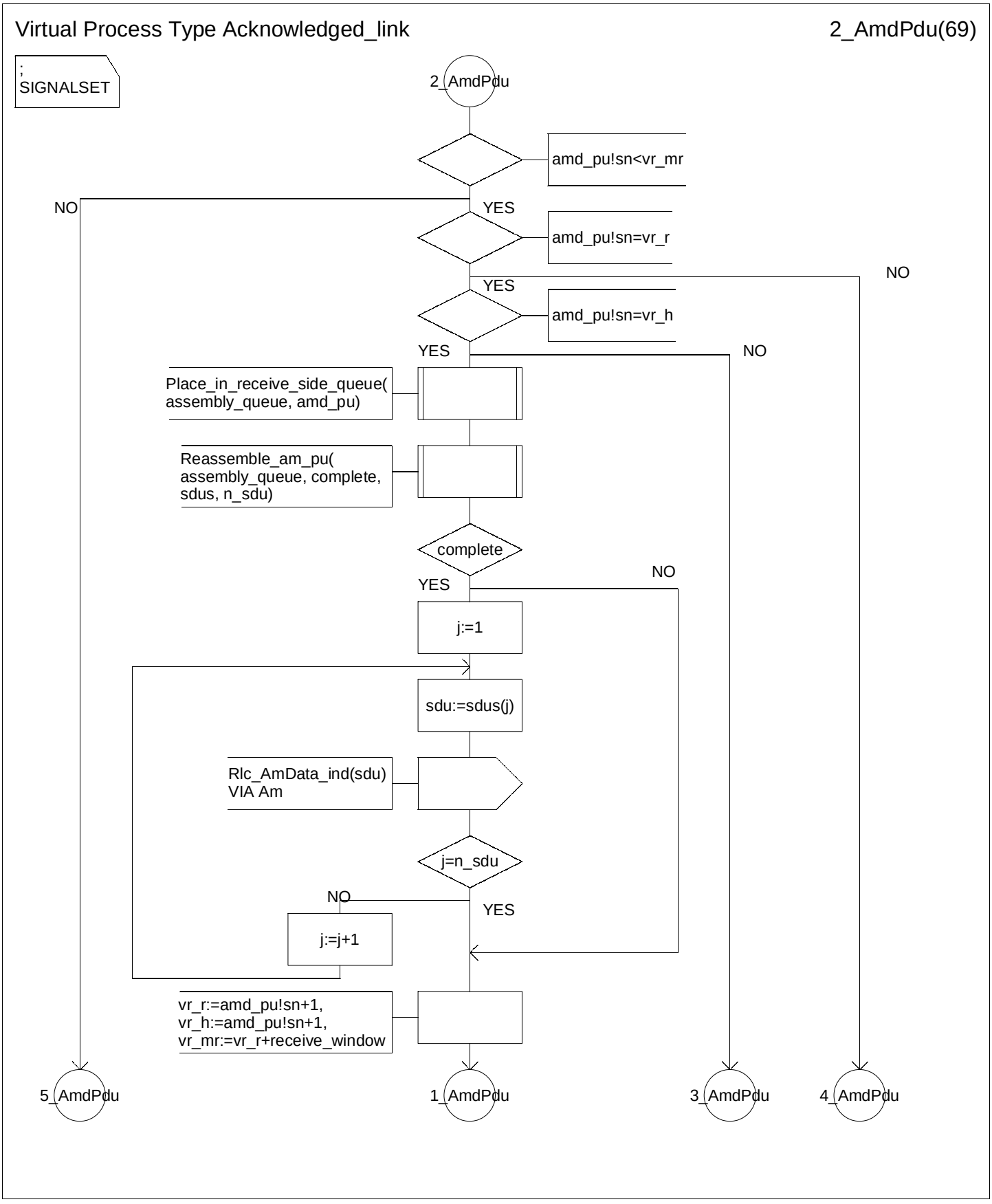


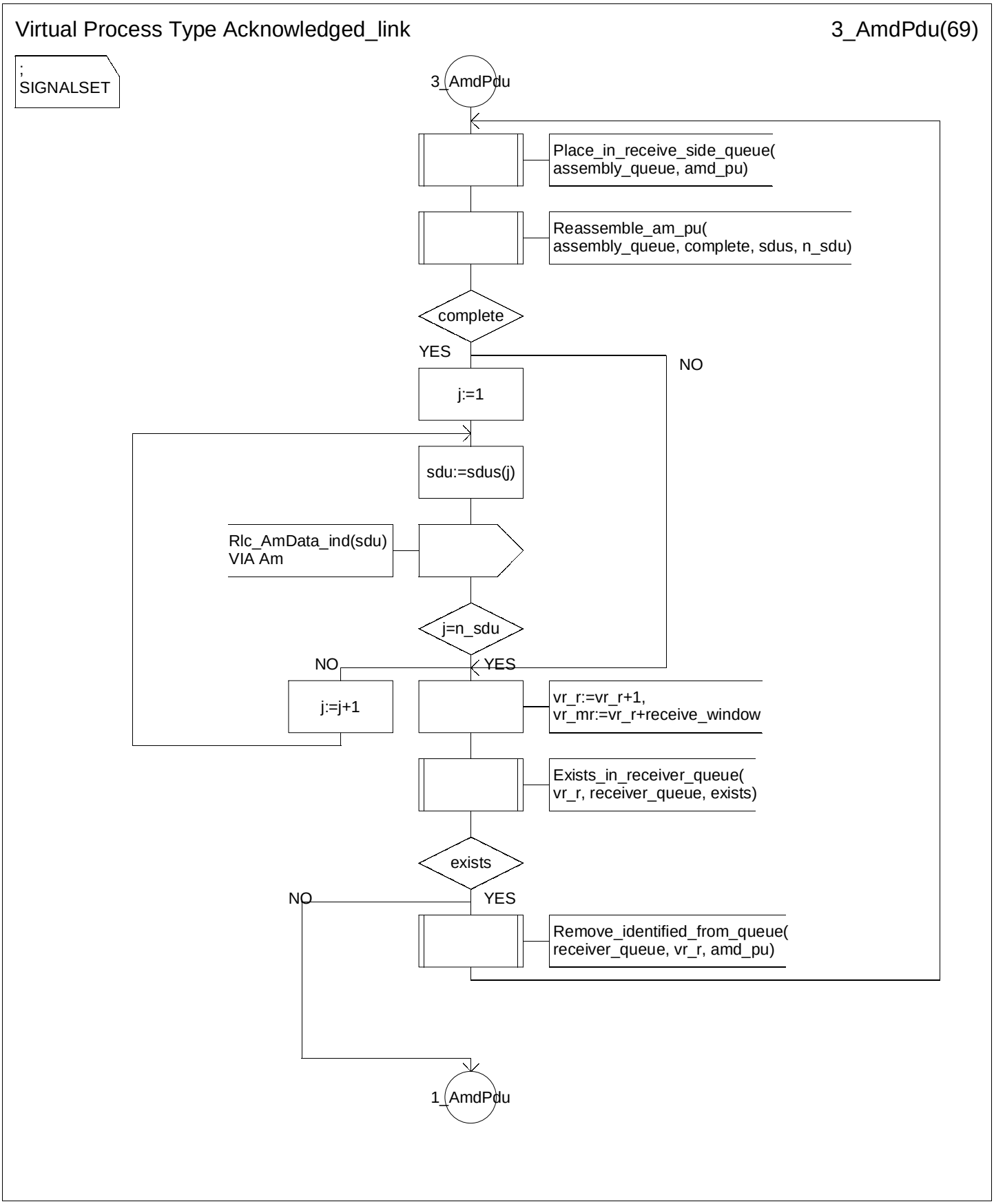

Virtual Process Type Acknowledged_link
1_TimerDiscard(69)
;
SIGNALSET
Acknowledged_data_transfer_ready
1_TimerDiscard
Timer_Discard(mui)
Remove_mui_from_queue(mui, amd_queue, transmitted_queue, retransmission_queue)
Update_state_variables(vt_a, vt_ms, transmit_window, amd_queue, transmitted_queue, retransmission_queue)
Remove_identified_from_mui_queue(mui_queue, mui)
discard(EXPLICIT)
NO
YES
epc_active
YES
NO
Set(NOW+timer_durations!mrw, Timer_MRW)
sn_mrw:=vt_a, vt_mrw:=1, poll_answer:=NO
-
1_TimerStatus

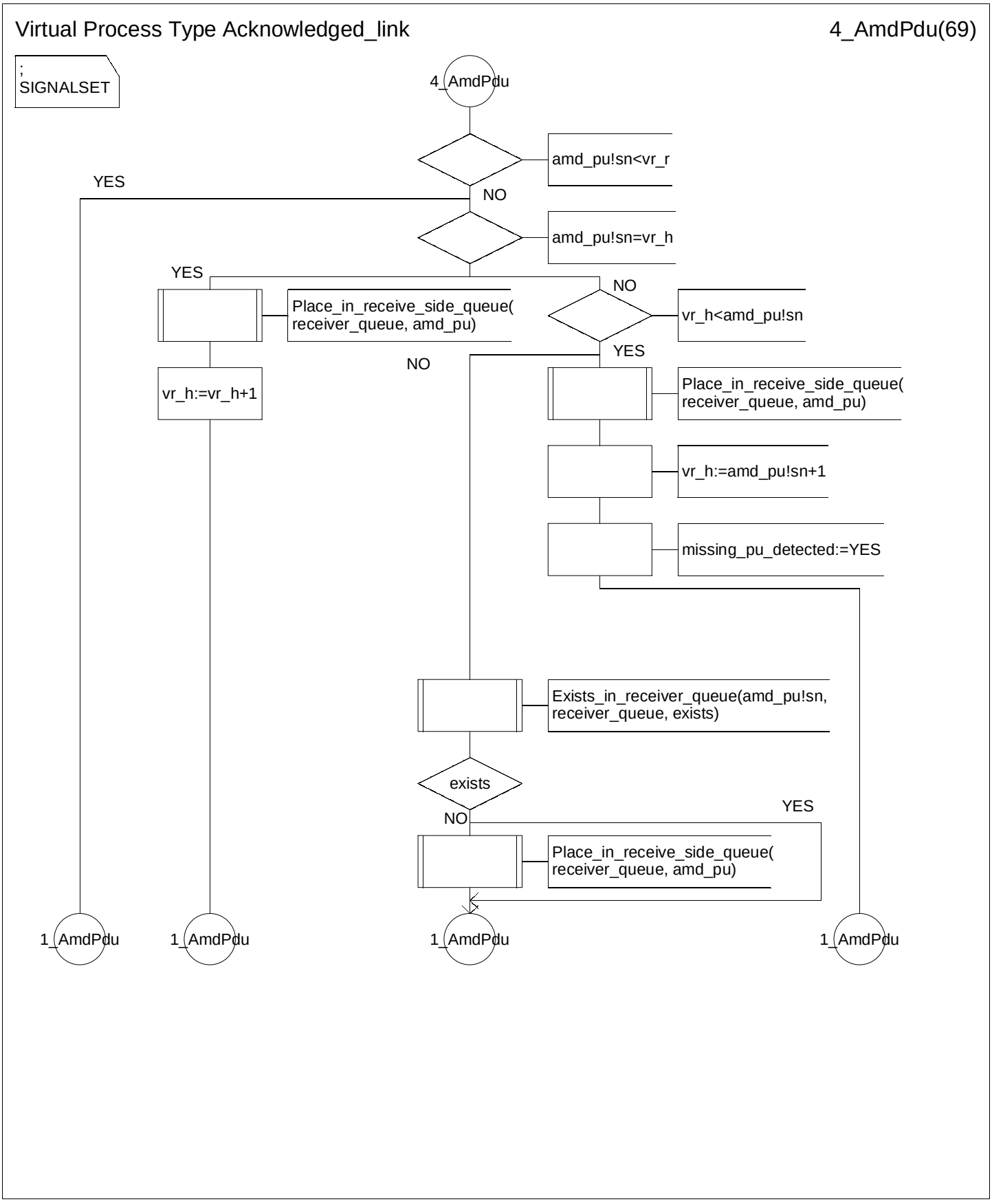
Virtual Process Type Acknowledged_link 1_TimerMRW(69)



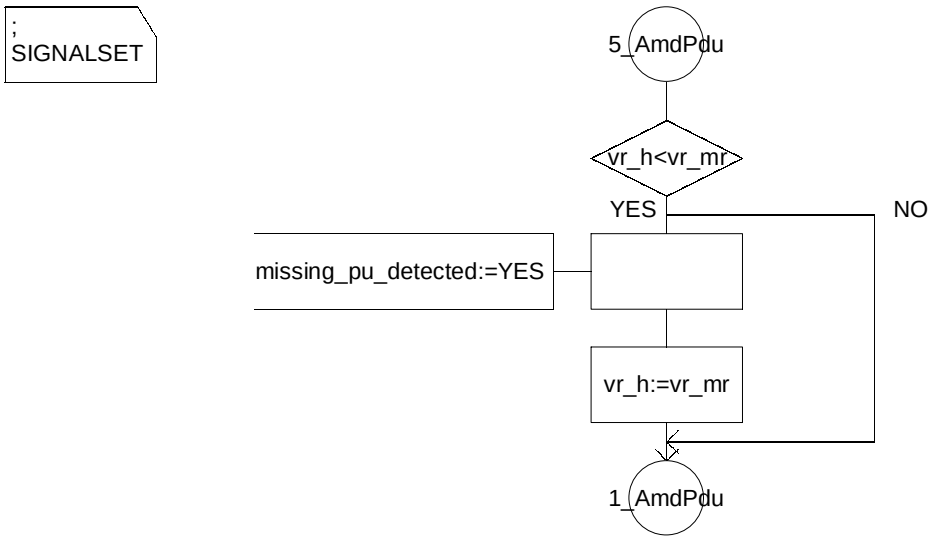

Virtual Process Type Acknowledged_link
1_AmdPdu(69)
;
SIGNALSET
Acknowledged_data_transfer_ready
1_AmdPdu
AmdPdu(amd_pdu)
amd_pdu!length=PIGGYBACKED
YES
NO
Extract_status_from_pdu(amd_pdu, status_pdu)
StatusPdu(status_pdu)
TO SELF
Extract_pus(amd_pdu, pus, n_pu)
i:=1
i>n_pu
YES
NO
i:=i+1
amd_pu:=pus(i)
6_AmdPdu
2_AmdPdu


Virtual Process Type Acknowledged_link
2_AmdPdu(69)
;
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_receive_side_queue(
assembly_queue, amd_pu)
Reassemble_am_pu(
assembly_queue, complete,
sdus, n_sdu)
complete
NO
YES
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

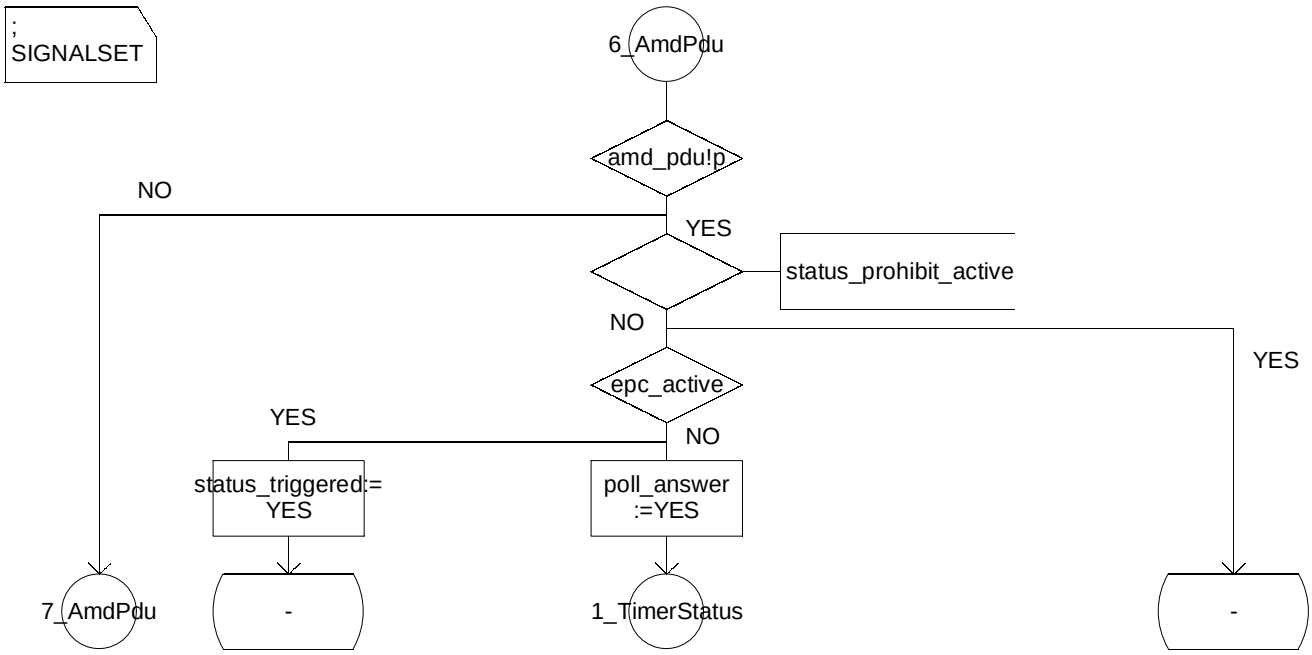

Virtual Process Type Acknowledged_link
3_AmdPdu(69)
;
SIGNALSET
3_AmdPdu
Place_in_receive_side_queue(
assembly_queue, amd_pu)
Reassemble_am_pu(
assembly_queue, complete, sdus, n_sdu)
complete
YES
NO
j:=1
sdu:=sdus(j)
Rlc_AmData_ind(sdu)
VIA Am
j=n_sdu
NO
YES
j:=j+1
vr_r:=vr_r+1,
vr_mr:=vr_r+receive_window
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(69)
;
SIGNALSET
4_AmdPdu
amd_pu!sn<vr_r
YES
NO
amd_pu!sn=vr_h
YES
NO
Place_in_receive_side_queue(
receiver_queue, amd_pu)
vr_h<amd_pu!sn
NO
YES
vr_h:=vr_h+1
Place_in_receive_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
YES
NO
Place_in_receive_side_queue(
receiver_queue, amd_pu)
1_AmdPdu
1_AmdPdu
1_AmdPdu
1_AmdPdu

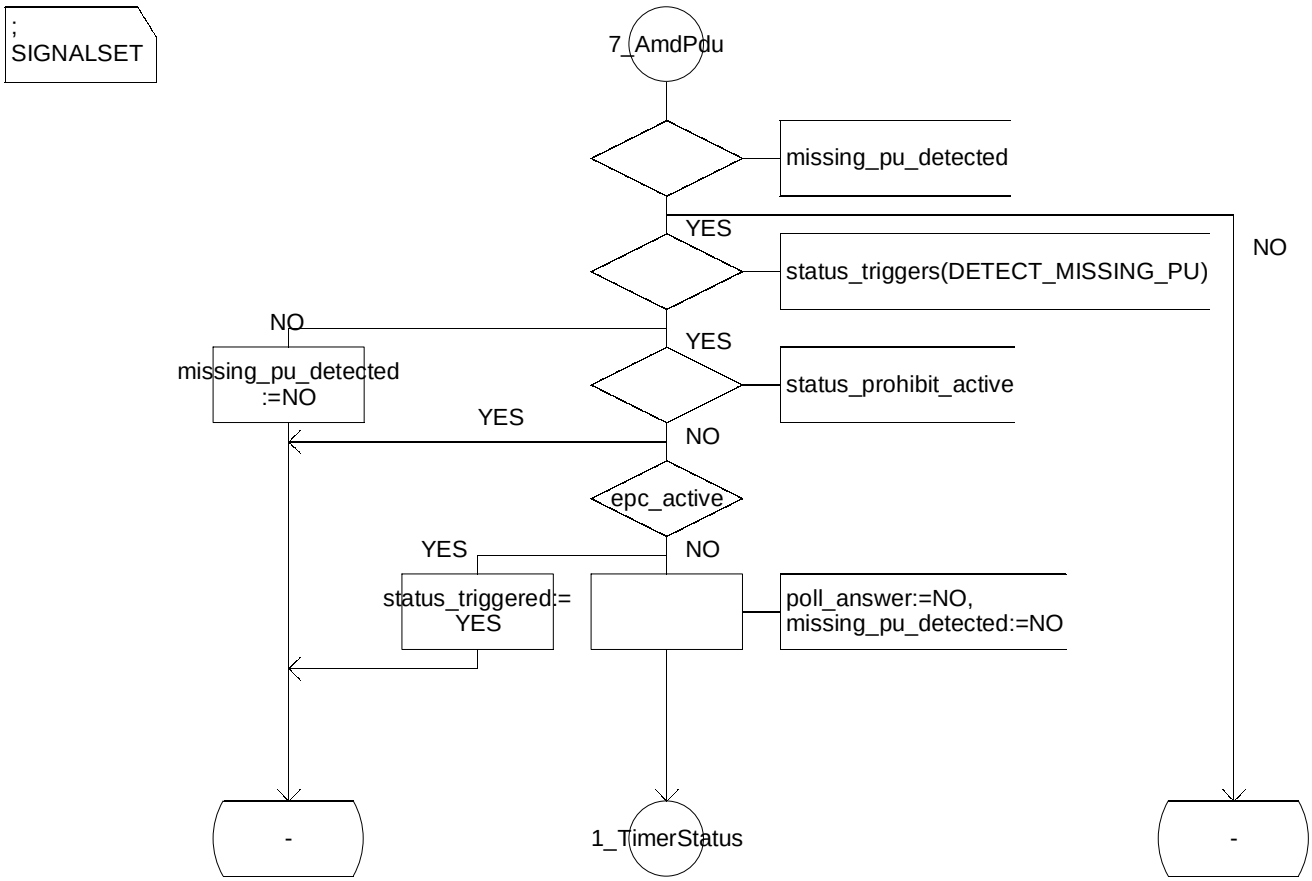
Virtual Process Type Acknowledged_link 5_AmdPdu(69)

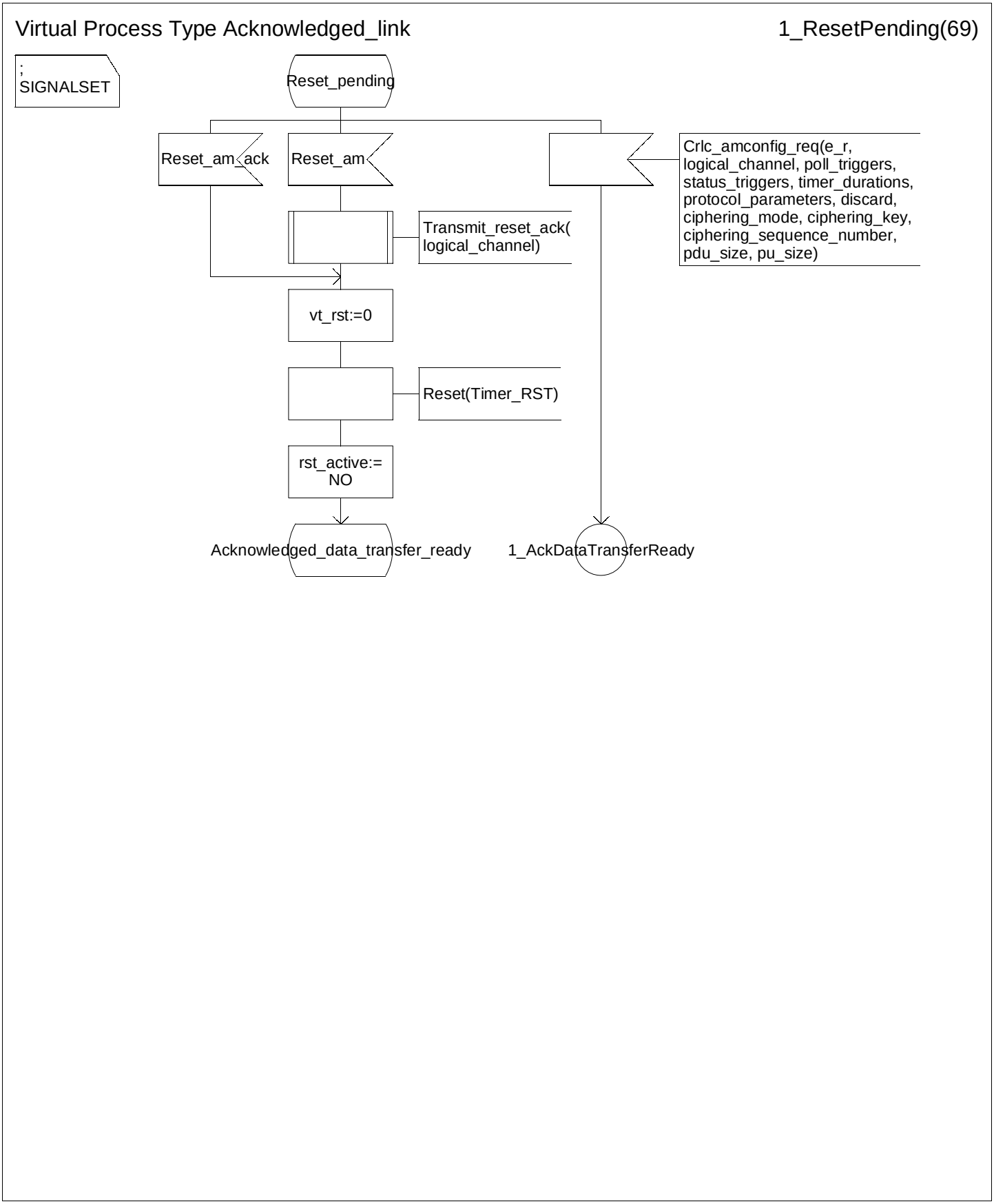


Virtual Process Type Acknowledged_link 6_AmdPdu(69)

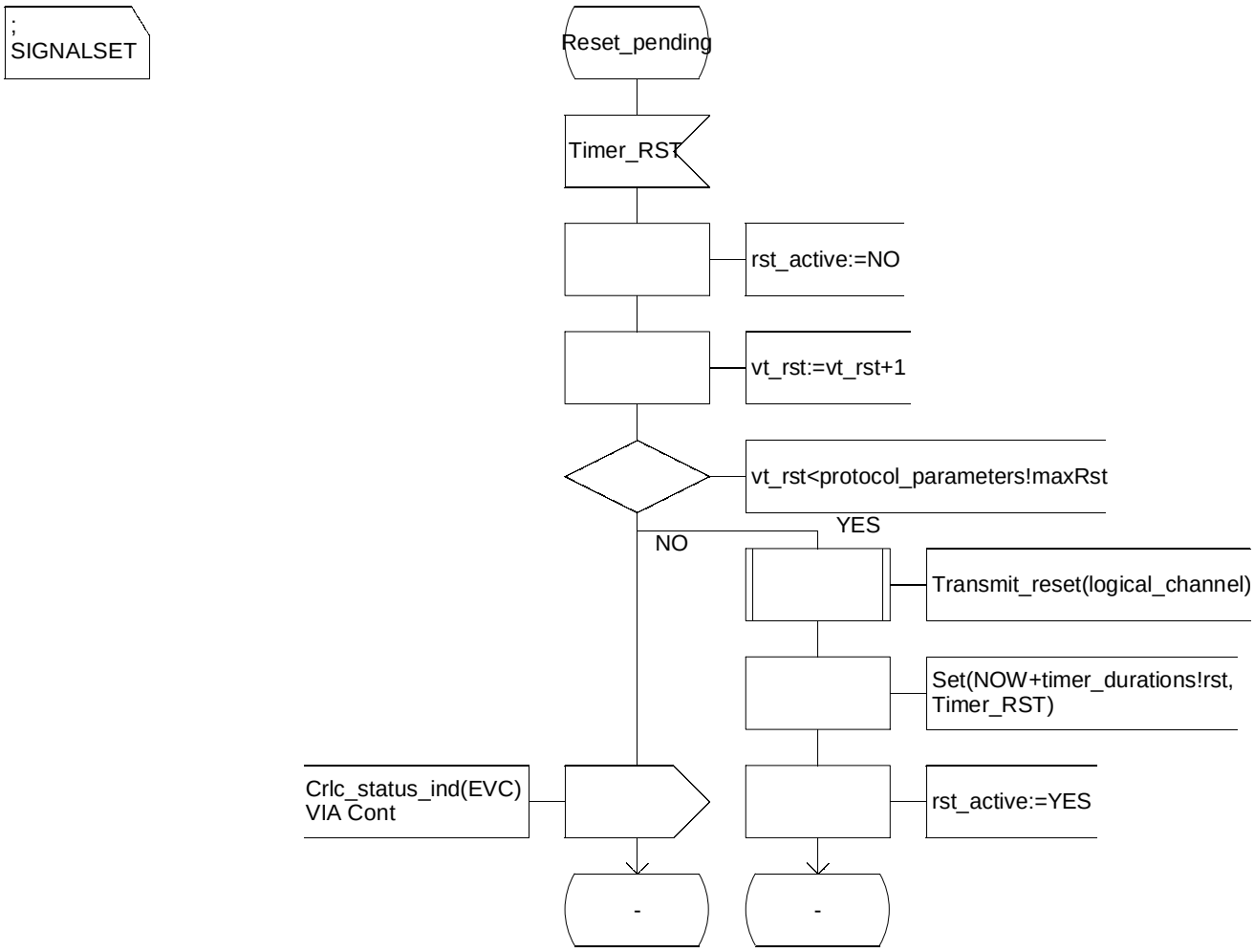


Virtual Process Type Acknowledged_link 7_AmdPdu(69)




Virtual Process Type Acknowledged_link
1_ResetPending(69)
;
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,
ciphering_sequence_number,
pdu_size, pu_size)
Transmit_reset_ack(
logical_channel)
vt_rst:=0
Reset(Timer_RST)
rst_active:=
NO
Acknowledged_data_transfer_ready
1_AckDataTransferReady

Virtual Process Type Acknowledged_link 2_ResetPending(69)



Virtual Process Type Acknowledged_connection 1_Declarations(44)

;

```
DCL

am_pdu, tmp, pdu                                AmPdu,
/*A representation of data contained within a AmPdu.*/

stat_pdu, rx_stat_pdu                           StatPdu,
/*A representation of data contained within a StatPdu.*/

pdus, rem_pdus                                  AmPduArrayType,
/*The initally segmented sdu.*/

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.*/

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

stat_queue                                      Queue,
/* Queues used for PDUs associated with STATUS Pdus to be transmitted.*/

prohibit , rx_prohibit, epc_active              IndicatorType,
/*An indicator used to determine whether the timer_PROHIBIT
  is running or not.*/

empty, no_tx, no_retx                           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.*/

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

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

rx_period                                       DURATION,
/*The duration of a periodic Statut report generation timer.*/

rx_prohibdur, epc_dur                           DURATION,
 /*The duration of a prohibit retransmission of status report timer.*/

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

complete, cnf                                   IndicatorType;
 /*An indicator used to determine whether an SDU has been
   completely reassembled or whether an SDU requires confirmation.*/
```

Virtual Process Type Acknowledged_connection 2_Declarations(44)

;

```
DCL
 period                                          DURATION,
/*The duration of a periodic Polling generation timer.*/

 retransmission                    IndicatorType,
/*An indicator used to determine whether the received PDU is a retransmission.*/

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

 i                                          INTEGER,
/*A local counter.*/

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

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

 no_sdu, no_pu, xpu, xsdu,tx_win, rx_win, no_of_pu_per_tti,
 rx_pu, rx_sdu, muis_tot, tot, k, no_of_sq, tot_rem, l, no_s      PduIndexType,
/*Counters used to manage the amount of PUs and SDUs received.*/

 percent, rx_percent                REAL,
 /*Percentages of the transmit and receive window.*/

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

 sdus                                       OctetArrayType,
/*A set of octets.*/

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

 vt_s                                       SequenceNumberType,
/*Send state variable: The sequence number of the next PU to be transmitted for the first time.
   It is incremented after transmission of a PU for the first time (i.e. excluding retransmissions).*/

 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 upon acknowledgement of in-sequence PUs.*/

 vt_dat                                     SequenceNumberType,
/*This variable is used to count the retransmission number of each PU. It is incremented by a
   PU transmission.*/

 vt_ms                                      SequenceNumberType;
/*Maximum send state variable: This is the sequence number of the first PU not allowed by the
   receiver. It thus represents the upper edge of the transmit window. If vt_s is equal to vt_ms, now
   new PU should be transmitted. The variable is updated based on receipt of STATUS PDU.*/
```

Virtual Process Type Acknowledged_connection 3_Declarations(44)

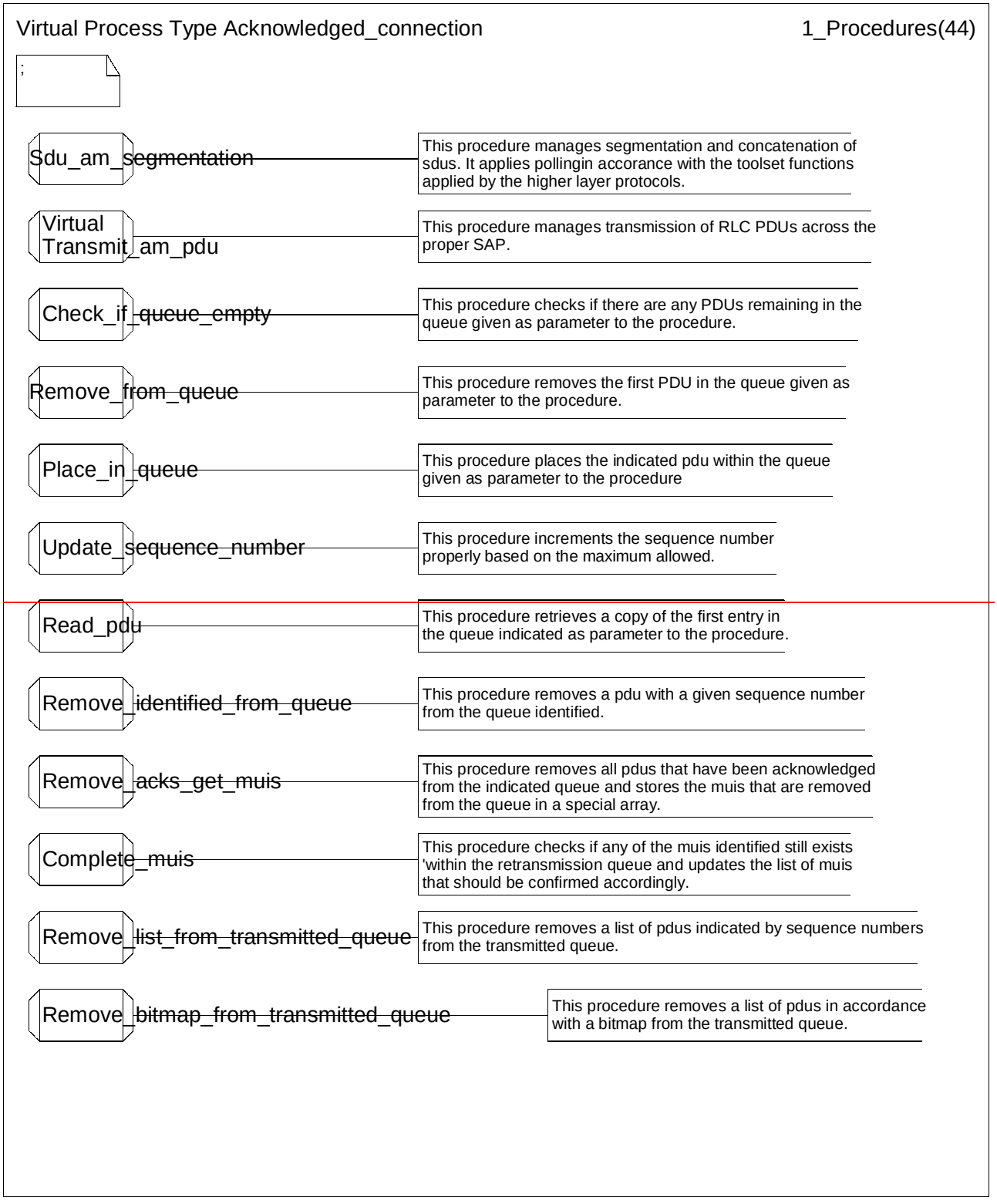
;

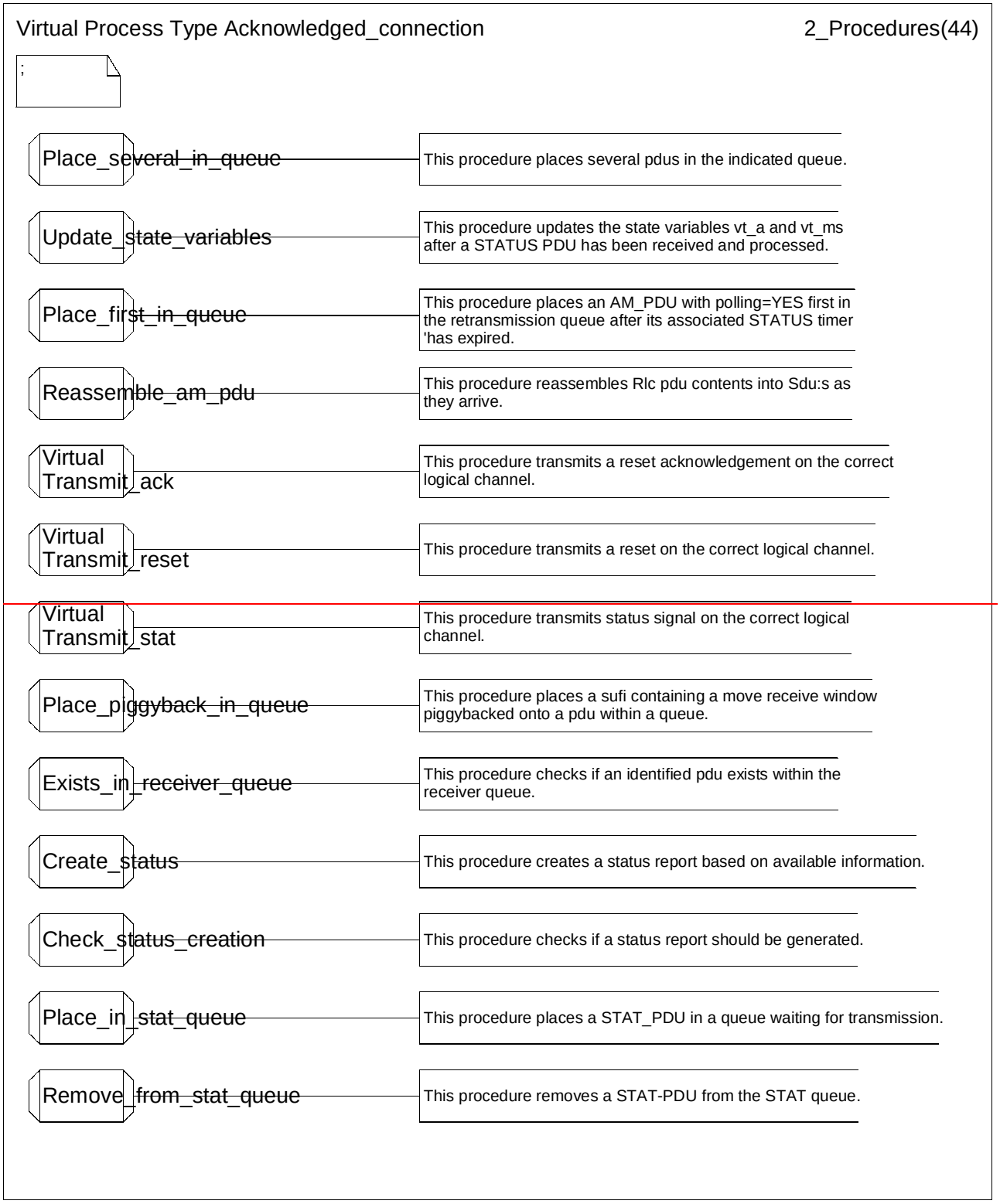
```
DCL

 vr_r                                  SequenceNumberType,
/*Receive state variable: The sequence number of the next in sequence PU expected to be received.
  It is incremented upon receipt of the next in-sequence PU.*/

vr_h                                   SequenceNumberType,
/*Highest expected state variable: The sequence number of the next highest expected PU. The variable
  is updated whenever a new PU is received.*/

vr_mr                                  SequenceNumberType,
/*Maximum acceptable receive state variable: The sequence number of the first PU not allowed by the
  receiver, thus the receiver shall discard PUs  with an n_s=vr_mr. Updating of vr_mr is implementation
  dependent but should not be set to a value less than vr_h.*/

rx_sufi_tot                            PduIndexType,
/*Local variable for maintaining knowledge of the number of super fields.*/

 tx_sufi                               SufiStructType,
 /*The contents of one superfield.*/

 rx_sufis, sufis, tx_sufis             SufiArrayStructType,
 /*The set of superfields associated with a status report.*/

flip, possible, status, rx_flip, polling_answer                IndicatorType,
/*An indicator used in or to determine whether the highest sequence number value has been passed or not.
  THe second is used to indicate whether status pigyyback is possible or not.*/

retransmissions_requested              IndicatorType,
 /*An indicator used to keep track whether a generated status report contains retransmission requests or not.*/

 status_timer_active, start_am         IndicatorType,
 /*This indicator keeps track of whether the timer_STATUS timer is running or not.*/

 per                                   REAL,
 /*Local storage of a percentage value.*/

 rx_ongoing, tx_ongoing                IndicatorType,
 /*These indicators are used to maintain information about whether something is in the process of being
   transmitted or received.*/

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

vr_ep                                  SequenceNumberType;
/*Estimated PDU counter state variable: The number of PUs that should have been received after the latest
  STATU PDU was sent. In acknowledged mode, this state variable is updated at the end of each transmission
  time interval. If vr_ep is equal to the number of requested PUs in the latest STATUS PDU it should be checked
  if all PUs requested for retransmission have been received.*/
```

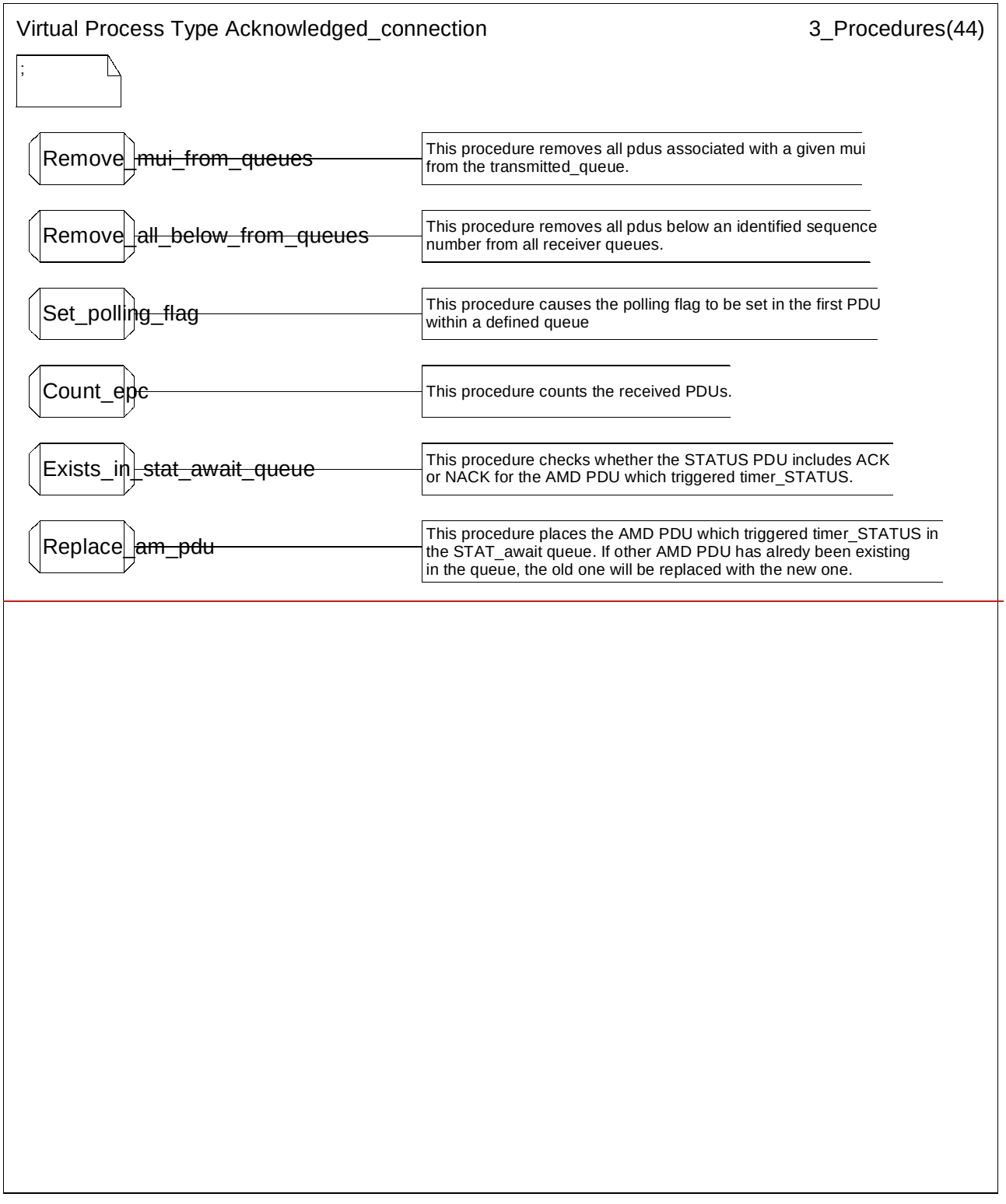
Virtual Process Type Acknowledged_connection 4_Declarations(44)

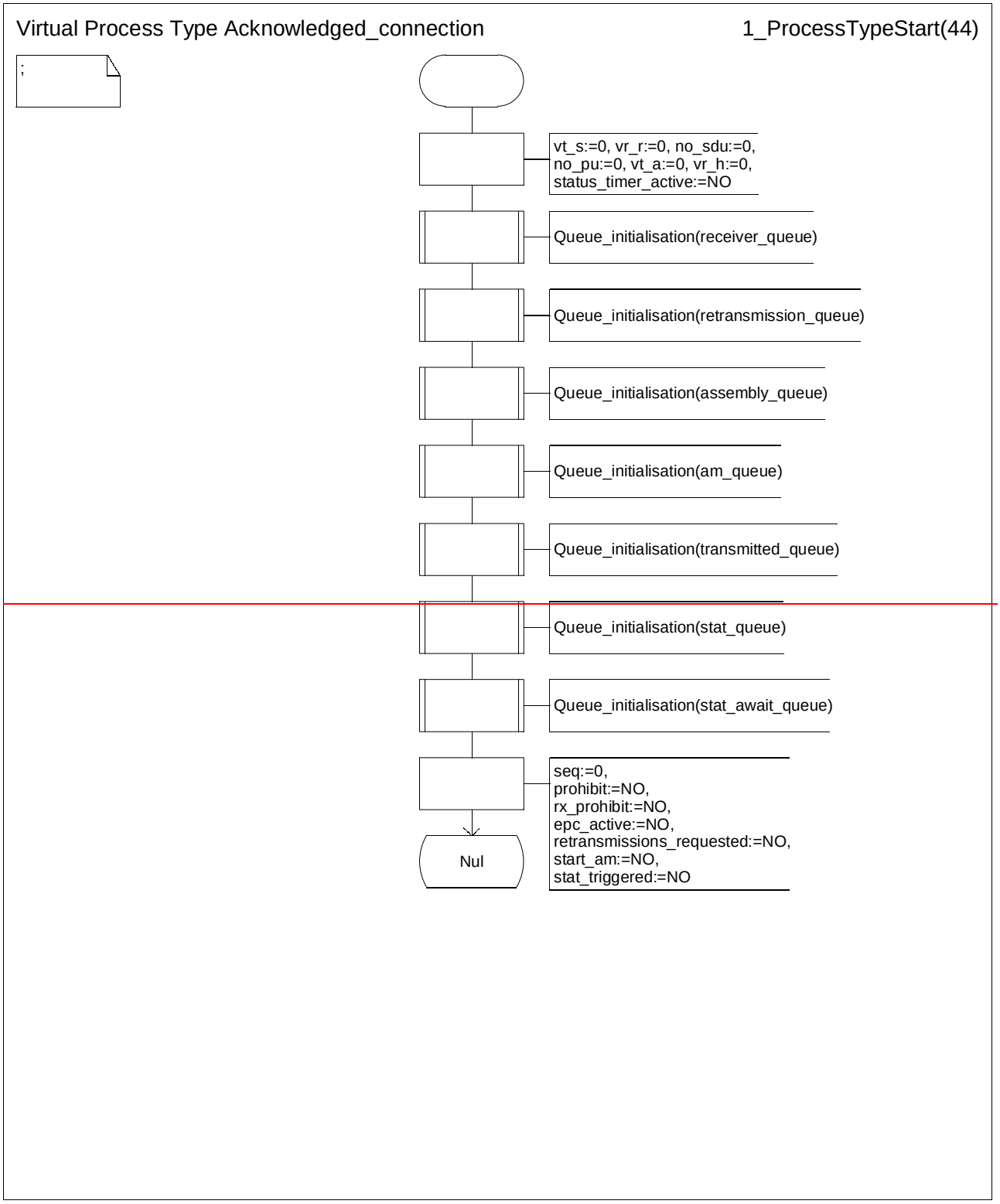
;

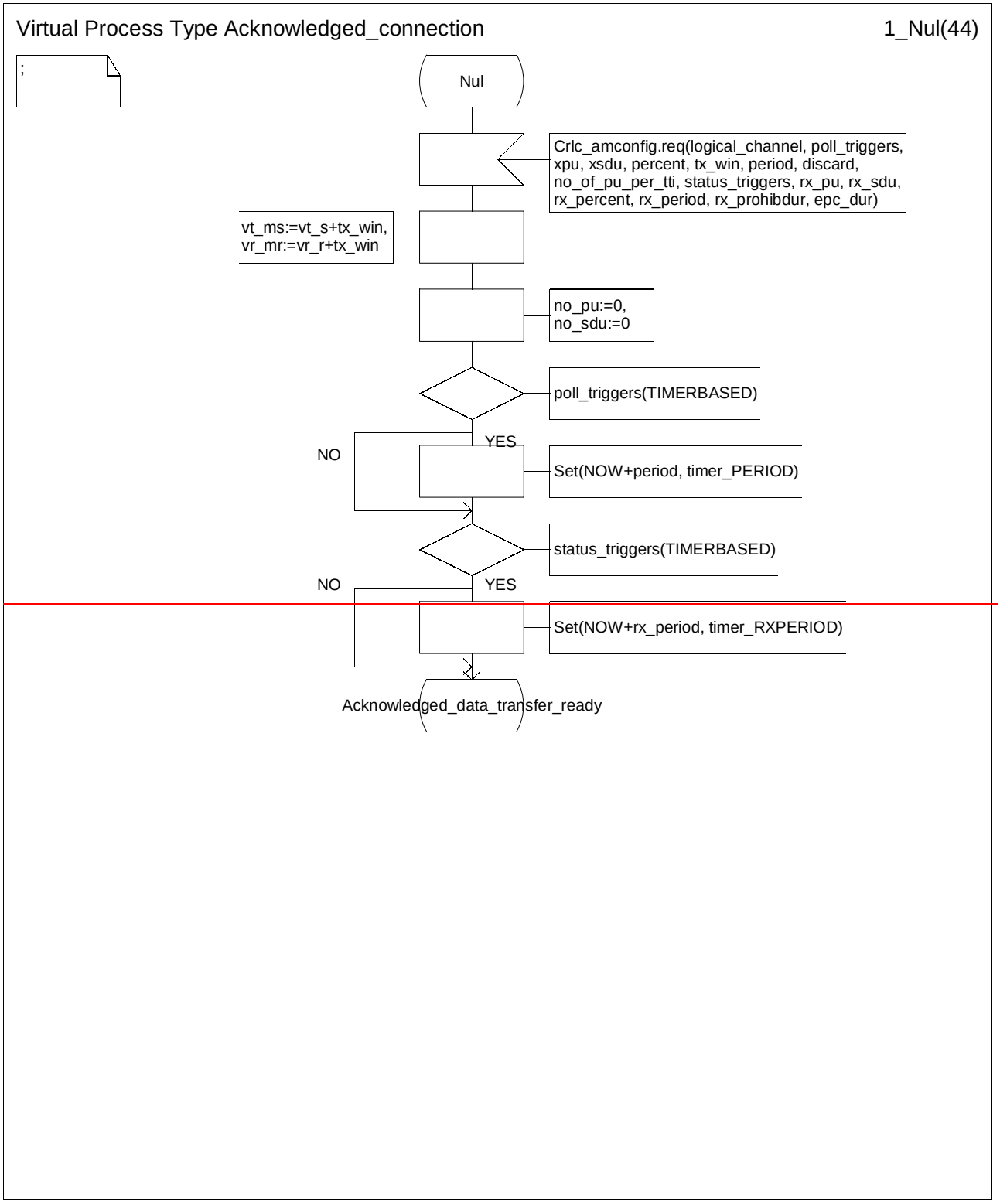
```
TIMER

 timer_AM,
 /*This timer is used to sequence transmissions.*/

 timer_EPC,
 /*This timer accounts for the round trip delay, i.e. the time when the first retransmitted PU should have been
    received after a status report has been sent. The value of timer is heavily based on the transmission time
    interval (layer 1 interleaving depth). When changing the transmission time interval, the value of the EPC
    timer also needs to be changed.*/

 timer_STATUS,
 /*This timer is used to detect the loss of response from the receiver side. The timer is set when a transmitted AmPdu
    requests a status report and it will be stopped when the transmitter receives acknowledgement of the pdu within StatPdu
    (positive) or UstatPdu (negative). When the timer expires, the pdus of the oldest unconfirmed pdus should be
    retransmitted together with a status report request and the timer set again. If polling takes place when this timer
    is active, it should be reset an then se again.*/

  timer_DISCARD(MuiType),
  /*This timer is used for the SDU discard function. In the transmitter, the timer is activated upon reception of an
     SDU from a higher layer. If the SDU has not been acknowledged when the timer expires, the SDU is discarded
     and a move receiving window request is sent to the receiver. If the SDU discard function does not use the move
     receiving window request, the timer is also used in the receiver, where it is activated once a PDU is detected as
     outstanding, i.e. there is a gap betweensequencembers of received PDUs.*/

  timer_PERIOD,
  /*A timer used for the periodic creation of polls.*/

  timer_RXPERIOD,
  /*A timer used for the periodic creation of status reports.*/

  timer_RXPROHIBIT,
  /*A timer used on the receive side to limit STATUS transmissions.*/

 timer_PROHIBIT;
 /*It is used to prohibit transmission of polling messages within a certain period. If polling takes place while the timer is
   active, it will be reset and then set again. No action other than indicating that the timer is not active is needed
   when it expires.*/
```

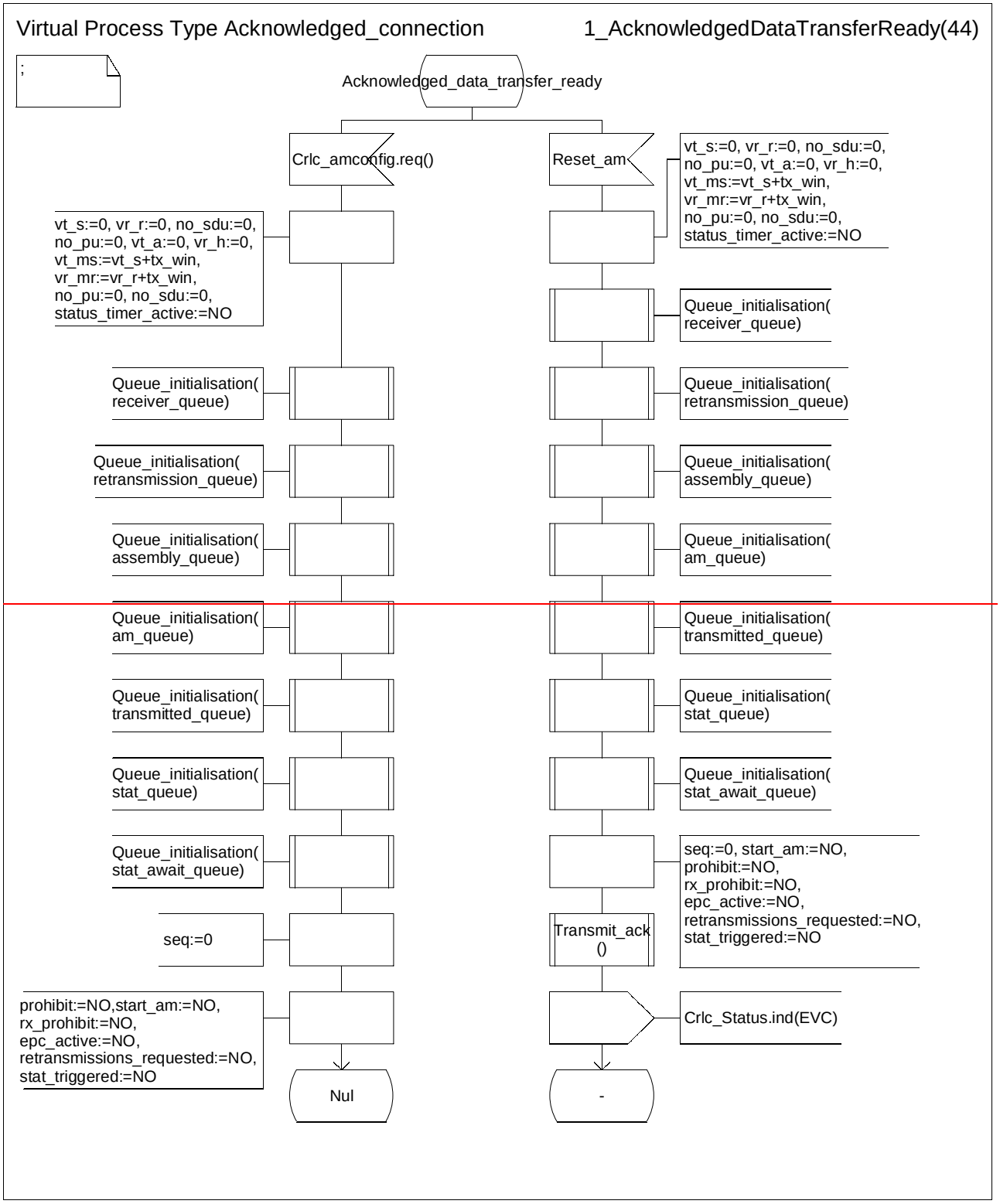

Virtual Process Type Acknowledged_connection
1_Procedures(44)
;
Sdu_am_segmentation
This procedure manages segmentation and concatenation of sdus. It applies pollingin accorance with the toolset functions applied by the higher layer protocols.
Virtual Transmit_am_pdu
This procedure manages transmission of RLC PDUs across the proper SAP.
Check_if_queue_empty
This procedure checks if there are any PDUs remaining in the queue given as parameter to the procedure.
Remove_from_queue
This procedure removes the first PDU in the queue given as parameter to the procedure.
Place_in_queue
This procedure places the indicated pdu within the queue given as parameter to the procedure
Update_sequence_number
This procedure increments the sequence number properly based on the maximum allowed.
Read_pdu
This procedure retrieves a copy of the first entry in the queue indicated as parameter to the procedure.
Remove_identified_from_queue
This procedure removes a pdu with a given sequence number from the queue identified.
Remove_acks_get_muis
This procedure removes all pdus that have been acknowledged from the indicated queue and stores the muis that are removed from the queue in a special array.
Complete_muis
This procedure checks if any of the muis identified still exists 'within the retransmission queue and updates the list of muis that should be confirmed accordingly.
Remove_list_from_transmitted_queue
This procedure removes a list of pdus indicated by sequence numbers from the transmitted queue.
Remove_bitmap_from_transmitted_queue
This procedure removes a list of pdus in accordance with a bitmap from the transmitted queue.

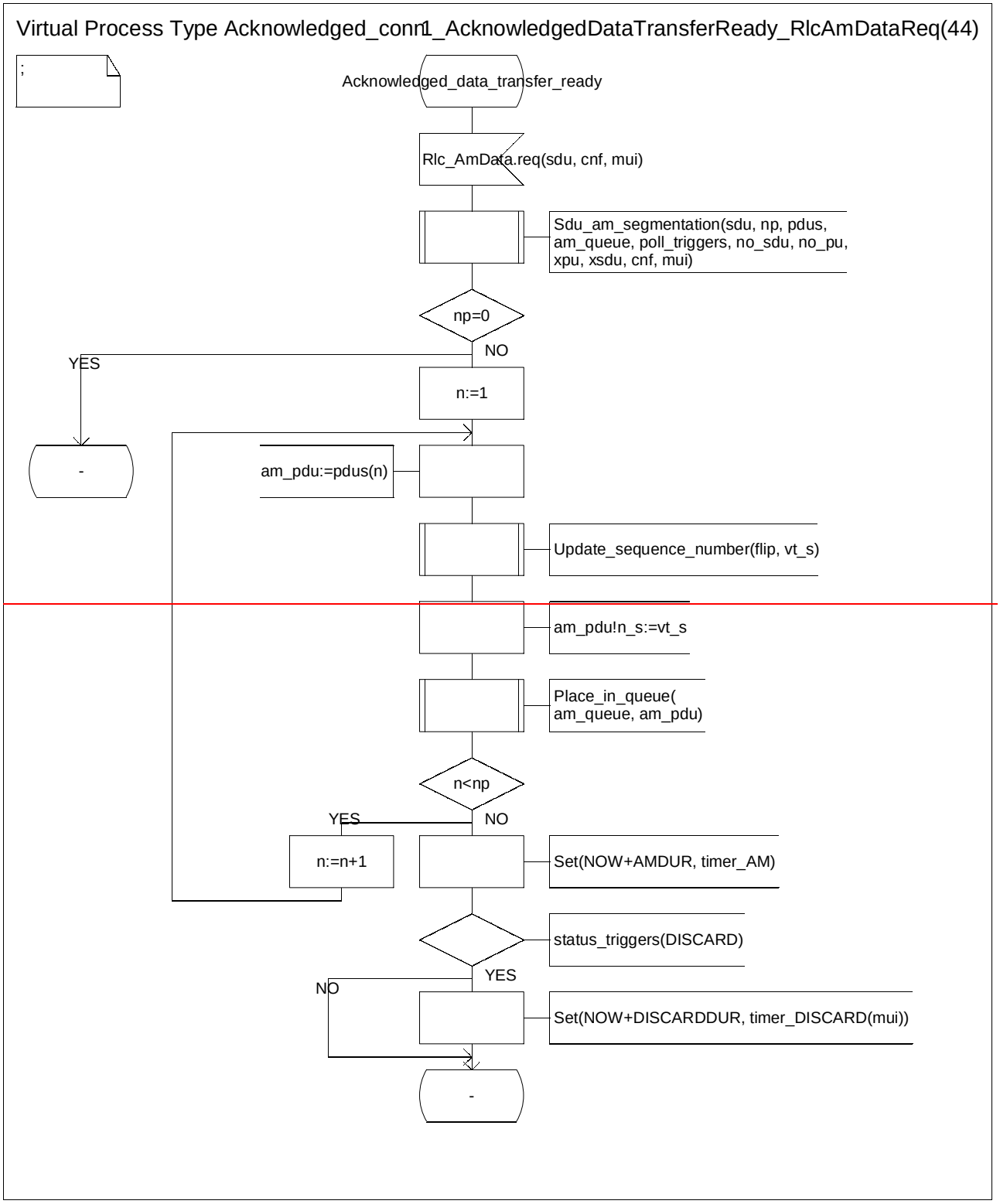

Virtual Process Type Acknowledged_connection
2_Procedures(44)
;
Place_several_in_queue
This procedure places several pdus in the indicated queue.
Update_state_variables
This procedure updates the state variables vt_a and vt_ms after a STATUS PDU has been received and processed.
Place_first_in_queue
This procedure places an AM_PDU with polling=YES first in the retransmission queue after its associated STATUS timer 'has expired.
Reassemble_am_pdu
This procedure reassembles Rlc pdu contents into Sdu:s as they arrive.
Virtual Transmit_ack
This procedure transmits a reset acknowledgement on the correct logical channel.
Virtual Transmit_reset
This procedure transmits a reset on the correct logical channel.
Virtual Transmit_stat
This procedure transmits status signal on the correct logical channel.
Place_piggyback_in_queue
This procedure places a sufi containing a move receive window piggybacked onto a pdu within a queue.
Exists_in_receiver_queue
This procedure checks if an identified pdu exists within the receiver queue.
Create_status
This procedure creates a status report based on available information.
Check_status_creation
This procedure checks if a status report should be generated.
Place_in_stat_queue
This procedure places a STAT_PDU in a queue waiting for transmission.
Remove_from_stat_queue
This procedure removes a STAT-PDU from the STAT queue.

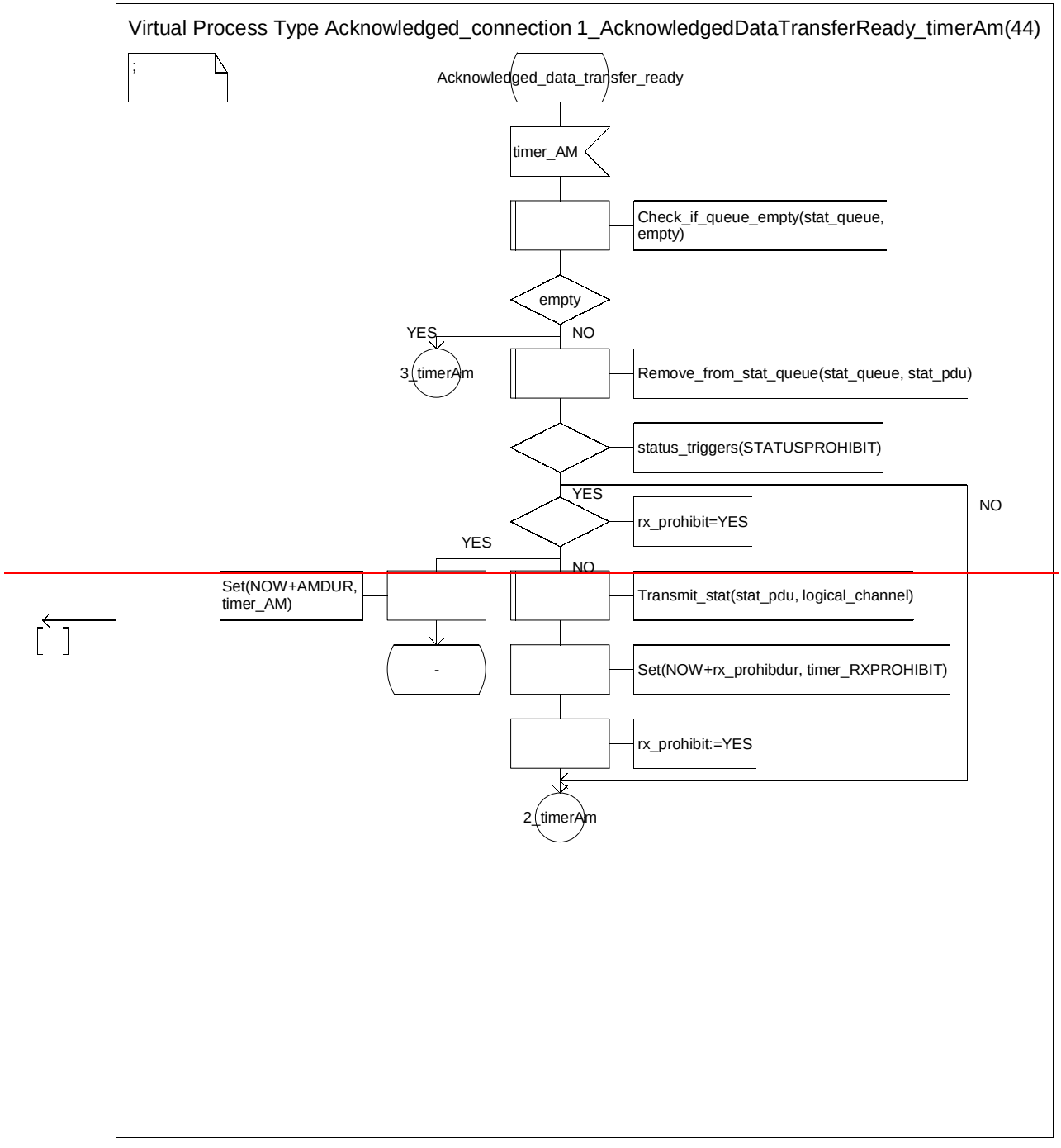

Virtual Process Type Acknowledged_connection
3_Procedures(44)
;
Remove_mui_from_queues
This procedure removes all pdus associated with a given mui from the transmitted_queue.
Remove_all_below_from_queues
This procedure removes all pdus below an identified sequence number from all receiver queues.
Set_polling_flag
This procedure causes the polling flag to be set in the first PDU within a defined queue
Count_epc
This procedure counts the received PDUs.
Exists_in_stat_await_queue
This procedure checks whether the STATUS PDU includes ACK or NACK for the AMD PDU which triggered timer_STATUS.
Replace_am_pdu
This procedure places the AMD PDU which triggered timer_STATUS in the STAT_await queue. If other AMD PDU has alredy been existing in the queue, the old one will be replaced with the new one.


Virtual Process Type Acknowledged_connection
1_ProcessTypeStart(44)
;
vt_s:=0, vr_r:=0, no_sdu:=0,
no_pu:=0, vt_a:=0, vr_h:=0,
status_timer_active:=NO
Queue_initialisation(receiver_queue)
Queue_initialisation(retransmission_queue)
Queue_initialisation(assembly_queue)
Queue_initialisation(am_queue)
Queue_initialisation(transmitted_queue)
Queue_initialisation(stat_queue)
Queue_initialisation(stat_await_queue)
seq:=0,
prohibit:=NO,
rx_prohibit:=NO,
epc_active:=NO,
retransmissions_requested:=NO,
start_am:=NO,
stat_triggered:=NO
Nul

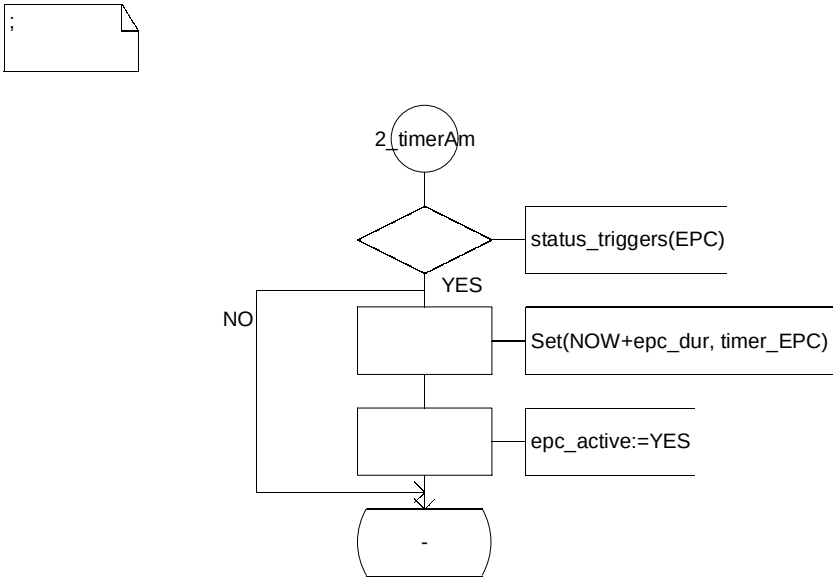

Virtual Process Type Acknowledged_connection
1_Nul(44)
;
Nul
Crlc_amconfig.req(logical_channel, poll_triggers, xpu, xsdu, percent, tx_win, period, discard, no_of_pu_per_tti, status_triggers, rx_pu, rx_sdu, rx_percent, rx_period, rx_prohibdur, epc_dur)
vt_ms:=vt_s+tx_win, vr_mr:=vr_r+tx_win
no_pu:=0, no_sdu:=0
poll_triggers(TIMERBASED)
YES
NO
Set(NOW+period, timer_PERIOD)
status_triggers(TIMERBASED)
NO
YES
Set(NOW+rx_period, timer_RXPERIOD)
Acknowledged_data_transfer_ready

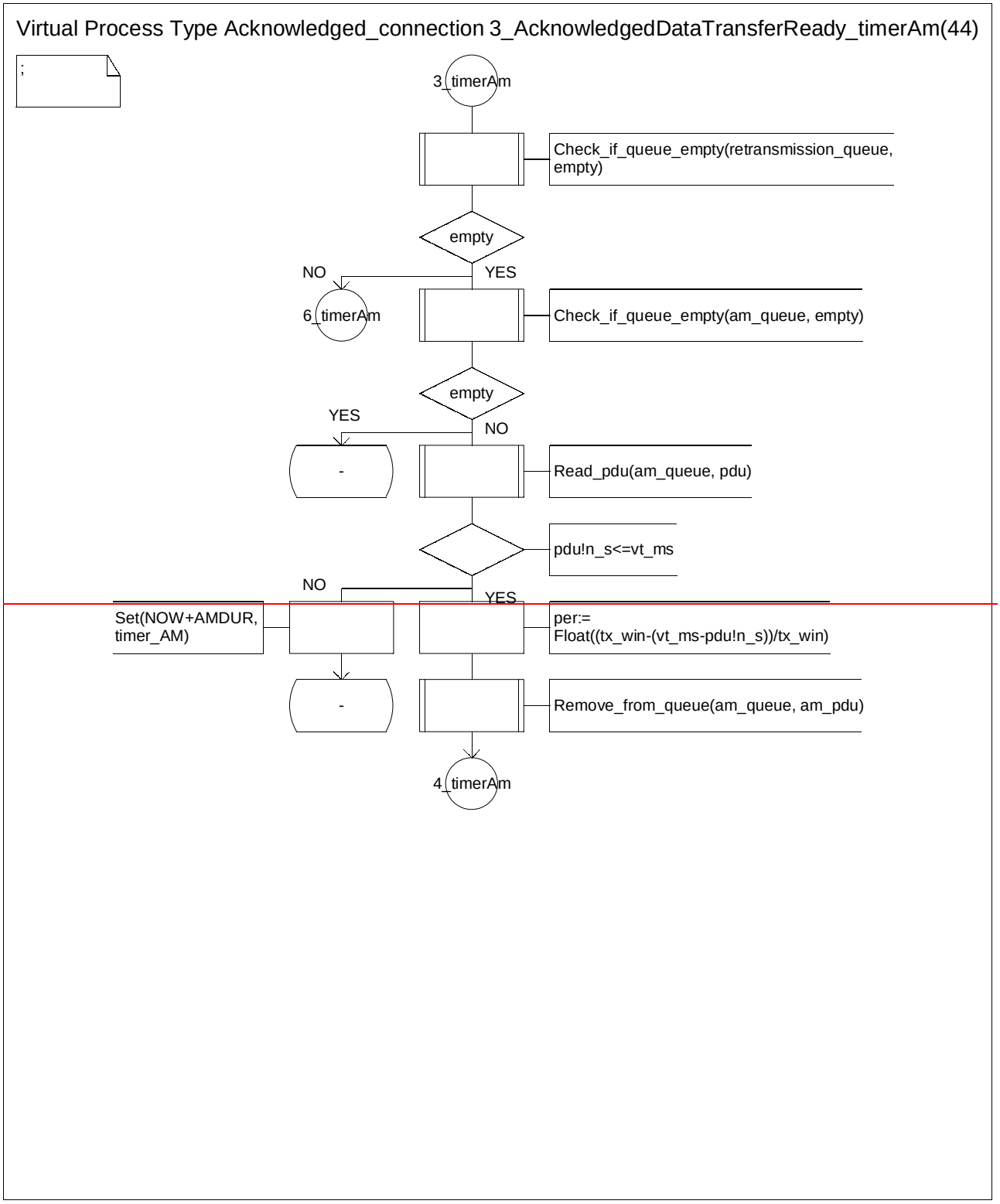

Virtual Process Type Acknowledged_connection
1_AcknowledgedDataTransferReady(44)
;
Acknowledged_data_transfer_ready
Crlc_amconfig.req()
Reset_am
vt_s:=0, vr_r:=0, no_sdu:=0, no_pu:=0, vt_a:=0, vr_h:=0, vt_ms:=vt_s+tx_win, vr_mr:=vr_r+tx_win, no_pu:=0, no_sdu:=0, status_timer_active:=NO
vt_s:=0, vr_r:=0, no_sdu:=0, no_pu:=0, vt_a:=0, vr_h:=0, vt_ms:=vt_s+tx_win, vr_mr:=vr_r+tx_win, no_pu:=0, no_sdu:=0, status_timer_active:=NO
Queue_initialisation(receiver_queue)
Queue_initialisation(receiver_queue)
Queue_initialisation(retransmission_queue)
Queue_initialisation(retransmission_queue)
Queue_initialisation(assembly_queue)
Queue_initialisation(assembly_queue)
Queue_initialisation(am_queue)
Queue_initialisation(am_queue)
Queue_initialisation(transmitted_queue)
Queue_initialisation(transmitted_queue)
Queue_initialisation(stat_queue)
Queue_initialisation(stat_queue)
Queue_initialisation(stat_await_queue)
Queue_initialisation(stat_await_queue)
seq:=0
seq:=0, start_am:=NO, prohibit:=NO, rx_prohibit:=NO, epc_active:=NO, retransmissions_requested:=NO, stat_triggered:=NO
Transmit_ack ()
prohibit:=NO,start_am:=NO, rx_prohibit:=NO, epc_active:=NO, retransmissions_requested:=NO, stat_triggered:=NO
Crlc_Status.ind(EVC)
Nul
-

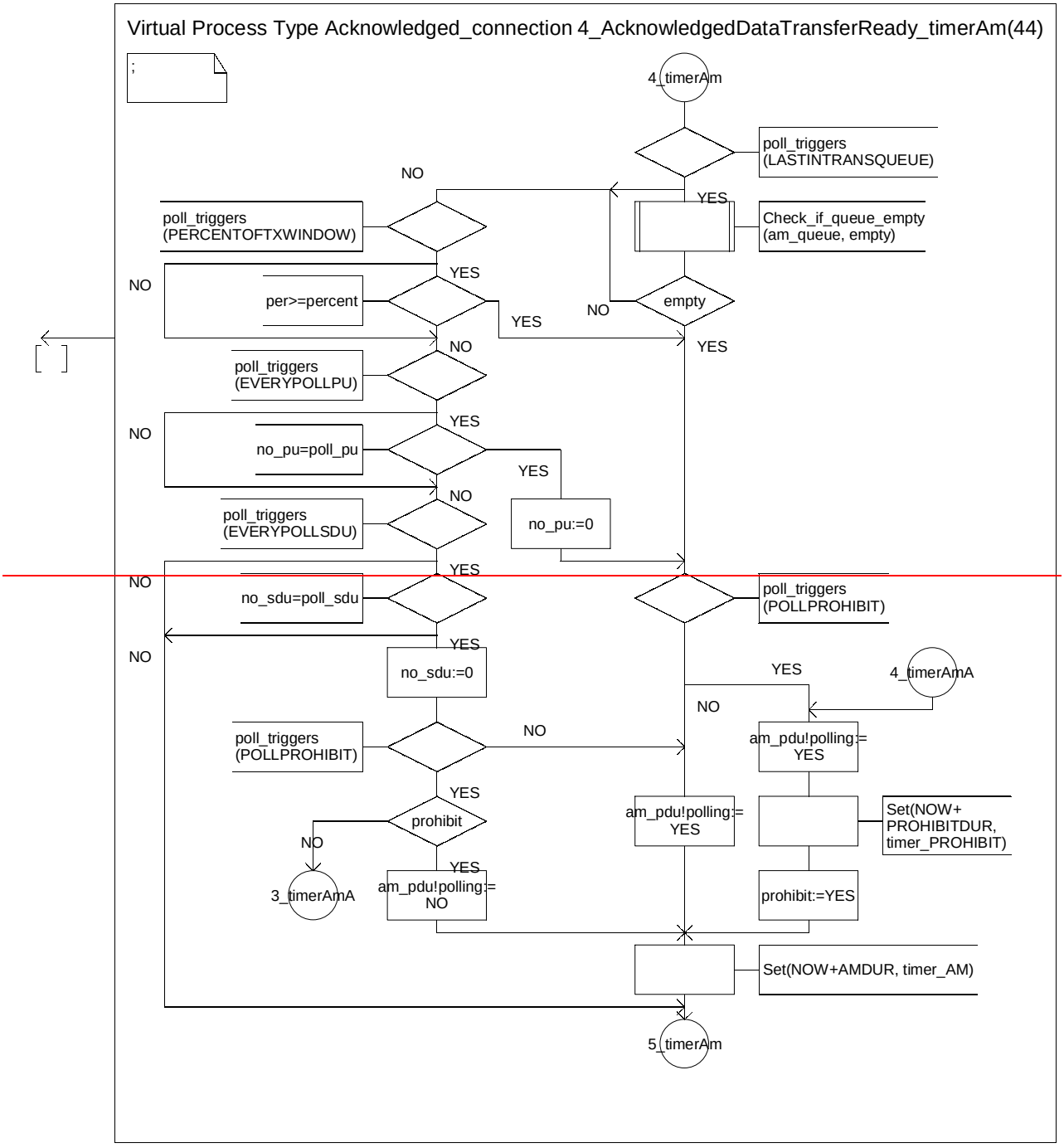

Virtual Process Type Acknowledged_conn1_AcknowledgedDataTransferReady_RlcAmDataReq(44)
;
Acknowledged_data_transfer_ready
Rlc_AmData.req(sdu, cnf, mui)
Sdu_am_segmentation(sdu, np, pdus, am_queue, poll_triggers, no_sdu, no_pu, xpu, xsdu, cnf, mui)
np=0
NO
YES
n:=1
-
am_pdu:=pdus(n)
Update_sequence_number(flip, vt_s)
am_pdu!n_s:=vt_s
Place_in_queue(am_queue, am_pdu)
n<np
YES
NO
n:=n+1
Set(NOW+AMDUR, timer_AM)
status_triggers(DISCARD)
YES
NO
Set(NOW+DISCARDDUR, timer_DISCARD(mui))
-

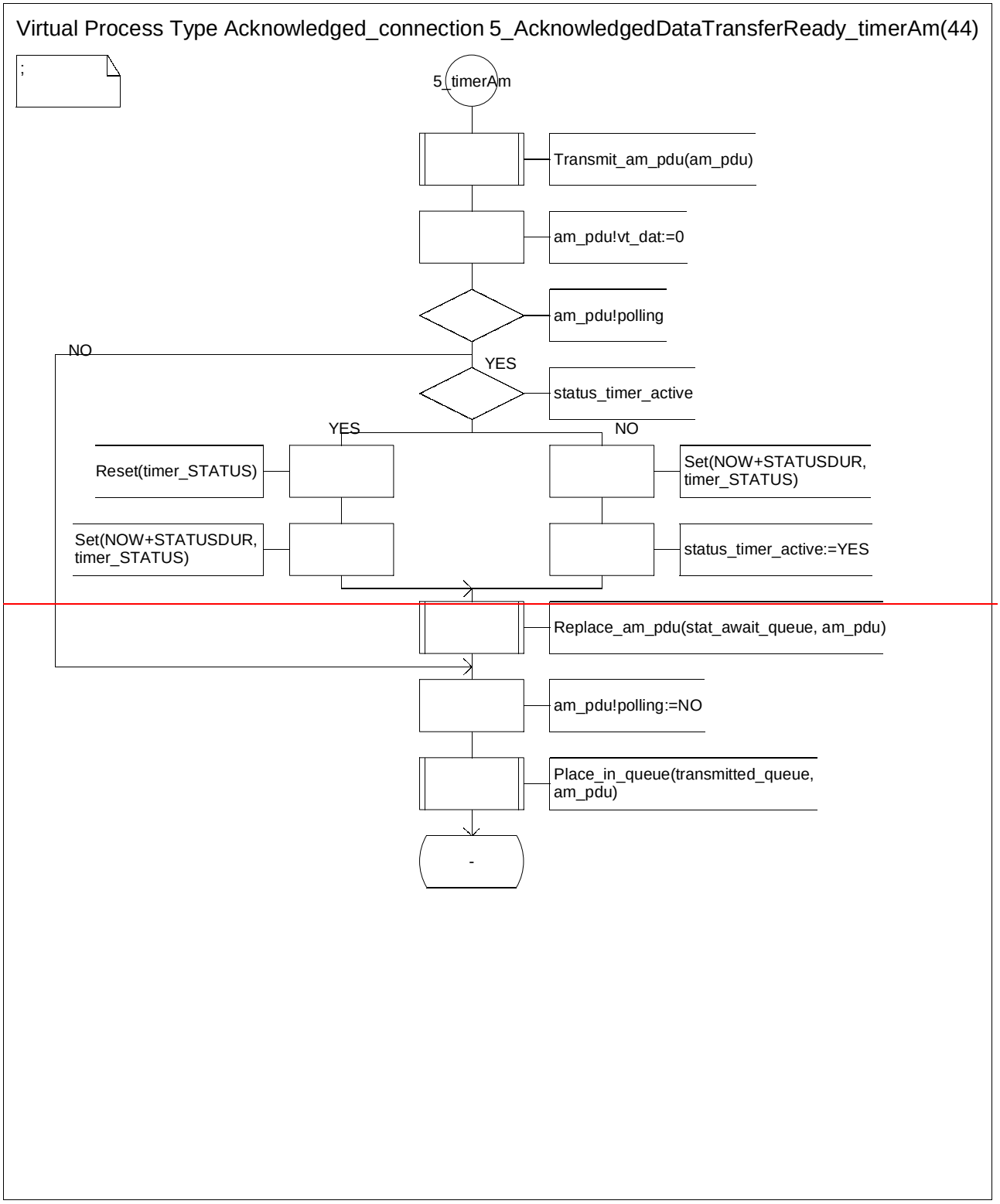

Virtual Process Type Acknowledged_connection 1_AcknowledgedDataTransferReady_timerAm(44)
;
Acknowledged_data_transfer_ready
timer_AM
Check_if_queue_empty(stat_queue, empty)
empty
YES
NO
3_timerAm
Remove_from_stat_queue(stat_queue, stat_pdu)
status_triggers(STATUSPROHIBIT)
YES
NO
rx_prohibit=YES
YES
NO
Set(NOW+AMDUR, timer_AM)
Transmit_stat(stat_pdu, logical_channel)
-
Set(NOW+rx_prohibdur, timer_RXPROHIBIT)
rx_prohibit:=YES
2_timerAm

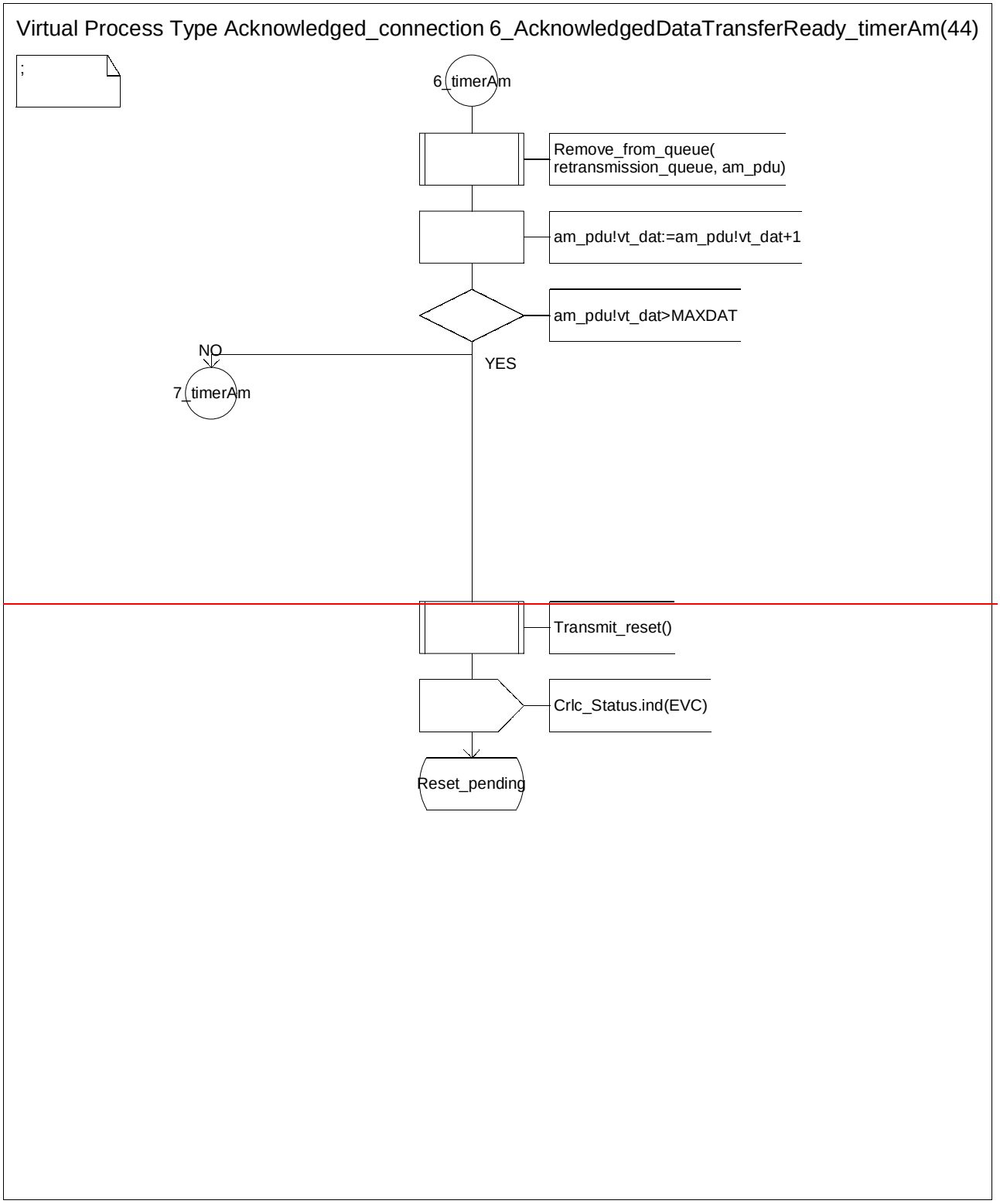
Virtual Process Type Acknowledged_connection 2_AcknowledgedDataTransferReady_timerAm(44)

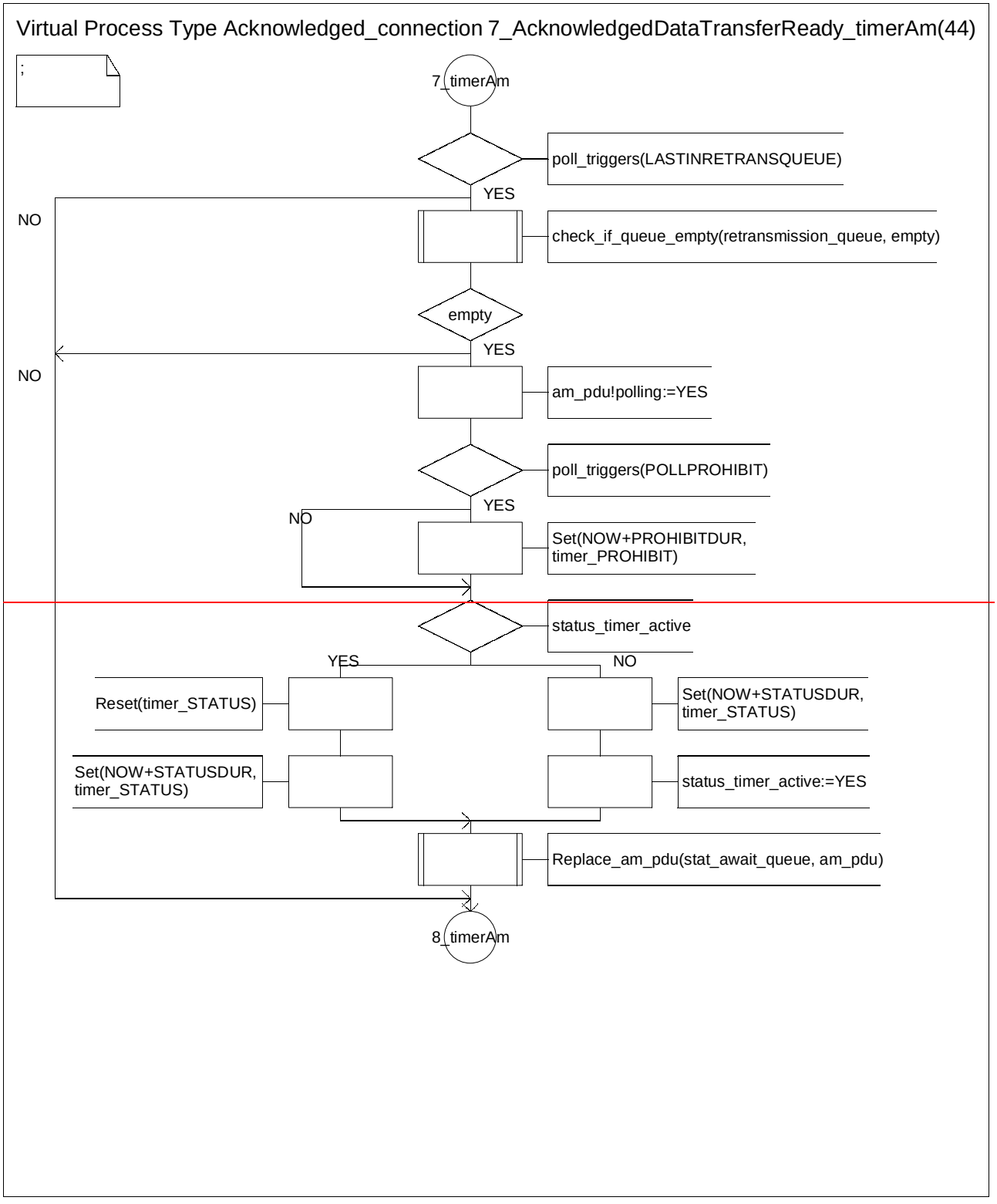


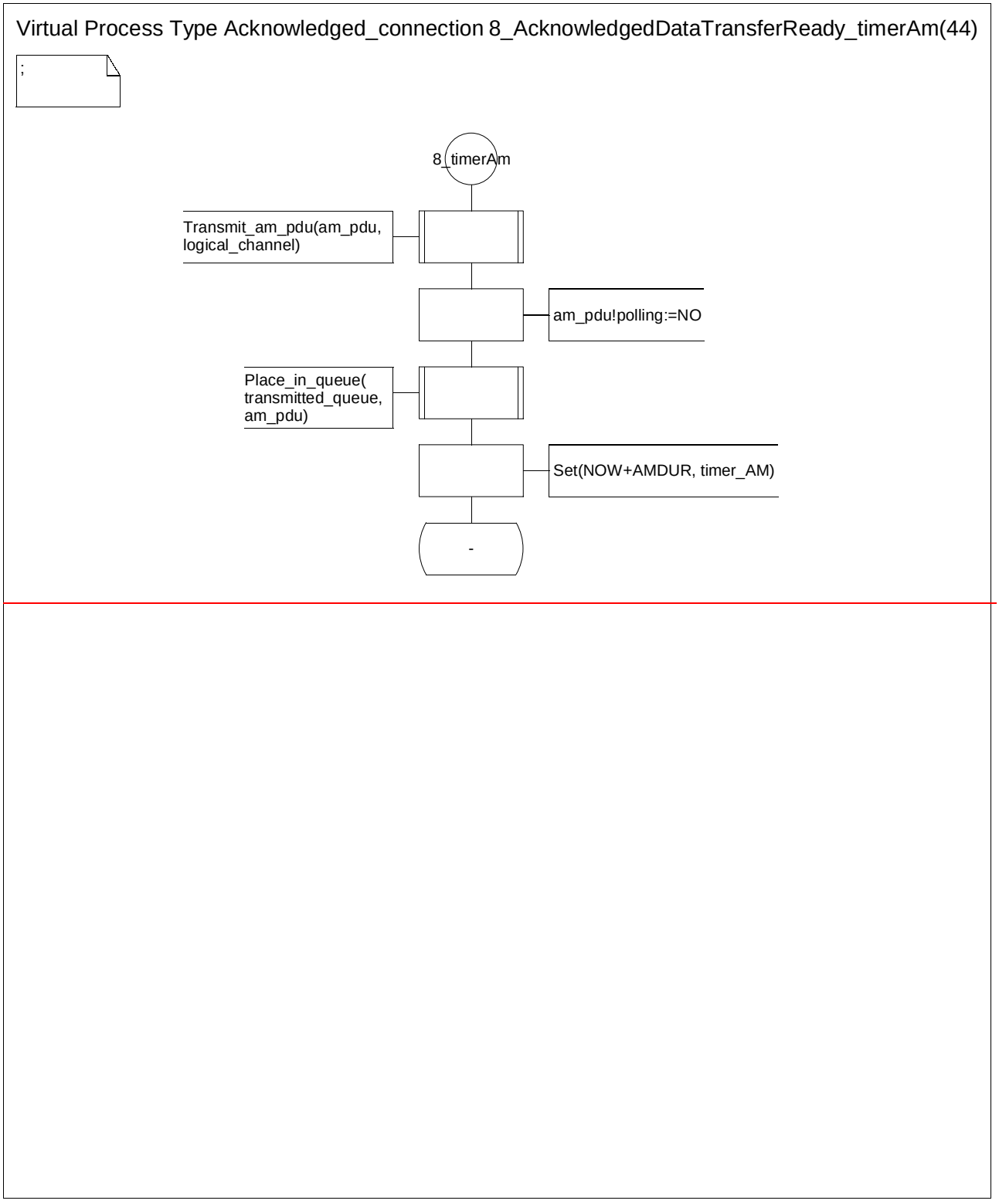

Virtual Process Type Acknowledged_connection 3_AcknowledgedDataTransferReady_timerAm(44)
;
3_timerAm
Check_if_queue_empty(retransmission_queue, empty)
empty
NO
YES
6_timerAm
Check_if_queue_empty(am_queue, empty)
empty
YES
NO
-
Read_pdu(am_queue, pdu)
pdu!n_s<=vt_ms
NO
YES
Set(NOW+AMDUR, timer_AM)
per:= Float((tx_win-(vt_ms-pdu!n_s))/tx_win)
-
Remove_from_queue(am_queue, am_pdu)
4_timerAm

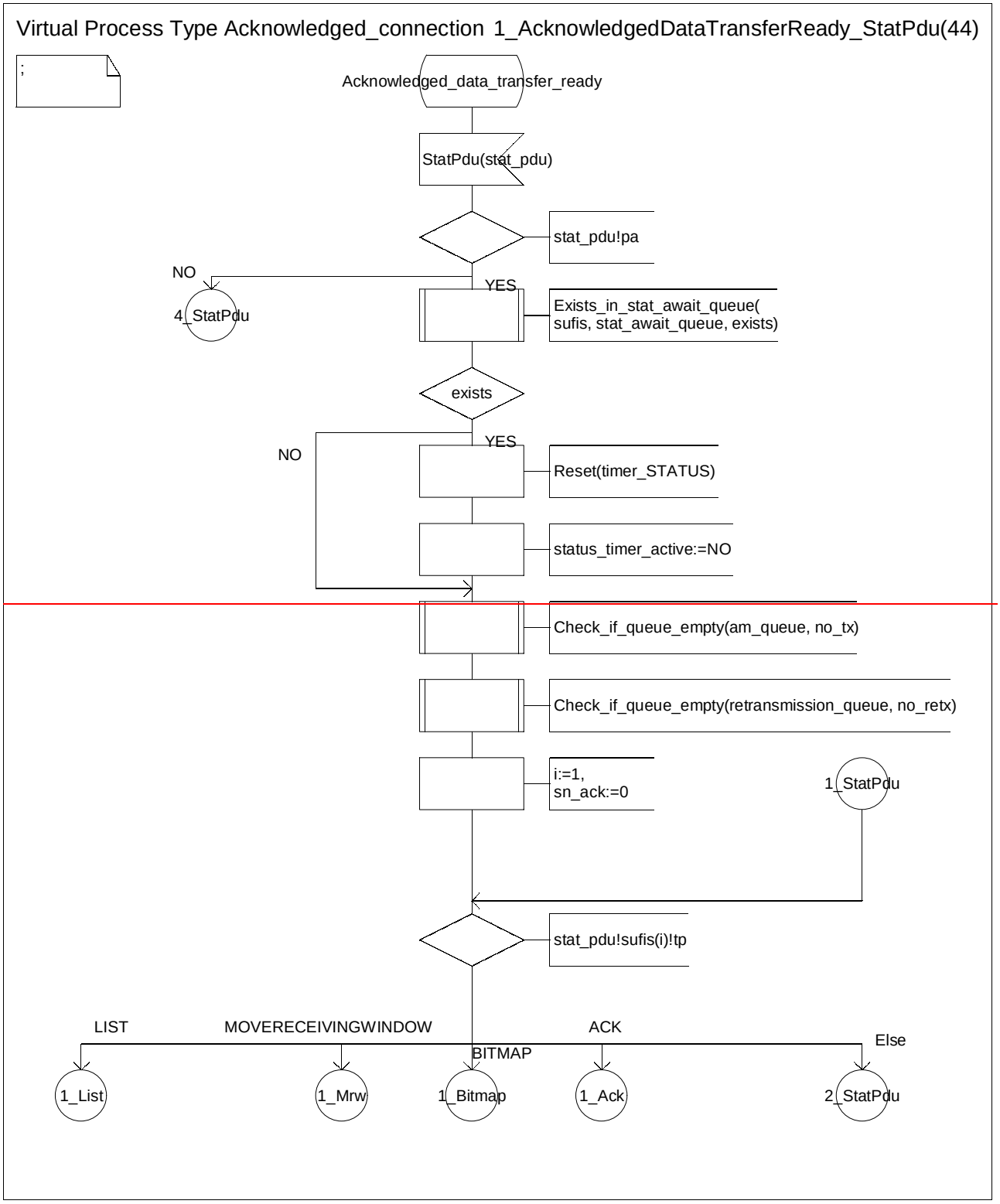

Virtual Process Type Acknowledged_connection 4_AcknowledgedDataTransferReady_timerAm(44)
;
4_timerAm
poll_triggers (LASTINTRANSQUEUE)
NO
YES
poll_triggers (PERCENTOFTXWINDOW)
Check_if_queue_empty (am_queue, empty)
YES
NO
per>=percent
empty
NO
YES
YES
NO
YES
poll_triggers (EVERYPOLLPU)
NO
YES
no_pu=poll_pu
YES
NO
poll_triggers (EVERYPOLLSDU)
no_pu:=0
YES
NO
no_sdu=poll_sdu
poll_triggers (POLLPROHIBIT)
YES
NO
no_sdu:=0
YES
4_timerAmA
NO
poll_triggers (POLLPROHIBIT)
NO
am_pdu!polling:= YES
YES
prohibit
am_pdu!polling:= YES
Set(NOW+ PROHIBITDUR, timer_PROHIBIT)
NO
YES
3_timerAmA
am_pdu!polling:= NO
prohibit:=YES
Set(NOW+AMDUR, timer_AM)
5_timerAm

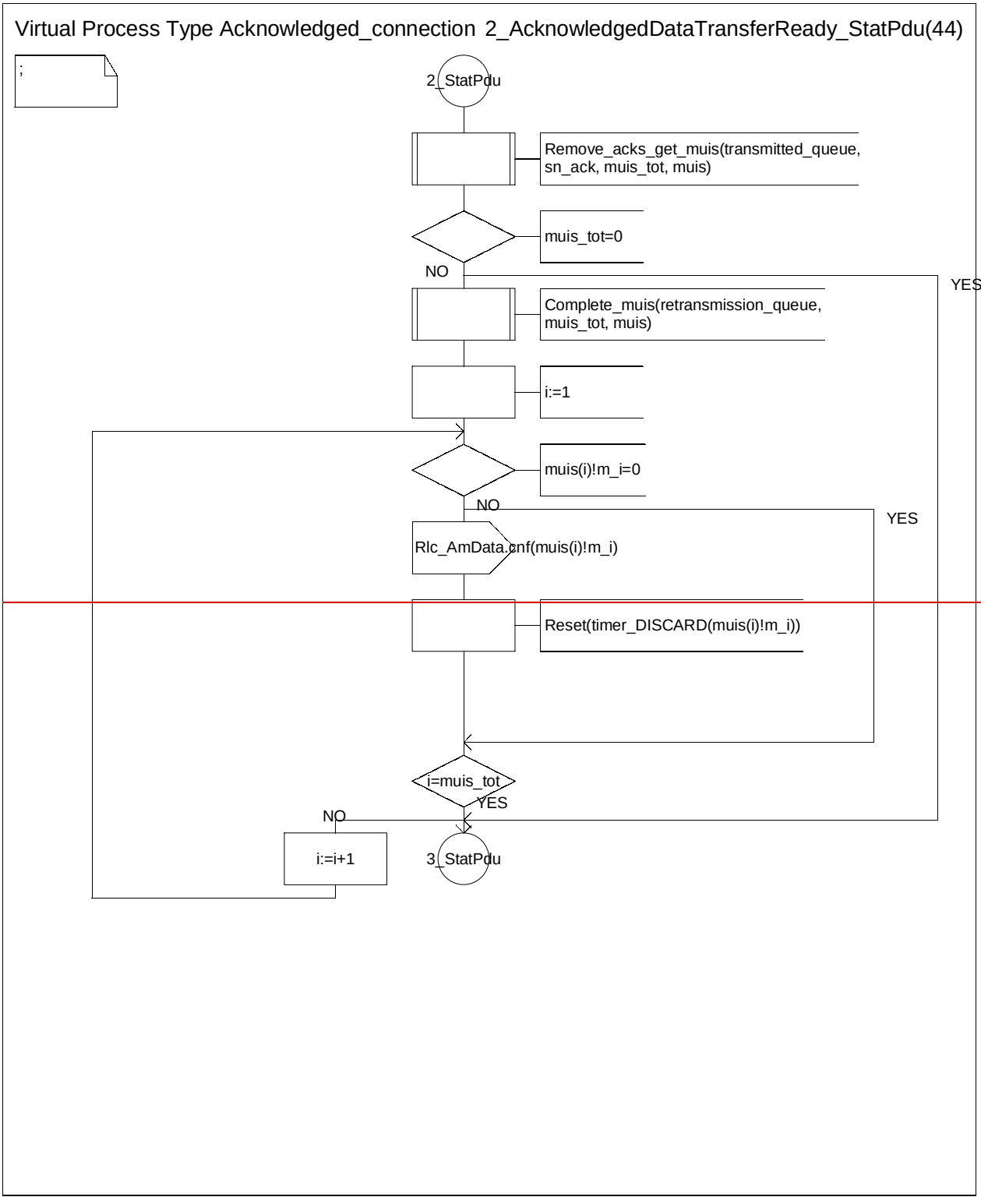

Virtual Process Type Acknowledged_connection 5_AcknowledgedDataTransferReady_timerAm(44)
;
5_timerAm
Transmit_am_pdu(am_pdu)
am_pdu!vt_dat:=0
am_pdu!polling
NO
YES
status_timer_active
YES
NO
Reset(timer_STATUS)
Set(NOW+STATUSDUR, timer_STATUS)
Set(NOW+STATUSDUR, timer_STATUS)
status_timer_active:=YES
Replace_am_pdu(stat_await_queue, am_pdu)
am_pdu!polling:=NO
Place_in_queue(transmitted_queue, am_pdu)
-

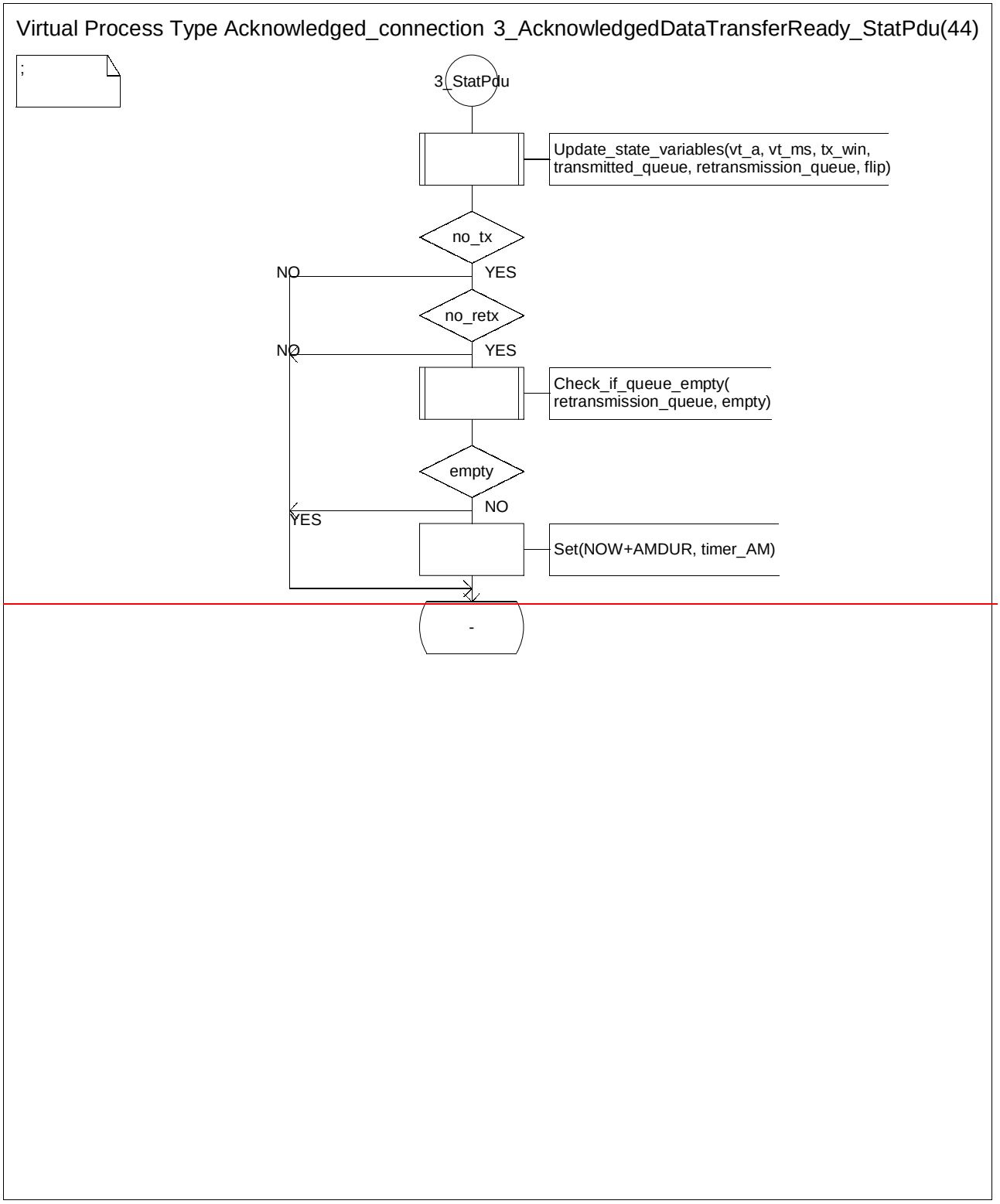

Virtual Process Type Acknowledged_connection 6_AcknowledgedDataTransferReady_timerAm(44)
;
6_timerAm
Remove_from_queue(retransmission_queue, am_pdu)
am_pdu!vt_dat:=am_pdu!vt_dat+1
am_pdu!vt_dat>MAXDAT
NO
YES
7_timerAm
Transmit_reset()
Crlc_Status.ind(EVC)
Reset_pending

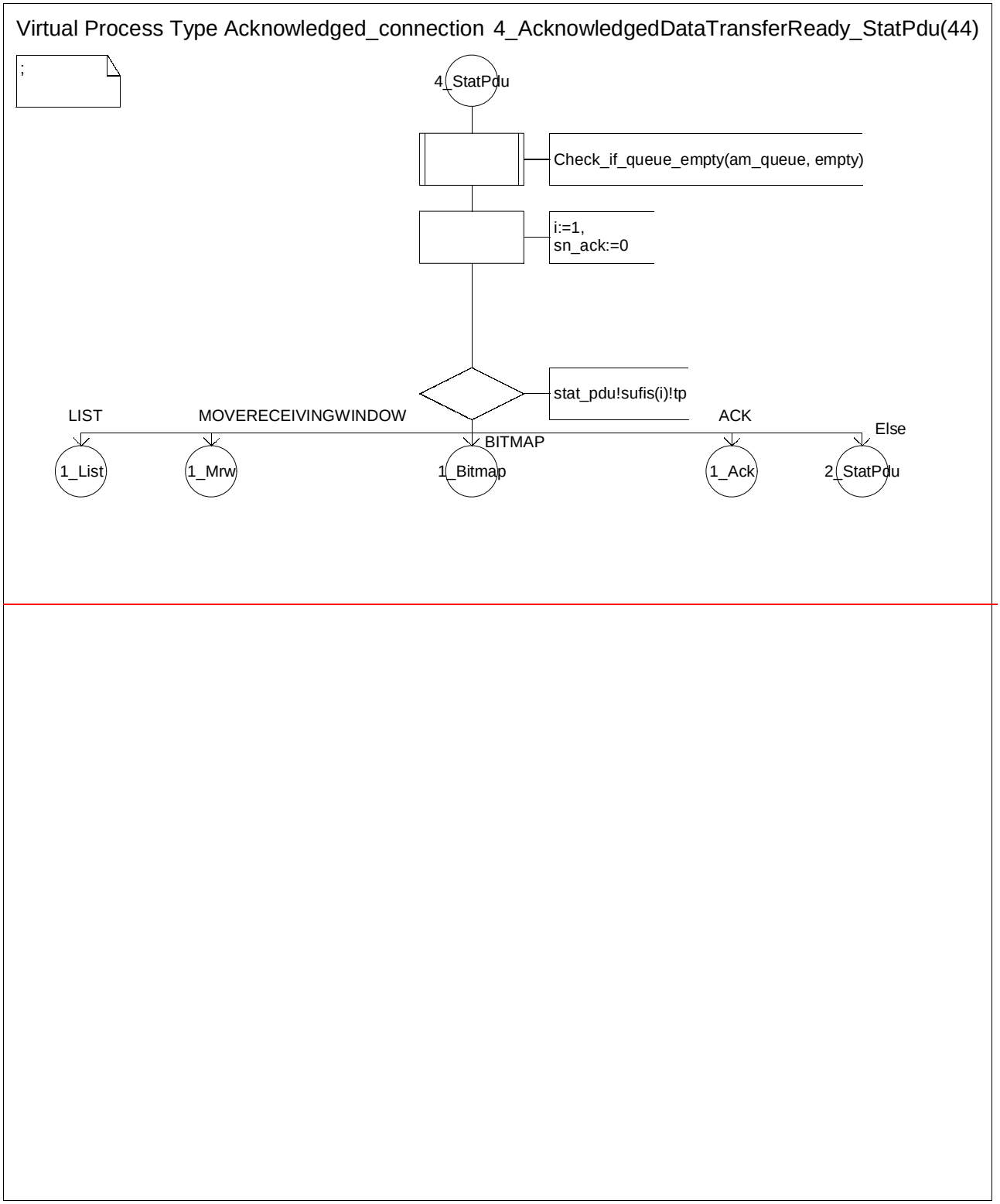

Virtual Process Type Acknowledged_connection 7_AcknowledgedDataTransferReady_timerAm(44)
;
7_timerAm
poll_triggers(LASTINRETRANSQUEUE)
YES
NO
check_if_queue_empty(retransmission_queue, empty)
empty
YES
NO
am_pdu!polling:=YES
poll_triggers(POLLPROHIBIT)
YES
NO
Set(NOW+PROHIBITDUR, timer_PROHIBIT)
status_timer_active
YES
NO
Reset(timer_STATUS)
Set(NOW+STATUSDUR, timer_STATUS)
Set(NOW+STATUSDUR, timer_STATUS)
status_timer_active:=YES
Replace_am_pdu(stat_await_queue, am_pdu)
8_timerAm

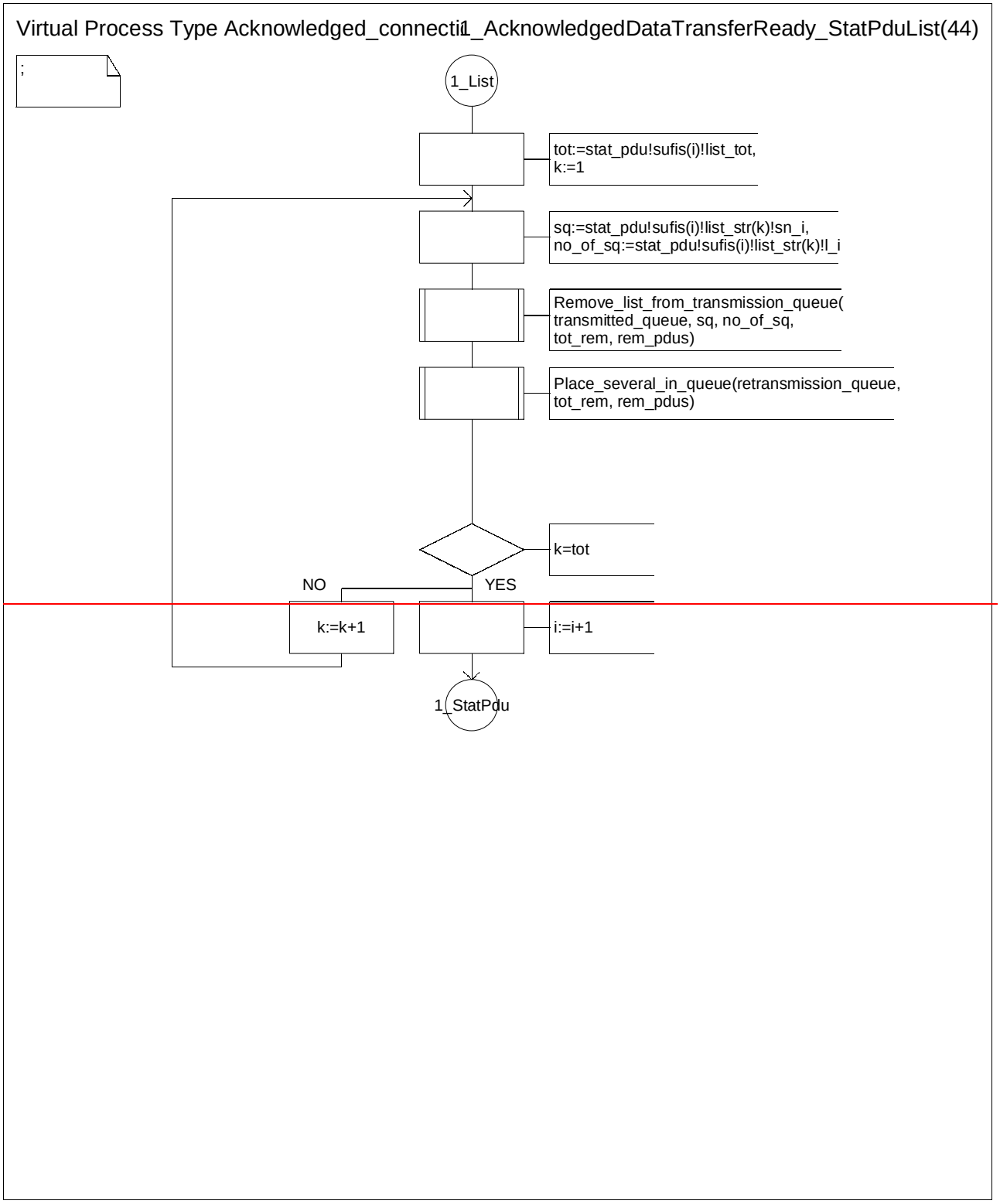

Virtual Process Type Acknowledged_connection 8_AcknowledgedDataTransferReady_timerAm(44)
;
8_timerAm
Transmit_am_pdu(am_pdu, logical_channel)
am_pdu!polling:=NO
Place_in_queue(transmitted_queue, am_pdu)
Set(NOW+AMDUR, timer_AM)
-

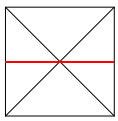

Virtual Process Type Acknowledged_connection 1_AcknowledgedDataTransferReady_StatPdu(44)
;
Acknowledged_data_transfer_ready
StatPdu(stat_pdu)
stat_pdu!pa
NO
YES
4_StatPdu
Exists_in_stat_await_queue(
sufis, stat_await_queue, exists)
exists
NO
YES
Reset(timer_STATUS)
status_timer_active:=NO
Check_if_queue_empty(am_queue, no_tx)
Check_if_queue_empty(retransmission_queue, no_retx)
i:=1,
sn_ack:=0
1_StatPdu
stat_pdu!sufis(i)!tp
LIST
MOVERECEIVINGWINDOW
BITMAP
ACK
Else
1_List
1_Mrw
1_Bitmap
1_Ack
2_StatPdu


Virtual Process Type Acknowledged_connection 2_AcknowledgedDataTransferReady_StatPdu(44)
;
2_StatPdu
Remove_acks_get_muis(transmitted_queue, sn_ack, muis_tot, muis)
muis_tot=0
NO
YES
Complete_muis(retransmission_queue, muis_tot, muis)
i:=1
muis(i)!m_i=0
NO
YES
Rlc_AmData.cnf(muis(i)!m_i)
Reset(timer_DISCARD(muis(i)!m_i))
i=muis_tot
YES
NO
i:=i+1
3_StatPdu


Virtual Process Type Acknowledged_connection 3_AcknowledgedDataTransferReady_StatPdu(44)
;
3_StatPdu
Update_state_variables(vt_a, vt_ms, tx_win, transmitted_queue, retransmission_queue, flip)
no_tx
NO
YES
no_retx
NO
YES
Check_if_queue_empty(retransmission_queue, empty)
empty
NO
YES
Set(NOW+AMDUR, timer_AM)
-


Virtual Process Type Acknowledged_connection 4_AcknowledgedDataTransferReady_StatPdu(44)
;
4_StatPdu
Check_if_queue_empty(am_queue, empty)
i:=1,
sn_ack:=0
stat_pdu!sufis(i)!tp
LIST
MOVERECEIVINGWINDOW
BITMAP
ACK
Else
1_List
1_Mrw
1_Bitmap
1_Ack
2_StatPdu


Virtual Process Type Acknowledged_connecti1_AcknowledgedDataTransferReady_StatPduList(44)
;
1_List
tot:=stat_pdu!sufis(i)!list_tot,
k:=1
sq:=stat_pdu!sufis(i)!list_str(k)!sn_i,
no_of_sq:=stat_pdu!sufis(i)!list_str(k)!l_i
Remove_list_from_transmission_queue(
transmitted_queue, sq, no_of_sq,
tot_rem, rem_pdus)
Place_several_in_queue(retransmission_queue,
tot_rem, rem_pdus)
k=tot
NO
YES
k:=k+1
i:=i+1
1_StatPdu



3GPP TSG-RAN Meeting #6
Nice, France, 13-15 December 1999

***Document* (R2-99k23)**
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.322	CR	014	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN#6	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 03 Dec 1999

Subject: Editorial changes

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction	
A	Corresponds to a correction in an earlier release	
B	Addition of feature	
C	Functional modification of feature	
D	Editorial modification	X

Release:

Phase 2	
Release 96	
Release 97	
Release 98	
Release 99	X
Release 00	

Reason for change: Development of suitable version for submission o RAN#6

Clauses affected: 5.2.2.1.1, 5.2.4

Other specs affected:

Other 3G core specifications	[ ]	→ List of CRs:
Other GSM core specifications	[ ]	→ List of CRs:
MS test specifications	[ ]	→ List of CRs:
BSS test specifications	[ ]	→ List of CRs:
O&M specifications	[ ]	→ List of CRs:

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

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

4 General

~~4.1 Objective~~

4.2 Overview on sublayer architecture

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

10 Handling of unknown, unforeseen and erroneous protocol data

~~A preliminary~~The list of ~~possible~~ 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 or BITMAP parameter 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

11 Elementary procedures

Annex A (informative): SDL diagrams

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

~~NOTE: All the section shall be reviewed when the protocol is defined;~~

NOTE: All the SDL diagrams presented are [FFS]

Virtual Process Type Acknowledged_connection 1_Declarations(44)

;

```
DCL

 am_pdu, tmp, pdu                                AmPdu,
 /*A representation of data contained within a AmPdu.*/

 stat_pdu, rx_stat_pdu                           StatPdu,
 /*A representation of data contained within a StatPdu.*/

 pdus, rem_pdus                                  AmPduArrayType,
 /*The initally segmented sdu.*/

 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.*/

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

 stat_queue                                      Queue,
 /* Queues used for PDUs associated with STATUS Pdus to be transmitted.*/

 prohibit , rx_prohibit, epc_active              IndicatorType,
 /*An indicator used to determine whether the timer_PROHIBIT
   is running or not.*/

 empty, no_tx, no_retx                           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.*/

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

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

 rx_period                                       DURATION,
 /*The duration of a periodic Statut report generation timer.*/

 rx_prohibdur, epc_dur                           DURATION,
  /*The duration of a prohibit retransmission of status report timer.*/

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

 complete, cnf                                   IndicatorType;
  /*An indicator used to determine whether an SDU has been
     completely reassembled or whether an SDU requires confirmation.*/
```

3GPP TSG-RAN Meeting #6
Nice, France, 13-15 December 1999

Document* *(R2-99k71)
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.322	CR	017r1	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to: TSG-RAN#6	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑	for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)*	(U)SIM		ME	X	UTRAN / Radio	X	Core Network	

Source: TSG-RAN WG2 **Date:** 29/11 1999

Subject: RLC Editorial corrections

Work item:

Category: *(only one category shall be marked with an X)*			**Release:**	
F	Correction		Phase 2	
A	Corresponds to a correction in an earlier release		Release 96	
B	Addition of feature		Release 97	
C	Functional modification of feature		Release 98	
D	Editorial modification	X	Release 99	X
			Release 00	

Reason for change: Correction and clarification of the specification

Clauses affected: 5 / 9.2.1.1 / 9.2.1.2 / 9.2.1.3 / 9.2.1.4 / 9.2.1.5 / 9.2.1.6 / 9.2.2 / 9.2.2.4 / 9.2.2.9 / 9.2.2.12/ 9.2.2.12.6 / 9.7.5 /

Other specs affected:			
Other 3G core specifications		→ List of CRs:	
Other GSM core specifications		→ List of CRs:	
MS test specifications		→ List of CRs:	
BSS test specifications		→ List of CRs:	
O&M specifications		→ List of CRs:	

Other comments:


help.doc

<--------- double-click here for help and instructions on how to create a CR.

5 Functions

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

- ~~Connection Control;~~
- Segmentation and reassembly;
- ~~Header compression;~~Multiple PU:s whithin a RLC PDU
- 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;

The following potential function(s) are regarded as further study items (FFS):

- Suspend/resume function;

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 receving 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 (Reset)

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 (Reset Acknowledge)

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

9.2.1.1 TrD PDU

The TrD PDU transfers user data when RLC is operating in transparent mode. No overhead is added by RLC. The TrD PDU is bit aligned

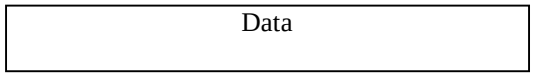


Figure 9-1: TrD PDU

9.2.1.2 UMD PDU

The UMD PDU transfers user data when RLC is operating in unacknowledged mode. The UMD PDU is octet aligned.

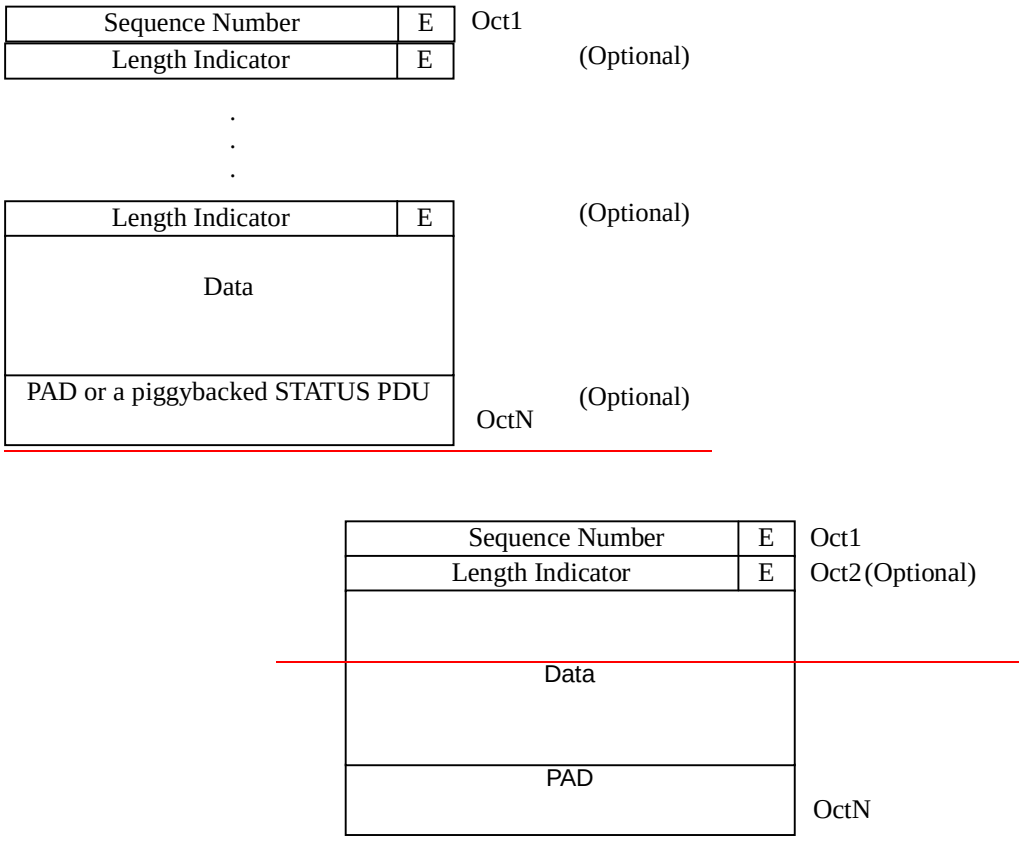
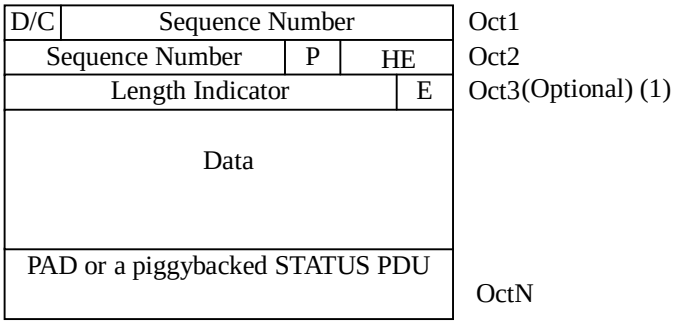


Figure 9-2: UMD PDU

9.2.1.3 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 AMD PDU is octet aligned



NOTE (1): The Length Indicator maybe 15bits.

Figure 9-3: AMD PDU

9.2.1.4 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.

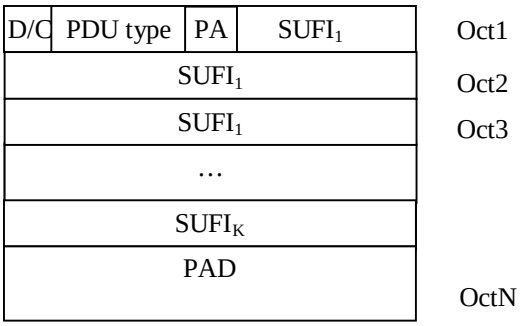


Figure 9-4: Status Information Control PDU (STATUS PDU)

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 AMD PDU is octet aligned

9.2.1.5 Piggybacked STATUS PDU

The format of the piggybacked STATUS PDU is the same as the ordinary STATUS PDU except that the D/C field and the PDU type field is omitted. This PDU can be used to piggyback STATUS PDU in a AMD PDU if the data does not fill the complete AMD PDU. The STATUS PDU is octet aligned

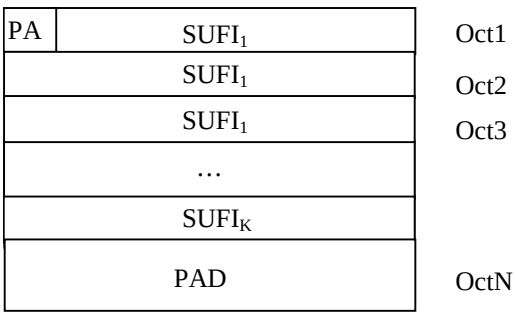


Figure 9-5: Piggybacked STATUS PDU

9.2.1.6 RESET, RESET ACK PDU

The RESET, RESET ACK PDU:S ARE octet aligned

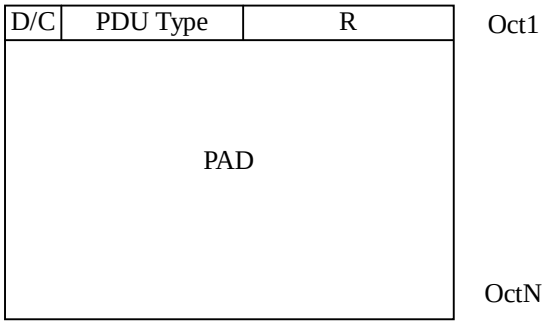


Figure 9-6: RESET, RESET ACK PDU

9.2.2 Parameters

If not otherwise ~~meantioned~~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 most significant and the right most bit is the least significant bit.

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

9.2.2.3 Sequence Number (SN)

This field indicates the sequence number of the payload unit. In a normal AMD-PDU the sequence number of the first PU in the PDU is indicated. If the PUs are not in sequence, a sequence number is indicated separately for each PU in the extended header.

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 (STATUS PDU) 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: 4 bits

This field is used to achieve octet alignment and for this purpose it is coded as 0000. 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 the format of the extended header.

Value	Description
00	The succeeding octet contains data
01	The succeeding octet contains a 7bit length indicator and E bit
10	The succeeding octet contains an extended header field
11	The succeeding octet contains a 15bit length indicator and E bit

9.2.2.7.1 AMD PDU Extended Header

The Extended Header is used when additional sequence numbers are needed to indicate PUs that are not sequential within a PDU or when the rest of a PDU, which is not filled by PUs, is equal or larger than the size of a PU. A PDU that includes more than one sequence number shall include sequence numbers for all PUs in the PDU. The n^{th} sequence number in the PDU indicates the sequence number of the n^{th} PU in the PDU. The decision to use Extended Header is made by the transmitting RLC.

First all the Extended Headers are listed. Then all Length Indicators are listed. Finally the PUs follow.

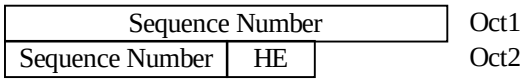


Figure 9-7: Format of the extended header

9.2.2.8 Length Indicator (LI)

This field is optional and is used if concatenation, padding or a piggybacked STATUS PDU takes place in a PU. It indicates the end of the last segment of a SDU. It points out the end of a segment by giving the number of octets between the end of the header fields (including the length indicator fields) and the end of the segment. The size of the Length Indicator may be either 7bits or 15bits. If the last segment of a SDU do not completely fill a PU either padding or a piggybacked STATUS PDU can be added. Predefined values of the length indicator are used to indicate this. The padding/piggybacked STATUS PDU predefined length indicators shall be added after the length indicator that indicates the end of the last SDU segment in the PU. The values that are reserved for special purposes are listed in the tables below depending on the size of the Length Indicator.

If a length indicator that indicates padding/piggybacked STATUS PDU refers to the last PU in the PDU it implicitly means that the rest of the PDU contains padding/piggybacked STATUS PDU. If the last PU in a PDU does not include padding or piggybacked STATUS PDU, but the PDU includes padding or a piggybacked STATUS PDU, an extra length indicator field shall be added as a normal length indicator to the last PU. This extra length indicator shall indicate either padding or a piggybacked STATUS PDU and shall be placed as the last length indicator in the PDU. The space needed for this length indicator shall not be taken from the data part in the PU, but from the padding or piggybacked STATUS PDU in the PDU. The receiving entity shall discard this length indicator.

If RLC PDUs always carry only one PU, 7bit indicators are used in a particular RLC PDU if the address space is sufficient to indicate all SDU segment borders. Otherwise 15bit Length Indicators are applied.

If RLC PDUs may carry more than one PUs the length of the Length Indicator only depends on the size of the largest RLC PDU and the size of the Length Indicator is always the same for all PUs.

Only one size of Length Indicators is used in one RLC PDU.

Length: 7bit

Bit	Description
0000000	The previous RLC PU was exactly filled with the last segment of a RLC SDU.
1111110	The rest part of the RLC PU includes a piggybacked STATUS PDU.
1111111	The rest part of the RLC PU is padding.

Length: 15bit

Bit	Description
000000000000000	The previous RLC PU was exactly filled with the last segment of a RLC SDU.
111111111111110	The rest part of the RLC PU includes a piggybacked STATUS PDU.
111111111111111	The rest part of the RLC PU is padding.

9.2.2.9 Data

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

Transparent mode data:

The RLC SDUs might be segmented. If segmented, then the segmentation is performed according to a predefined pattern. The allowed size for RLC SDUs and segments shall be known. The RLC PDU~~s~~ belonging to one RLC SDU shall be sent in one transmission time interval. Only one RLC SDU is segmented in one transmission time interval.

Unacknowledged mode data and Acknowledged mode data

If a SDU is too large to fit into the data field it is 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 may have any value and the receiving entity shall disregard it.

9.2.2.11 Poll Answer (PA)

Length: 1bit

The PA (Poll Answer) field indicates whether the status report is the answer to a poll or not

Bit	Description
0	The status report is not the answer to a polling request
1	The status report is the answer to a polling request

9.2.2.12 SUFI

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 9-8 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.

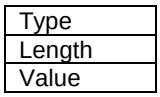


Figure 9-8: The Structure of a Super-Field

The length of the type field is 3 bits and it may have any of following values.

Bit	Description
000	No More Data (**NO_MORE**)
001	Window Size (**WINDOW**)
010	Acknowledgement (**ACK**)
011	List (**LIST**)
100	Bitmap (**BITMAP**)
101	Relative list (**Rlist)**
110	Move Receiving Window **(MRW)**
111	*Reserved ~~for future super-field types~~(PDUs with this coding will be discarded by 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 or the type sub-field.

9.2.2.12.1 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 9-9 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.

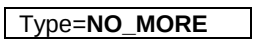


Figure 9-9: NO_MORE field in a STATUS PDU

9.2.2.12.2 The Acknowledgement super-field

The 'Acknowledgement' super-field consists of a type identifier field (ACK) and a sequence number (LSN) as shown in figure 9-10 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 9-10: 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. The LSN should not be set to a value ≥ VR(H). 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 VT(A) will be updated according to the first error indicated in the STATUS PDU.

9.2.2.12.3 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 9-11 below. The receiver is always allowed to change the window size during a connection.

Type = **WINDOW**
WSN

Figure 9-11: The WINDOW fields in a STATUS PDU

WSN

Length: 12 bits

The allowed window size to be used by the transmitter. The range of the window size is [0, 2^{12}-1]. The Window_Size parameter is set equal to WSN.

9.2.2.12.4 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 9-12 below:

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

Figure 9-12: 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.

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.

9.2.2.12.5 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 9-13 below:

Type = **BITMAP**
LENGTH
FSN
Bitmap

Figure 9-13: The Bitmap fields in a STATUS PDU

LENGTH

Length: 4 bits

The size of the bitmap in octets (maximum bitmap size: 2^4*8=128 bits).

FSN

Length: 12 bits

The sequence number for the first bit in the bitmap.

Bitmap

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

Status of the SNs in the interval [FSN, FSN + LENGTH*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*8 - 1]):

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

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

9.2.2.12.6 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 9-14 below.

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

Figure 9-14: 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.

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 ~~from~~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 CW:s will represent the number of subsequent erroneous PU:s (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.

9.2.2.12.7 The Move Receiving Window super-field

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

Type = **MRW**
SN

Figure 9-15: The MRW fields in a STATUS PDU

SN

Length: 12 bits

Requests the RLC receiver to discard all PUs with sequence number < SN, and to move the receiving window accordingly.

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.

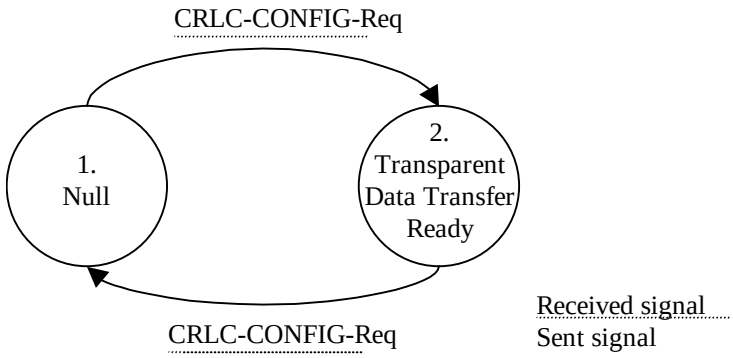


Figure 9-16: The state model for transparent mode entities

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.

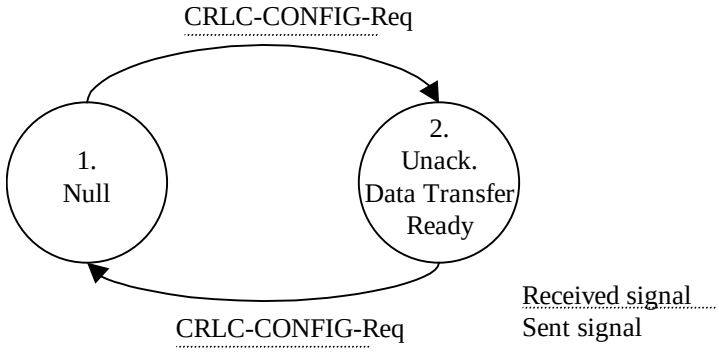


Figure 9-17: The state model for unacknoledged mode entities

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 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, the RLC entity resets the protocol and enters the acknowledged data transfer ready state.

Upon reception of a RESET PDU, the RLC entity resets the protocol, send a RESET ACK PDU and enters the acknowledged data tranfer ready state.

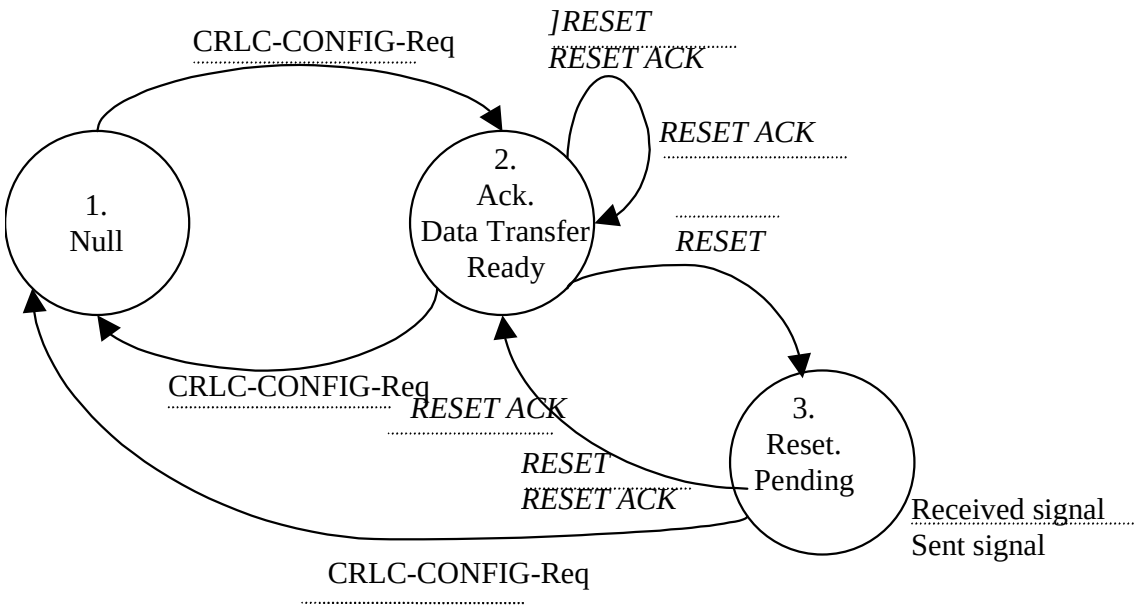


Figure 9-18: The state model for the acknoledged mode entities

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. 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 acknowledgments. VT(A) is updated based on receipt of a STATUS PDU including an 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) + 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 a WINDOW super-field.

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 upon the reception of a RESET ACK PDU. 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 updated upon receipt of the next 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 updated when a new PU is received with 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) + Window_Size. The receiver shall discard PUs with SN $\geq$ VR(MR), (in one case, such a PU may cause the transmission of an unsolicited STATUS PDU).

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 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.

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 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 a new poll is sent when the timer is running it is restarted.

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 activeOnly one poll shall be transmitted when the timer expires even if several polls were triggered when the timer was active. 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 has been sent. The timer is started when a STATUS report is transmitted and when it expires EPC can start decrease (see section 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. If the SDU has not been acknowledged when the timer expires, the SDU is discarded and a Move Receiving Window request is sent to the receiver. If the SDU discard function does not use the Move Receiving Window request, the timer is also used in the receiver, where it is activated once a PDU is detected as outstanding, i.e. there is a gap between sequence numbers of received PDUs. 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 and the timer is restarted. The value of the timer is signalled by RRC.

f) 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.

g) 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.

h) 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.

9.6 Protocol Parameters

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 transmitted when:

$$1 - \frac{(Window\_Size + VT(MS) - VT(S)) \bmod Window\_Size}{Window\_Size} > Poll\_Window.$$

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) Window_Size

The maximum allowed transmitter window size.

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 PDU. 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 no STATUS PDU has been received 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.

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 PDU transmission for acknowledged mode

The receiver of AMD PDUs transmits 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 PDU. 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 PDU 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 PDU to the sender.

2) Timer based STATUS PDU transfer

   The receiver transmits a STATUS PDU periodically to the sender. The time period is controlled by the timer Timer_Status_Periodic.

3) The EPC mechanism

   The EPC is started when a STATUS PDU is transmitted to the peer entity. If not all PUs requested for retransmission have been received before the EPC has expired a new STATUS PDU is transmitted to the peer entity. A more detailed description of the EPC mechanism is given in section 9.7.2.

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

1) STATUS PDU prohibit

   The Timer_Status_Prohibit is started when a STATUS PDU is transmitted to the peer entity. As long as the timer is running the receiving side is not allowed to send a STATUS PDUs to the peer entity. The STATUS PDU is transmitted after the timer has expired. The receiver shall only send information about a PU once, even if there are several triggers when the timer running.

2) The EPC mechanism

   If the EPC mechanism is active and the sending of a STATUS PDU is triggered it shall be delayed until the EPC mechanism has ended. The receiver shall only send information about a PU once, even if there are several triggers when the timer is active or the counter is counting down.

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, in the case of non-transparent transmission mode. 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 section 9.2), and piggy backed to status information of transmissions in the opposite direction. 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 section 9.7.3.1. The difference is that this discard method does not use any peer-to-peer signalling. For unacknowledged mode RLC, peer-to-peer signalling is never needed. The SDUs are simply discarded in the transmitter, once the transmission time is exceeded. For acknowledged mode RLC, peer-to-peer signalling can be avoided as long as SDU discard is always performed in the transmitter before it is performed in the receiver. As long as the corresponding SDU is eventually discarded in the receiver too, possible retransmission requests of PDU of discarded SDUs can be ignored by the transmitter. The bigger the time difference is between the triggering of the discard condition at the transmitter and the receiver, the bigger the unnecessary buffering need is at the receiver and the more bandwidth is lost on the reverse link due to unnecessary retransmission requests. On the other hand, forward link bandwidth is saved, as no explicit SDU discard signalling is needed.

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 PDU 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 PDU 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 (Status PDU). 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 PDU will be transmitted and the EPC will be reset accordingly. The EPC timer will be started once more.

The EPC is based on the estimation of the number of PUs that should have been received during a transmission time interval. To estimate this number is easiest done by means of the TFI bits. However, if these bits are lost due to some reason or another, this estimation must be based on something else. A straightforward solution is to base the estimation on the number done in the previous transmission time interval. Only if the rate has changed this estimation is incorrect. Another method of estimating the number of PUs is based on the maximum allowable rate. The consequence of this is that if the estimation is incorrect, the Status PDU is sent too early. Alternatively, the estimation can be based on the lowest possible transmission rate. In this case, if the estimation is incorrect, the Status PDU will most likely be transmitted too late.

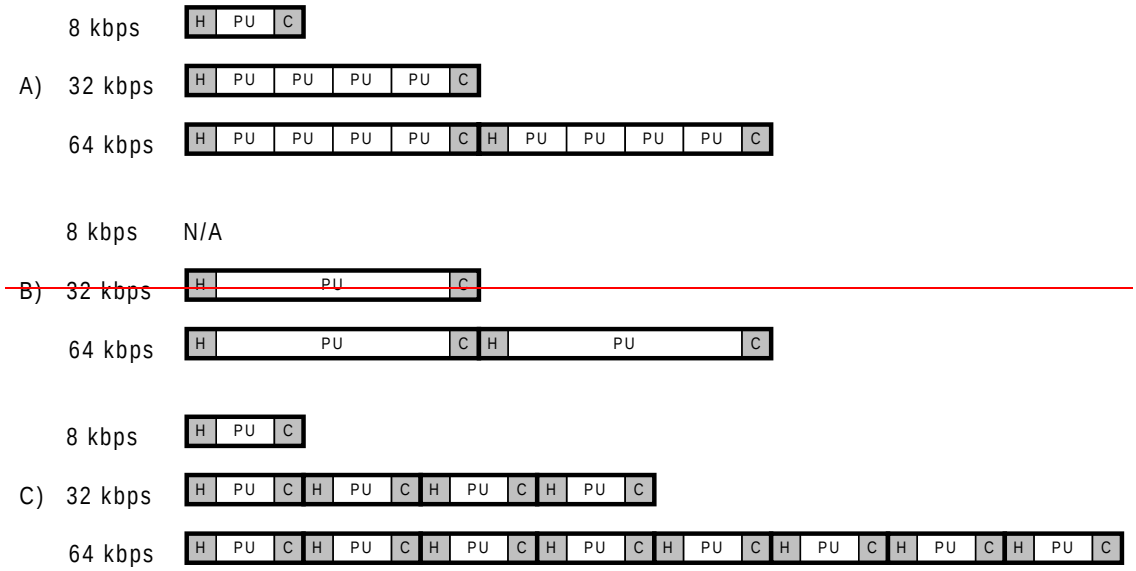
9.7.5 Multiple payload units in a RLC PDU ~~and header compression~~

The possibility to include multiple payload units (PU) into one RLC AMD PDU ~~provides a way to support variable bit rate services with lower overhead. The method~~ is ~~to be a~~ part of the service capabilities of a UE ~~capable to support user plane traffic~~ in acknowledged mode. For Release 99, there shall be only one PU per AMD PDU.

A ~~semi-static sized~~ 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 segmentation and concatenation of SDUs into the RLC transmitter buffer is always performed as if there would be only one PU in each AMD PDU. However one AMD PDU may contain also multiple PUs. The header compression is applied to incorporate several PUs effectively into the AMD PDU and it takes place when the transport block size for the next transmission time interval indicated by MAC can accommodate several PUs. The concept of having multiple PUs in one AMD PDU does not remove the need to still have several transport blocks (RLC PDUs) in the transport block set.~~

~~Determination of t~~The ~~configuration~~ of PU and PDU sizes is ~~determined~~ based on ~~by~~ the needed transmission rates. The size of the PU is set by the RRC. ~~is derived directly from the lowest common block size that can be used to compose all the applicable data rates. To support faster transmission rates more payload units are combined into one AMD PDU with header compression function as long as the size of the PDU reaches the optimum. When higher transmission rates are used, several transport blocks (RLC PDUs) are delivered within one transport block set keeping the amount of PUs the same in one AMD PDU. The AMD PDU Extended Header is needed when several PU:s out-of-sequence is included into one RLC PDU. When multiple PU:s in one RLC PDU is used then the RLC is able to construct PDU:s of variable size.~~

~~The method of multiple PUs in one PDU combined with the header compression is especially useful when the lowest needed transmission bit rate cannot be effectively supported with PDUs optimized for the highest transmission bit rate. Figure 3-1 presents a few examples illustrating the applicability of the method. To illustrate the method, figure 9-19 shows some examples where the RLC AMD PDU size has been set to 320 bits.~~



~~Figure 9-19: Example cases of payload units and PDUs at different transmission rates~~

~~Assuming there is a service having a need for variable transmission rate with the maximum above 32 kbps (e.g. 64 kbps) and the minimum of e.g. 8 kbps. With the PU size of 80 bits the flexibility for transmission rates of 8 kbps, 16 kbps or 32 kbps can be achieved by adjusting simply the amount of PUs in one RLC PDU. This also affects on the size of the RLC PDU. While having the optimum sized PDU at 32 kbps, faster transmission rates are constructed by sending several PDUs, each of them containing four PUs (case A).~~

~~In case B no data rates below 32 kbps are in the transport format set and the size of the PU is the same as the size of the payload in the RLC PDU. To support transmission bit rates above 32 kbps several PDUs (transport blocks) are sent within one transport block set.~~

~~Without the capability to append several PU:s in the PDU the overhead becomes rather high if several transport blocks are sent within one transport block set without header compression at a higher transmission rate (case C).~~