```
{               Estelle Specification of MIL-STD-188-220B

             Datalink Layer Class B (Types 1 and 2) Operation

                    Protocol Engineering Laboratory
                  Department of Computer and Information Science
                             University of Delaware

                                  Paul D. Amer
                                  Greg Burch
                                  Adarsh Sethi
                                  Dong Zhu
                                  Mariusz Fecko
                                  M. Umit Uyar

                                  Date: 6/14/96
                                  Revised: 2/27/97
                                      removal of XID, NETCON, join/leave net
                                      response mode becomes quiet mode
                                  Revised: Dec 97 Update DL.Unitdata/Status.primitives
                                      in accordance with 19Dec97 comment resolution



Note: * denotes a recommendation not currently specified in MIL-STD-188-220A.
Definitions: (for convenience of reference and consistency in the Estelle code.)
   request: This is an internal item used by the SAP component only. The "queue"
            is, as the name implies, a sequence of outstanding requests. The
            request is a representation of a Protocol Request
            (e.g., DL_Unitdata_Req) which the SAP component knows how to handle.
            Included in the request is the request_id, which is the id assigned
            to the Protocol Request by the entity that initiated it
            (e.g., OPERATOR, NL).
   message: An internal exchange of information between the SAP component and
            Station component of the Estelle architecture. These "messages" are
            used only locally, and consist of a DL_PDU, unique message_id, and
            P-bit (from SAP to Station only).
   transmission: This signifies the bit pattern that the datalink layer sends to
            physical layer; as such, a transmission consists of a transmission
            header, and one or more "DL_PDU"'s (after they have seen
            bit-stuffing, FEC, TDC, and scrambling).
   DL_PDU:  datalink layer PDU (with length information) before applying FEC,
            TDC, data scrambling and 0 bit stuffing.

Other: There are two data structures used implicitly by both the type2SAP and
       type1SAP components. These structures are not declared explicitly, and
       as such their detailed contents and organizations are not listed here.
       The first such structure is the "queue." This structure is, as the name
       suggests, a FIFO list of outstanding requests (Test Requests, Unitdata
       Requests) waiting to be handled by the type1SAP component. The second
       structure does not have a name and is referenced by procedures such as
       "add_message." This structure holds information about all outgoing PDU's
       requiring acknowledgements (including transmission count, which stations
       have responded, what the DL_Unitdata_id is for the corresponding request
       which resulted in the construction of that DL_PDU, etc.)

       Also, we have implemented the busy-status table and timers as
       2 orthogonal timers (one for type 1 and one for type 2). When
       a type 1 URNRC or URNRR message comes in, a copy is forwarded to
       both the Type 1 and Type 2 SAP components, and both mark their
       stations as busy. When a type 1 CLEAR (URRC or whatever) comes in,
       no copy is forwarded to the type2SAP component... it must wait to
       timeout. This method of specifying the busy-status table does not
       preclude a one-timer, one-table implemention.
}

specification datalink_layer systemactivity;

default
   individual queue;
   timescale millisecond;

const
   K = any integer;  { 1-127; the maximum number of outstanding PDU's in type 2 }
   N4 = any integer; { Type 1 max retransmission count }
   N2 = any integer; { Type 2 max retransmission count }
   NS = any integer; { Number of stations > 0         }
   NS_LESS_1 = any integer; { NS - 1 }
   NULL_ID = -1;            { No ID needed for NETCON or station message }
   NEED_COUPLED_ACK = 1;
   NOT_NEED_COUPLED_ACK = 0;

   { Estelle-defined transition firing priorities }
   LOW = 2;
   MEDIUM = 1;
   HIGH = 0;

   { Timer values }
   JOIN_TIME_WHEN_P_IS_0 = any integer;
   JOIN_TIME_WHEN_P_IS_1 = any integer;
   BUSY_TIMER_TIMEOUT_PERIOD = any integer;

   { Link addresses }
   GLOBAL_MULTICAST_ADDRESS = 127;
   NETCON_ADDRESS = 2;
   NET_ENTRY_ADDRESS = 1;
   ALL_ADDRESSEE_LIST_SIZE = any integer;
   MAX_JOIN_REQUEST_QUEUE_LENGTH = any integer;
   MAX_NO_OF_DEST_ADDRESS = 16;
   STATION_MESSAGE_TABLE_SIZE = any integer;
   TRANSMISSION_TIMER_LIST_SIZE = any integer;

type
   type_statename = ( ADM_state, SETUP_state, D_CONN_state, RESET_state, ERROR_state,
      CONN_state, RESET_CHECK_state, RESET_WAIT_state, ISO_TRANSFER_STATES_states );
   type_station_number_range = 3..95; { The space of individual addresses }
   type_type2_retransmission_data_buffer = ...;
   type_incoming_type2_DL_PDU_type = ( DISC_CMD, DM_RSP, FRMR_RSP, I_CMD, I_RSP,
      REJ_CMD, REJ_RSP, RNR_CMD, RNR_RSP, RR_CMD, RR_RSP, SABME_CMD, UA_RSP, RSET_CMD,
      SREJ_CMD, SREJ_RSP, REJ_XXX, SREJ_XXX );
   type_type2_out_of_order_data_buffer = ...;
   type_type2_sequence_number = 0..127;
   type_FRMR_reason = (ILLEGAL_PDU, EXCEED_MAX_INFO_LENGTH, N_R_ERROR, N_S_ERROR);
   type_switch = 0..1;
   type_type2_unitdata_buffer = ...;
      { Connection component buffer for unprocessed data request }
   type_multi_info = ...; { Information needed by Type2SAP component for
                            processing a multi-destination I PDU }
   type_type2_connection_status = (ESTABLISHED, DISCONNECTED, RESET);
   type_type2_timers_needed = record
                                 ack2_timer_needed: 0..1;
                                 P_timer_needed  : 0..1;
                                 REJ_timer_needed: 0..1;
                              end;
   type_type2_timer_list =
      array [type_station_number_range] of type_type2_timers_needed;
      { Timer information for a Type 2 PDU; needed by the station component to start
        timers (Ack/P/REJ timers) when the PDU is transmitted }
   type_transmission_type2_timer_list =
```

```
   array [0..TRANSMISSION_TIMER_LIST_SIZE, type_station_number_range]
   of type_type2_timers_needed;
   { Type 2 timer information for a transmission; needed by the station component to
     start timers (Ack/P/REJ timers) when transmission is successful }
type_type2_reset_originator = (REMOTE, LOCAL);
type_type2_report_status = (FRMR_RECEIVED, FRMR_SENT, REMOTE_BUSY, MEMOTE_NOT_BUSY);
type_type2_multi_response = (READY, NOT_READY);
type_type2_PDU_control_info = ...; { PDU control information (needed for
                                     multi-destination I PDU operation) }
type_8_bit_field =  integer; { More precisely: array [0..7]  of bit }
type_16_bit_field = integer; { More precisely: array [0..15] of bit }
type_32_bit_field = integer; { More precisely: array [0..31] of bit }
type_DL_Unitdata_id = integer; { * }
type_FEC_TDC_scrambling = type_8_bit_field;
type_P_bit = 0..1;
type_DL_PDU_type = (UI_COMMAND,
                    URR_COMMAND,  URR_RESPONSE,
                    URNR_COMMAND, URNR_RESPONSE,
                    TEST_COMMAND, TEST_RESPONSE);
type_address = type_8_bit_field;
type_all_addressee_list = array [1..ALL_ADDRESSEE_LIST_SIZE] of
                            record
                            address: type_address;
                            P_bit: type_P_bit;
                            end;
type_coupled = (COUPLED, NOT_COUPLED); { * }
type_data = ...; { variable length bit string }
type_data_length = integer; { length of bit string of type type_data }
type_delay = (DELAY_NORMAL, DELAY_LOW);
type_fail =  (FAIL, NOT_FAIL);
type_from_component = (FROM_TYPE1SAP, FROM_TYPE2SAP);
type_DL_PDU_addressee_list = array [1..MAX_NO_OF_DEST_ADDRESS] of
                            record
                            address: type_address;
                            P_bit: type_P_bit;
                            end;
type_id = integer; { * }
type_individual_address = 3..95; { 1-Net Entry; 2-NETCON; 96~126-Groups; 127-Global }
type_link_parameter_table = record
                           message_number: type_8_bit_field;
                           unique_id: type_32_bit_field;
                           group_address: type_32_bit_field;
                           link_address: type_8_bit_field;
                           station_class: type_8_bit_field;
                           robust_frame_format: type_8_bit_field;
                           EDC_mode: type_8_bit_field;
                           net_busy_detect_time: type_16_bit_field;
                           equipment_preamble_time: type_16_bit_field;
                           key_time: type_16_bit_field;
                           equipment_postamble_time: type_16_bit_field;
                           carrier_dropout_time: type_16_bit_field;
                           equipment_lag_time: type_16_bit_field;
                           tolerance_time: type_16_bit_field;
                           processing_time: type_8_bit_field;
                           NAD_method: type_8_bit_field;
                           subscriber_rank: type_8_bit_field;
                           number_of_stations: type_8_bit_field;
                           urgent_percent: type_8_bit_field;
                           priority_percent: type_8_bit_field;
                           traffic_load: type_8_bit_field;
                           max_transmit_seconds: type_8_bit_field;
                           physical_concatenation: type_8_bit_field;
                           data_link_concatenation: type_8_bit_field;
                           max_UI_DIA_I_info_bytes: type_16_bit_field;
                           type2_ack_timer: type_16_bit_field;
                           type2_K_window: type_8_bit_field;
                           type4_ack_timer: type_16_bit_field;
                           type4_K_window: type_8_bit_field;
                           quiet_mode: type_8_bit_field;
                           end;
   type_message_id = integer; { * }
   type_multidwell = (SIX, ELEVEN, THIRTEEN);
   type_net_activity = (CLEAR, BUSY, BUSY_WITH_DATA, BUSY_WITH_VOICE);
   type_unique_id = type_32_bit_field;
   type_precedence =  (PREC_URGENT, PREC_PRIORITY, PREC_ROUTINE);
   type_reliability = (REL_NORMAL, REL_HIGH);
   type_quiet_mode = (OFF, ON);
   type_rexmis_count = 0..N4;
   type_throughput = (THRU_NORMAL, THRU_HIGH);
   type_time = integer;
   type_topology_update_id = ...;
   type_transmission_status = (TRANSMIT_COMPLETE_OR_IDLE, TRANSMIT_IN_PROCESS,
                               TRANSMIT_ABORTED);
   type_DL_PDU = record
                 DL_PDU: type_data;
                 length: type_data_length;
                 end;
   type_station_message_table = array [1..STATION_MESSAGE_TABLE_SIZE ] of
                                   record
                                   message_id: type_message_id;
                                   message: type_DL_PDU;
                                   from_component: type_from_component;
                                   precedence: type_precedence;
                                   topology_update_id: type_topology_update_id;
                                   P_bit: type_P_bit;
                                   end;
   type_ack_failure = type_fail;
   type_addr_list = record
                    number_of_address: integer;
                    address: array [0..NS_LESS_1] of type_address;
                    end;
   type_assigned_address_table = array [type_individual_address] of
                                   record
                                   unique_id: type_unique_id;
                                   address: type_address;
                                   end;
   type_busy = (STATION_BUSY, STATION_FREE);
   type_type2_busy_status = (TYPE2_BUSY, TYPE2_NOTBUSY);
   type_queued_request = ...;
   type_request_type = (DL_REQUEST, TEST_REQUEST);
   type_incoming_DL_PDU_type = (UIC, TESTR, TESTC, URRC, URRR, URNRC, URNRR);
   type_test_result =  (SUCCESS, FAILURE);
   type_service_type = (TYPE1ACK, TYPE1NOACK, TYPE2P0, TYPE2P1, TYPE4, TYPE2);
   type_ack_required = (YESACK, NOACK);
   type_DRR_DRNR_last_sent = (DRR, DRNR);

{*******************************************************************************}
{     Functions/Procedures Used In Module m_type2_connection_component          }
{*******************************************************************************}
{ Check if DL_PDU is a bad_PDU }
function bad_PDU(DL_PDU: type_DL_PDU): boolean; primitive;

{ Buffer unitdata_request }
procedure buffer_unitdata_request(var unitdata_buffer: type_type2_unitdata_buffer);
   primitive;

{ Buffer multidata_request }
procedure buffer_multidata_request(var unitdata_buffer: type_type2_unitdata_buffer);
   primitive;
```

```
{ Test for the condition for ISO OPTIONAL_SEND_RNR_XXX(X=0) }
function choose_to_send_RNR_XXX: boolean; primitive;

{ Test for the condition for sending RR in ISO RESEND_I_...OR_SEND_RR }
function choose_to_send_RR: boolean; primitive;

{ Construct I PDU for the first transmission }
function construct_1_I_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_1_I_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_1_I_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_ACKNOWLEDGE_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_ACKNOWLEDGE_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_ACKNOWLEDGE_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_DISC_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_DM_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_FRMR_RSP(F_bit: type_switch): type_DL_PDU; primitive;

{ Construct control field of PDU (type2SAP will collect all such information
  from different connection component and construct a multi-destination PDU }
function construct_PDU_control(PF_bit: type_switch): type_type2_PDU_control_info;
   primitive;

function construct_RNR: type_DL_PDU; primitive;
function construct_RNR_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_RNR_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_RNR_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_REJ_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_REJ_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_REJ_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_RR: type_DL_PDU; primitive;
function construct_RR_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_RR_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_RR_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_RSET_CMD: type_DL_PDU; primitive;
function construct_SABME_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_SREJ_CMD(P_bit: type_switch): type_DL_PDU; primitive;
function construct_SREJ_RSP(F_bit: type_switch): type_DL_PDU; primitive;
function construct_SREJ_XXX(PF_bit: type_switch): type_DL_PDU; primitive;
function construct_UA_RSP(F_bit: type_switch): type_DL_PDU; primitive;

{ Check if there is any DATA_REQUEST }
function exist_buffered_data_request(unitdata_buffer: type_type2_unitdata_buffer):
   boolean; primitive;

{ Check if there is any I PDU still waiting for acknowledgement }
function exist_unacked_PDU: boolean; primitive;

function FRMR_reason(DL_PDU: type_DL_PDU): type_FRMR_reason; primitive;

{ Get N(R) of a DL-PDU }
function get_N_R(DL_PDU: type_DL_PDU): type_type2_sequence_number; primitive;

{ This could be the precedence field in DL_Unitdata_Req; or if a control PDU
  is to be transmitted, choose an appropriate precedence }
function get_precedence: type_precedence; primitive;

{ This could be the topology_update_id field in DL_Unitdata_Req; or if a control PDU
  is to be transmitted, use the newest topology_update_id from network layer }
function get_topology_update_id: type_topology_update_id; primitive;

function get_type2_PDU_type(DL_PDU : type_DL_PDU): type_incoming_type2_DL_PDU_type;
   primitive;

{ Initiate out_of_order_data_buffer }
procedure init_out_of_order_data_buffer(
   var out_of_order_data_buffer: type_type2_out_of_order_data_buffer); primitive;

{ Initiate retransmission_data_buffer }
procedure init_retransmission_data_buffer(
   var retransmission_data_buffer: type_type2_retransmission_data_buffer); primitive;

{ Initiate unitdata_buffer }
procedure init_unitdata_buffer(
   var unitdata_buffer: type_type2_unitdata_buffer); primitive;

function initiate_P_F_cycle: boolean; primitive;

{ Check if PDU is a response PDU with F=1 }
function is_F_bit_1(DL_PDU: type_DL_PDU): boolean; primitive;

{ Check if PDU is a command PDU with P=1 }
function is_P_bit_1(DL_PDU: type_DL_PDU): boolean; primitive;

{ Check if Selective Reject is used }
function is_SREJ_used: boolean; primitive;

{ Check if another packet may be sent without exceeding the send window }
function is_send_window_open: boolean; primitive;

{ Check if the DL_PDU is an unlisted XXX PDU in state statename }
function is_XXX_CMD(DL_PDU: type_DL_PDU; statename: type_statename): boolean;
   primitive;
function is_XXX_RSP(DL_PDU: type_DL_PDU; statename: type_statename): boolean;
   primitive;
function is_XXX_YYY(DL_PDU: type_DL_PDU; statename: type_statename): boolean;
   primitive;
function is_ZZZ_CMD(DL_PDU: type_DL_PDU): boolean; primitive;
function is_ZZZ_RSP(DL_PDU: type_DL_PDU): boolean; primitive;

function local_busy_detected: boolean; primitive;
function local_busy_cleared: boolean; primitive;

{ Get sequence number of the I PDU which needs to be retransmitted next }
function next_unacked_seqno: type_type2_sequence_number; primitive;

function no_more_buffered_out_of_order_data(
   out_of_order_data_buffer: type_type2_out_of_order_data_buffer): boolean; primitive;

procedure process_and_buffer_out_of_order_data(
   DL_PDU: type_DL_PDU;
   var out_of_order_data_buffer: type_type2_out_of_order_data_buffer); primitive;

procedure process_buffered_out_of_order_data(
   var out_of_order_data_buffer: type_type2_out_of_order_data_buffer); primitive;

function reconstruct_1_I_CMD(P_bit: type_switch; N_S: type_type2_sequence_number):
   type_DL_PDU; primitive;

function reconstruct_1_I_RSP(F_bit: type_switch; N_S: type_type2_sequence_number):
   type_DL_PDU; primitive;

function reconstruct_1_I_XXX(PF_bit: type_switch; N_S: type_type2_sequence_number):
   type_DL_PDU; primitive;

{ Check if the retransmission of all unacknowledged I_PDUs is finished }
function resend_I_PDU_finished: boolean; primitive;

{ Choose either 1 or 0 for P_bit of a command PDU }
function select_P_bit: type_switch; primitive;
```

```
{ Construct timer_list which contain only one none-null element.
  NB, for multi-destination PDU timer_list could contain > 1 none-null elements  }

{ Timer information for a Type 2 PDU; needed by the station component to start
  timers (Ack/P/REJ timers) when the PDU is transmitted }
function timers_needed_according_to_PDU_sent(DL_PDU : type_DL_PDU):
   type_type2_timers_needed; primitive;

{ Construct timers-to-be-started information for a type 2 DL-PDU; this information is
  needed by station component to start timers when the PDU is transmitted }
function ack_P_REJ_timers_needed(
   ack2_timer_needed: type_switch;
   P_timer_needed  : type_switch;
   REJ_timer_needed: type_switch): type_type2_timers_needed; primitive;

{ If the N(R) of the received PDU acks any previously unacked PDUs, then
  update local record of N(R)_RECEIVED }
procedure update_N_R; primitive;

{ Check if N(S) is out of order }
function with_unexpected_N_S(DL_PDU: type_DL_PDU): boolean; primitive;

{ Check if N(S) is invalid }
function with_invalid_N_S(DL_PDU: type_DL_PDU): boolean; primitive;

{ Check if N(R) is invalid }
function with_invalid_N_R(DL_PDU: type_DL_PDU): boolean; primitive;

{*****************************************************************************}
{     Functions/Procedures Used In Module m_type2SAP_component                }
{*****************************************************************************}
{ Check if all multi-responses are received from all involved connection components }
function are_all_multi_responses_received(v_multi_info: type_multi_info): boolean;
   primitive;

{ Construct a multi-destination I PDU }
function construct_DL_PDU_from_multi_info(
   multi_info: type_multi_info): type_DL_PDU; primitive;

{ Construct timer_list which contain only one none-null element.
  NB, for multi-destination PDU timer_list could contain > 1 none-null elements  }
function construct_timer_list_from_timers_needed(
   timers_needed: type_type2_timers_needed): type_type2_timer_list; primitive;

{ Obtain destination address (= connection component number) from DL-PDU }
function get_connection_component_number(PDU: type_DL_PDU): integer; primitive;

{ Get type 2 connection status for connection component i }
function get_connection_status(i: integer): type_type2_connection_status; primitive;

{ Get the only destination from destination list }
function get_destination(address_list: type_addr_list): type_address; primitive;

function get_precedence_from_multi_info(
   multi_info: type_multi_info): type_precedence; primitive;

function get_topology_update_id_from_multi_info(
   multi_info: type_multi_info): type_topology_update_id; primitive;

{ Check if destination address list contains multi-destinations }
function has_multi_destinations(address_list: type_addr_list): boolean; primitive;

{ Set null_ack_failure to NOT_FAIL }
procedure init_null_ack_failure(var null_ack_failure: type_ack_failure); primitive;

{ Set null_DL_Unitdata_id to null }
procedure init_null_DL_Unitdata_id(var null_DL_Unitdata_id: type_DL_Unitdata_id);
   primitive;

{ Set null_address_list to null list }
procedure init_null_address_list(var null_address_list: type_addr_list);
   primitive;

{ Initiate timer_list so that it could be used to record timer information for a new
  multi-destination PDU }
procedure init_timer_list(var timer_list: type_type2_timer_list); primitive;

{ Check if remote connection request is acceptable for connection_component i }
function is_connect_acceptable(i: integer): boolean; primitive;

function is_destination_in_dest_addr_list(
   i: integer;  dest_addr_list: type_addr_list): boolean; primitive;

{ Check if reset is acceptable for connection_component i; if not acceptable,
  disconnect should be initiated }
function is_reset_acceptable(i: integer): boolean; primitive;

{ Check if a multi-response is pending from connection_component i }
function is_waiting_multi_response(i: integer): boolean; primitive;

{ Record connection status information for connection_component i }
procedure record_connection_status(i: integer); primitive;

{ Record the multi-destination data request for later processing }
procedure record_DL_Unitdata_Req_for_multi_processing(var multi_info: type_multi_info);
   primitive;

{ Record multi_not_ready information for connection_component i, which essentially
  means deleting the destination address i from multi-destination list and leave
  the transmission to the connection component }
procedure record_multi_not_ready(i: integer; var multi_info: type_multi_info);
   primitive;

{ Record multi_ready information for connection_component i }
procedure record_multi_ready(
   i: integer; PDU_control: type_type2_PDU_control_info;
   timers_needed: type_type2_timers_needed;
   var multi_info: type_multi_info; var timer_list: type_type2_timer_list); primitive;

{ Record report status information for connection_component i }
procedure record_report_status(
   i: integer; report_status: type_type2_report_status); primitive;

{ Check if there is a need for connection_component i to initiate a connection }
function should_init_connect(i: integer): boolean; primitive;

{ Check if there is a need for connection_component i to close a connection }
function should_init_disconnect(i: integer): boolean; primitive;

{ Check if there is a need for connection_component i to initiate a reset }
function should_init_reset(i: integer): boolean; primitive;

{ Return the connection component number for this station }
function whoami: integer; primitive;

{*****************************************************************}
{     Functions/Procedures Used In Module m_station_component     }
{*****************************************************************}
{ Get DL_PDU type of a given DL_PDU }
function get_DL_PDU_type(data: type_DL_PDU): type_DL_PDU_type; primitive;
```

```
{ Get P/F bit of a given DL_PDU }
function get_PF_bit(message: type_DL_PDU): type_P_bit; primitive;

{ Get the SAP a PDU is aimed for }
function get_typeSAP(station_message: type_DL_PDU): type_service_type; primitive;

{ Get source address of a given DL_PDU }
function get_src_address(station_message: type_DL_PDU): type_address; primitive;

{ Get Unique ID of a station who send the DL_PDU }
function get_unique_id(DL_PDU: type_DL_PDU): type_unique_id; primitive;

{ Determine whether the type1 message is a URNRC or URNRR PDU }
function URNR_message(station_message: type_DL_PDU): boolean; primitive;

{*****************************************************************************}
{     Functions/Procedures Used In Module m_type1SAP_component                }
{*****************************************************************************}
procedure add_message(message_id : integer;
                      message : type_DL_PDU;
                      NL_id: type_id); primitive;
{ Enter a type1SAP message into an unspecified structure used only by the type1SAP
  component. Set the list of received acks for that message's corresponding
  DL_PDU to nil. Set the transmission count for that message's corresponding
  DL_PDU to 1.  The NL_id is the ID given back to the NL if there is an eventual
  ack_failure sent up.}

function addr_index(address : type_address) : integer; primitive;
{ Access an unspecified structure to get index into the BUSY array for given address }

function addr_list_size(addr_list: type_addr_list) : integer; primitive;
{ Return the number of addresses in list addr_list }

function all_acked(message_id : integer) : boolean; primitive;
{ Return TRUE if all acks for msg_id are received, else return FALSE. }

function choose_message_id : integer; primitive;
{ Return an unused message id number }

function create_busy_address_list(addr_list: type_addr_list): type_addr_list; primitive;
{ Take an address list, check all entries against the array BUSY.
  return an address list containing all busy addresses.}

function create_non_busy_address_list(addr_list: type_addr_list) :
   type_addr_list; primitive;
{ Take an address list, check all entries against the array BUSY.
  return an address list containing all non-busy addresses.}

function create_queued_request(addr_list : type_addr_list;
                               src_addr : type_address;
                               precedence : type_precedence;
                               acknoack : type_ack_required;
                               data : type_data;
                               data_length : type_data_length;
                               req_type : type_request_type;
                               request_id : type_id) :
   type_queued_request; primitive;
{ Make a request from the given information }

function create_type1SAP_message(userdata : type_data;
                                 addr_list : type_addr_list;
                                 data_length : type_data_length;
                                 P_bit: type_P_bit) :
   type_DL_PDU; primitive;
{ Make a message for the station component out of the given arguments.}

function create_updated_type1SAP_message(message_id : integer) : type_DL_PDU; primitive;
{ Take an already existing message id and create a new type1SAP message,
  changing the address list to include only those addresses which
  have not yet acked (using the ack info for that id as well). }

function get_DL_Unitdata_id_from_id(message_id : integer) :
         type_DL_Unitdata_id; primitive;
{ For abstraction purposes. }

function get_addr_from_list({var} addr_list: type_addr_list) : type_address; primitive;
{ Remove the first address from the list addr_list, and return it.
  side effect of changing addr_list IS permanent and intended. }

function get_data(message : type_DL_PDU) : type_data; primitive;
{ For abstraction purposes. }

function get_data_length(message : type_DL_PDU) : type_data_length; primitive;
{ For abstraction purposes. }

function get_dest_addr_list(message : type_DL_PDU) : type_addr_list; primitive;
{ For abstraction purposes. }

function get_message_id_being_acked(message : type_DL_PDU): integer; primitive;
{ Get message_id being responded to in current URR or URNR
  sent from the station component in argument "message". }

function get_message_transmission_count(message_id : integer) : integer; primitive;
{ Return transmission count for the DL_PDU assoc. with id message_id }

function get_message_type(message : type_DL_PDU) :
   type_incoming_DL_PDU_type; primitive;
{ For abstraction purposes. }

function get_msg_precedence(message_id : integer) : type_precedence; primitive;
{ Return the message precedence assoc. with id message_id }

function get_nonresponding_addresses(message_id : integer) : type_addr_list; primitive;

function get_request_DL_Unitdata_id(request : type_queued_request) :
         type_DL_Unitdata_id; primitive;
{ For abstraction purposes. The request argument MUST have type DL_REQUEST }

function get_request_OP_test_req_id(request : type_queued_request) : type_id; primitive;
{ For abstraction purposes. The request argument MUST have type TEST_REQUEST }

function get_request_data(request: type_queued_request) : type_data; primitive;
{ For abstraction purposes. }

function get_request_data_length(request : type_queued_request) :
         type_data_length; primitive;
{ For abstraction purposes. }

function get_request_nonresponding_addresses(request: type_queued_request) :
         type_addr_list; primitive;
{ Return all addresses in a queued request (they are non-responding, since
  this is only called when leaving the net, and hence the request is not
  going to be handled). }

function get_request_src_addr(request: type_queued_request) : type_address; primitive;
{ For abstraction purposes. }

function get_request_type(request : type_queued_request) : type_request_type; primitive;
{ For abstraction purposes. }
```

```
function get_src_addr(message : type_DL_PDU) : type_address; primitive;
{ For abstraction purposes. }

procedure increment_trans_count(msg_id : integer); primitive;
{ Increment transmission count for the DL_PDU assoc. with id message_id }

function make_addr_list(address : type_address) : type_addr_list; primitive;
{ Type conversion from type_address to type_addr_list }

procedure process_queue_entry; primitive;
{ Similar to transition TYPE1SAP_2 below. Must split the address list into busy
  and nonbusy addresses, create type1SAP message for non-busy destinations, and
  requeue the balance of the request for later try }

procedure queue_request(request : type_queued_request); primitive;
{ Add the request to the "queue" }

procedure remove_message(msg_id : integer); primitive;
{ Remove message with (unique) id msg_id from the type1SAP structure }

function remove_queued_request : type_queued_request; primitive;
{ Remove and return head of queue. Decrease variable current_no_queue_entries by 1. }

procedure update_ack_list(msg_id : integer; addr : type_address); primitive;
{ Mark addr as having replied to id msg_id }

procedure update_addr_table(addr : type_address); primitive;
{ Update address list with argument addr (if necessary). }

{*****************************************************************************}
{      Functions/Procedures Used In Module m_ack_timer                        }
{*****************************************************************************}
function compute_delay(number_acks: integer) : type_time; primitive;
{ Compute the time required for coupled responses to be issued,
  in succession, by a number of stations equal to number_acks }


{*****************************************************************************}
{      Functions/Procedures Used In Module m_type2_ack_timer                  }
{*****************************************************************************}
function lookup_type2_timeout_value : integer; primitive;
{ Return the current value for the type 2 timeout }

{*****************************************************************************}
{     Functions/Procedures Used In Module m_station_component                 }
{*****************************************************************************}
{ Test if the DL_PDU is valid.
  A DL_PDU is invalid if it has one or more of the following:
      a. not bounded by a beginning and ending flag
      b. too short
      c. too long
      d. has an invalid address or control field
      e. has an FCS error }
function DL_PDU_valid(
   DL_PDU: type_DL_PDU): boolean; primitive;

{ Buffer message from type1SAP or type2SAP in station_message_table }
procedure buffer_message_in_station_message_table(
   var station_message_table: type_station_message_table;
   message_id: type_message_id;
   message: type_DL_PDU;
   from_component: type_from_component;
   precedence: type_precedence;
   toplogy_update_id: type_topology_update_id); primitive;

{ Get message(s), applying FEC, TDC, data scrambling (if appropriate) and 0 bit
  stuffing, concatenate them together if desired and possible }
procedure construct_and_concatenate_DL_PDU_of_highest_priority(
   var transmission_to_be_sent: type_data;
   var transmission_length: type_data_length;
   var need_coupled_ack: boolean; { TRUE if coupled ack is expected }
   var message_id: type_message_id;
   var from_component: type_from_component; { Which component the message is from }
   var no_of_expected_acks: integer;
   var station_message_table: type_station_message_table;
   FEC_TDC_scrambling: type_FEC_TDC_scrambling;
   multidwell: type_multidwell); primitive;

{ Construct a coupled ack for the DL_PDU_received }
procedure construct_coupled_ack(
   var coupled_ack: type_DL_PDU;
   src_address: type_address;
   DL_PDU_received: type_DL_PDU); primitive;

{ Construct a transmission by applying FEC, TDC and data scrambling (if required) and
  0 bit stuffing to message in station_message_table.
  Used for coupled ack}
procedure construct_transmission(
   var transmission_to_be_sent: type_data;
   var transmission_length: type_data_length;
   DL_PDU: type_DL_PDU;
   FEC_TDC_scrambling: type_FEC_TDC_scrambling;
   multidwell: type_multidwell); primitive;

{ Fill in DL_PDU_addressee_list with addressee who will appear in the next DL_PDU }
procedure decide_DL_PDU_addressee(
   var DL_PDU_addressee_list: type_DL_PDU_addressee_list;
   var all_addressee_list: type_all_addressee_list); primitive;

{ Fill in all_addressee_list with all addressee this station want to send message to }
procedure decide_all_addressee(
   var all_addressee_list: type_all_addressee_list); primitive;

{ Check to see whether desired addressee exist(s).
  There are 2 types of addressee: P=0 or P=1 }
function exist_addressee(
   all_addressee_list: type_all_addressee_list;
   P_bit: type_P_bit): boolean; primitive;

{ Test if DL_PDU_addressee_list is empty }
function DL_PDU_addressee_list_empty(
   DL_PDU_addressee_list: type_DL_PDU_addressee_list): boolean; primitive;

{ Get n_th DL_PDU in a transmission }
function get_DL_PDU_from_transmission(
   transmission: type_data;
   transmission_length: type_data_length;
   n_th: integer; { n_th DL-PDU in transmission }
   FEC_TDC_scrambling: type_FEC_TDC_scrambling;
   multidwell: type_multidwell): type_DL_PDU; primitive;

{ Get the most recent Topology Update Id sent by network layer }
function get_newest_topology_update_id: type_topology_update_id; primitive;

{ Get topology_updata_id from a transmission }
function get_topology_update_id_from_transmission(transmission: type_data):
   type_topology_update_id; primitive;

{ Init FEC_TDC_scrambling variable }
```

```
procedure init_FEC_TDC_scrambling(
   var FEC_TDC_scrambling: type_FEC_TDC_scrambling); primitive;

procedure init_link_parameter_table(
   var link_parameter_table: type_link_parameter_table); primitive;

procedure init_station_message_table(
   var station_message_table: type_station_message_table); primitive;

{ Assign an unique ID to this station }
procedure init_unique_id(var unique_id: type_unique_id); primitive;

{ Test if the PDU is a DRNRC }
function is_RNRC(PDU : type_DL_PDU) : boolean; primitive;

{ Test if the received DL_PDU is coupled ack or not }
function is_coupled_ack(DL_PDU_received: type_DL_PDU): boolean; primitive;

{ Test if the message is addressed to this station }
function is_for_me(DL_PDU_received: type_DL_PDU; my_address: type_address): boolean;
   primitive;

{ Record all timer information which is needed to start the timers once the
transmission is sent.  Timers could be Ack, P, REJ timer }
procedure record_type2_transmission_timer_information(
   var transmission_type2_timer_list: type_transmission_type2_timer_list); primitive;

{ Having retrieved no_retrieved DL_PDU from transmission, test if there are more }
function more_DL_PDU_in_transmission(
   transmission: type_data;
   transmission_length: type_data_length;
   no_retrieved: integer): boolean; primitive;

{ First determine the source address of argument PDU. Then searche
  the Station Message table (outgoing messages waiting for transmission) for
  any messages with that address as a destination AND a transmission count
  greater than 1. Any such messages have the address in question deleted
  from their destination list. If that is the only destination, the
  message is removed entirely.}
procedure purge_station_message_table(PDU : type_DL_PDU); primitive;

{ When a response is received, the source address of the response is removed }
procedure remove_addressee_from_DL_PDU_addressee_list(
   addressee: type_address;
   var DL_PDU_addressee_list: type_DL_PDU_addressee_list); primitive;

{ Test if coupled ack is required for the DL_PDU_received }
function require_coupled_ack(DL_PDU_received: type_DL_PDU): boolean; primitive;

{ Test if station_message_table is empty }
function station_message_table_empty(
   station_message_table: type_station_message_table): boolean; primitive;

{ Check the parameter Delay, Throughput, and Reliablilty. Determine from the table
  provided in MIL-STD-188-220 which type of service is desired for these parameters.
  Then check what type of services are available. Return the type of service
  to be USED given the computed desired service. }
function service_type_to_be_used(
  delay_req: type_delay;
  throughput: type_throughput;
  reliability: type_reliability) : type_service_type; primitive;


{*************************}
{   Channel  Definitions   }
{*************************}
channel NL_DL (network, datalink);
   by network:
      DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;          { * }
                       dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       precedence: type_precedence;
                       throughput: type_throughput;
                       delay_req: type_delay;
                       reliability: type_reliability;
                       data: type_data;
                       data_length: type_data_length);
   by datalink:
      DL_Unitdata_Ind (dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       data: type_data;
                       data_length: type_data_length);
      DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;            { * }
                     ack_failure: type_ack_failure;
                     address_list: type_addr_list;                   { * }
                     type2_connection_status: type_type2_connection_status
                                                 { not used in type 1 });

channel DL_PL (datalink, physical);
   by datalink:
      PL_Unitdata_Req (data: type_data;
                       data_length: type_data_length;
                       FEC_TDC_scrambling: type_FEC_TDC_scrambling;
                       multidwell: type_multidwell;
                       coupled: type_coupled { * });
   by physical:
      PL_Unitdata_Ind (data: type_data;
                       data_length: type_data_length;
                       FEC_TDC_scrambling: type_FEC_TDC_scrambling;
                       multidwell: type_multidwell);
      PL_Status_Ind (net_activity: type_net_activity;
                     transmission_status: type_transmission_status);

channel OPERATOR_DL (operator, datalink);
   by operator:
      OP_quiet_mode_Req (quiet_mode: type_quiet_mode);
      OP_reset_Req;             { * }
   by datalink:
      OP_reset_Ind;             { * }

channel OPERATOR_type1SAP (operator, type1SAP);
   by operator:
      OP_test_req(OP_test_req_id: type_id;   {*}
                  destinations: type_addr_list;
                  src_addr: type_address;
                  precedence: type_precedence;
                  acknoack: type_ack_required;
                  data: type_data;
                  data_length: type_data_length);
   by type1SAP:
      OP_test_ind (result: type_test_result;
                   src_addr: type_address;
                   data: type_data;
                   data_length: type_data_length);

channel OPERATOR_type2SAP (operator, type2SAP);
   by operator:
      OP_type2_change_busy_status_Req(busy_status: type_type2_busy_status);
```

```
channel transmission_to_DL_PDU_convertor (transmission, DL_PDU);
   by transmission:
      DL_PDU_form_PL_Unitdata_Ind(DL_PDU: type_DL_PDU;
                                  topology_update_id: type_topology_update_id);
      { Station Component takes a PL_Unitdata_Ind and deconcatenate it, destuff 0 bit,
        strip off FEC, TDC, Data Scrambling to get the DL-PDU(s) and send them through
        this channel.  The queue at the other end holds separated DL_PDU's. }

channel station_type1SAP_lower_mux (station, type1SAP);
   by station:
      station_message(message_id: type_message_id;
                      message: type_DL_PDU;
                      topology_update_id: type_topology_update_id);
   by type1SAP:
      type1SAP_message(message: type_DL_PDU;
                       precedence: type_precedence;
                       message_id: type_message_id;
                       topology_update_id: type_topology_update_id);

channel station_type2SAP_lower_mux (station, type2SAP);
   by station:
      station_message(message: type_DL_PDU;
                      topology_update_id: type_topology_update_id);
   by type2SAP:
      type2SAP_message(message: type_DL_PDU;
                       precedence: type_precedence;
                       topology_update_id: type_topology_update_id;
                       timer_list: type_type2_timer_list);

channel station_type1SAP_upper_mux(station, type1SAP);
   by station:
      DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;          { * }
                       dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       precedence: type_precedence;
                       acknoack: type_ack_required;
                       data: type_data;
                       data_length: type_data_length);
   by type1SAP:
      DL_Unitdata_Ind (dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       data: type_data;
                       data_length: type_data_length);
      DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;            { * }
                     ack_failure: type_ack_failure;
                     address_list: type_addr_list;                   { * }
                     type2_connection_status: type_type2_connection_status);

channel station_type2SAP_upper_mux(station, type2SAP);
   by station:
      DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;          { * }
                       dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       precedence: type_precedence;
                       data: type_data;
                       data_length: type_data_length);
   by type2SAP:
      DL_Unitdata_Ind (dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       data: type_data;
                       data_length: type_data_length);
      DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;            { * }
                     ack_failure: type_ack_failure;
                     address_list: type_addr_list;                   { * }
                     type2_connection_status: type_type2_connection_status);

channel station_ack_timer (station, ack_timer);
   by station:
      start_ack_timer(message_id: type_message_id; from_component: type_from_component;
                      no_expected_acks: integer);
      stop_ack_timer;
   by ack_timer:
      ack_timer_timeout;

channel station_ack2_timer (station, ack_timer);
   by station:
      start_ack_timer;

channel station_P_timer (station, P_timer);
   by station:
      start_P_timer;

channel station_REJ_timer (station, REJ_timer);
   by station:
      start_REJ_timer;

channel type1SAP_busy_timer (type1SAP, busy_timer);
   by type1SAP:
      start_busy_timer;
      stop_busy_timer;
   by busy_timer:
      busy_timer_timeout;

channel type1SAP_ack_timer (type1SAP, ack_timer);
   by type1SAP:
      stop_ack_timer;
   by ack_timer:
      ack_timer_timeout(message_id: type_message_id);

channel type2SAP_connection_component_lower_mux (type2SAP, connection_component);
   by type2SAP:
      type2SAP_message(
         message: type_DL_PDU;
         topology_update_id: type_topology_update_id);
   by connection_component:
      connection_component_message(
         message: type_DL_PDU;
         precedence: type_precedence;
         topology_update_id: type_topology_update_id;
         timers_needed: type_type2_timers_needed);
      multi_ready(
         PDU_control: type_type2_PDU_control_info;
         timers_needed: type_type2_timers_needed);
      multi_not_ready;

channel type2SAP_connection_component_upper_mux (type2SAP, connection_component);
   by type2SAP:
      unitdata_request   (DL_Unitdata_id: type_DL_Unitdata_id;       { * }
                          dest_addr_list: type_addr_list;
                          src_addr: type_address;
                          topology_update_id: type_topology_update_id;
                          precedence: type_precedence;
                          data: type_data;
                          data_length: type_data_length);
      multidata_request (DL_Unitdata_id: type_DL_Unitdata_id;        { * }
```

```
                        dest_addr_list: type_addr_list;
                        src_addr: type_address;
                        topology_update_id: type_topology_update_id;
                        precedence: type_precedence;
                        data: type_data;
                        data_length: type_data_length);
      connect_request;
      connect_response;
      disconnect_request;
      reset_request;
      reset_response;
      OP_type2_change_busy_status(busy_status: type_type2_busy_status);
   by connection_component:
      data_indication (dest_addr_list: type_addr_list;
                       src_addr: type_address;
                       topology_update_id: type_topology_update_id;
                       data: type_data;
                       data_length: type_data_length);
      connect_indication;
      disconnect_indication;
      reset_indication(reset_originator: type_type2_reset_originator);
      connect_confirm;
      reset_confirm;
      report_status(status_reported: type_type2_report_status);
      ack_failure_indication (DL_Unitdata_id: type_DL_Unitdata_id;
                              address_list: type_addr_list);

channel connection_component_busy_timer (connection_component, busy_timer);
   by connection_component:
      start_busy_timer;
      stop_busy_timer;
   by busy_timer:
      busy_timer_timeout;

channel connection_component_ack_timer (connection_component, ack_timer);
   by connection_component:
      stop_ack_timer;
      start_ack_timer;   { Connection component restart the timer directly }
   by ack_timer:
      ack_timer_timeout;

channel connection_component_P_timer (connection_component, P_timer);
   by connection_component:
      stop_P_timer;
   by P_timer:
      P_timer_timeout;

channel connection_component_REJ_timer (connection_component, REJ_timer);
   by connection_component:
      stop_REJ_timer;
   by REJ_timer:
      REJ_timer_timeout;

{************************************}
{            Network layer           }
{************************************}
module m_NL activity;
   ip ip_datalink: NL_DL (network);
end;

body b_NL for m_NL;
external;            { Network layer not specified here. }


{************************************}
{           Physical layer           }
{************************************}
module m_PL activity;
   ip ip_datalink: DL_PL (physical);
end;

body b_PL for m_PL;
external;            { Physical layer not specified here. }


{************************************}
{          Operator module           }
{************************************}
module m_OPERATOR activity;
   ip ip_datalink: OPERATOR_DL (operator);
      ip_type1SAP: OPERATOR_type1SAP (operator);
      ip_type2SAP: OPERATOR_type2SAP (operator);
end;

body b_OPERATOR for m_OPERATOR;
external;            { Operator module not specified here. }


{************************************}
{            Datalink layer          }
{************************************}
{  Station Component }
{********************}
module m_station_component activity;
   ip ip_LSAP: NL_DL (datalink);
      ip_PSAP: DL_PL (datalink);
      ip_OSAP: OPERATOR_DL (datalink);
      ip_OPERATOR_type1SAP: OPERATOR_type1SAP (type1SAP);
      ip_OPERATOR_type2SAP: OPERATOR_type2SAP (type2SAP);
      { a link connecting the OPERATOR and SAP directly }
end;

body b_station_component for m_station_component;
   ip ip_type1SAP_lower_mux: station_type1SAP_lower_mux (station);
      ip_type2SAP_lower_mux: station_type2SAP_lower_mux (station);
      ip_type1SAP_upper_mux: station_type1SAP_upper_mux (station);
      ip_type2SAP_upper_mux: station_type2SAP_upper_mux (station);
      ip_ack_timer: station_ack_timer (station);
      ip_ack2_timer: array [type_station_number_range] of station_ack2_timer (station);
      ip_P_timer  : array [type_station_number_range] of station_P_timer   (station);
      ip_REJ_timer: array [type_station_number_range] of station_REJ_timer (station);
      ip_transmission: transmission_to_DL_PDU_convertor (transmission);
      ip_DL_PDU      : transmission_to_DL_PDU_convertor (DL_PDU);


   {************************}
   {   Type1SAP Component   }
   {************************}
   module m_type1SAP_component activity;
      ip ip_station_upper_mux: station_type1SAP_upper_mux (type1SAP);
         ip_busy_timer: array [0..NS_LESS_1] of type1SAP_busy_timer (type1SAP);
         ip_ack_timer: type1SAP_ack_timer (type1SAP);
         ip_station_lower_mux: station_type1SAP_lower_mux (type1SAP);
         ip_OPERATOR: OPERATOR_type1SAP (type1SAP);
   end;
   body b_type1SAP_component for m_type1SAP_component;
        state
           ACTIVE_STATE;

        var
```

```
    ack_failure_ind : type_ack_failure;
    type1_busy_status : array[0..NS_LESS_1] of type_busy;
    busy_addresses : type_addr_list;
    connection_status : type_type2_connection_status;
    current_no_queue_entries : integer;
    id_num : integer;
    index : integer;
    non_busy_addresses : type_addr_list;
    non_responding_addresses : type_addr_list;
    number_of_addresses : integer;
    message : type_DL_PDU;
    msg_type: type_incoming_DL_PDU_type;
    reply_to_id : integer;
    request : type_queued_request;
    request_type : type_request_type;
    temp_addr : type_address;

initialize
   to ACTIVE_STATE
      var count : integer;
      begin
      for count := 0 to NS_LESS_1 do
         begin
         type1_busy_status[count] := STATION_FREE;
         current_no_queue_entries := 0;
         end;
      end;

trans
   { Unitdata Request (NOACK) received from upper layer }
   from ACTIVE_STATE to same
   when ip_station_upper_mux.DL_Unitdata_Req
   provided (acknoack = NOACK)
   name TYPE1SAP_2:
      begin
      non_busy_addresses := create_non_busy_address_list(dest_addr_list);
      busy_addresses := create_busy_address_list(dest_addr_list);
      message := create_type1SAP_message(data, non_busy_addresses,
                                         data_length, 0);
      id_num := 0;
      output ip_station_lower_mux.type1SAP_message(
         message, precedence, id_num, topology_update_id);
      request_type := DL_REQUEST;
      request := create_queued_request(busy_addresses,src_addr,precedence,
                                       acknoack,
                                       data, data_length, request_type,
                                       DL_Unitdata_id);
      queue_request(request);
      end;

   { Unitdata Request (YESACK) received from upper layer }
   from ACTIVE_STATE to same
   when ip_station_upper_mux.DL_Unitdata_Req
   provided (acknoack = YESACK)
   name TYPE1SAP_3:
      begin
      non_busy_addresses := create_non_busy_address_list(dest_addr_list);
      busy_addresses := create_busy_address_list(dest_addr_list);
      message := create_type1SAP_message(data, non_busy_addresses,
                                         data_length, 1);
      id_num := choose_message_id;
      output ip_station_lower_mux.type1SAP_message(
         message, precedence, id_num, topology_update_id);
      request_type := DL_REQUEST;
      request := create_queued_request(busy_addresses,src_addr,precedence,
                                       acknoack,
                                       data, data_length, request_type,
                                       DL_Unitdata_id);
      queue_request(request);
      add_message(id_num, message, DL_Unitdata_id);
      end;

   { Test Request (YESACK) received from Operator }
   from ACTIVE_STATE to same
   when ip_OPERATOR.OP_test_req
   provided (acknoack = YESACK)
   name TYPE1SAP_4:
      begin
      non_busy_addresses := create_non_busy_address_list(destinations);
      busy_addresses := create_busy_address_list(destinations);
      message := create_type1SAP_message(data, non_busy_addresses,
                                         data_length, 1);
      id_num := choose_message_id;
      output ip_station_lower_mux.type1SAP_message(
         message, precedence, id_num, get_topology_update_id);
      request_type := TEST_REQUEST;
      request := create_queued_request(busy_addresses,src_addr,precedence,
                                       acknoack,
                                       data, data_length, request_type,
                                       OP_test_req_id);
      queue_request(request);
      add_message(id_num, message, OP_test_req_id);
      end;

   { Test Request (NOACK) received from Operator }
   from ACTIVE_STATE to same
   when ip_OPERATOR.OP_test_req
   provided (acknoack = NOACK)
   name TYPE1SAP_5:
      begin
      non_busy_addresses := create_non_busy_address_list(destinations);
      busy_addresses := create_busy_address_list(destinations);
      message := create_type1SAP_message(data, non_busy_addresses,
                                         data_length, 0);
      id_num := 0;
      output ip_station_lower_mux.type1SAP_message(
         message, precedence, id_num, get_topology_update_id);
      request_type := TEST_REQUEST;
      request := create_queued_request(busy_addresses, src_addr, precedence,
                                       acknoack,
                                       data, data_length, request_type,
                                         OP_test_req_id);
      queue_request(request);
      end;

   { Message received from Station Component }
   from ACTIVE_STATE to same
   when ip_station_lower_mux.station_message
   name TYPE1SAP_6to14:
      begin
      msg_type := get_message_type(message);
      case msg_type of
         UIC:   begin { TYPE1SAP_6 }
                { Station Component received a UI Command packet }
                output ip_station_upper_mux.DL_Unitdata_Ind(
                   get_dest_addr_list(message),
                   get_src_addr(message),
                   topology_update_id,
                   get_data(message),
                   get_data_length(message));
```

```
            end;
      TESTR: begin { TYPE1SAP_9 }
             { Station Component received a TEST Response packet }
             output ip_OPERATOR.OP_test_ind(SUCCESS,
                                      get_src_addr(message),
                                      get_data(message),
                                      get_data_length(message));
             end;
      TESTC: begin { TYPE1SAP_10 }
             { Station Component received a TEST Command packet }
             { send back the same data you got in the TESTC DL_PDU }
             message := create_type1SAP_message(get_data(message),
                          make_addr_list(get_src_addr(message)),
                          get_data_length(message), 0);
             output ip_station_lower_mux.type1SAP_message(
                message, PREC_ROUTINE, 0, topology_update_id);
             end;
      URRC:  begin { TYPE1SAP_11 }
             { Station Component received a URR Command packet }
             index := addr_index(get_src_addr(message));
             type1_busy_status[index] := STATION_FREE;
             output ip_busy_timer[index].stop_busy_timer;
             end;
      URRR:  begin { TYPE1SAP_12 }
             { Station Component received a URR Response packet }
             reply_to_id := get_message_id_being_acked(message);
             update_ack_list(reply_to_id, get_src_addr(message));
             if all_acked(reply_to_id) then
                begin
                output ip_ack_timer.stop_ack_timer;
                remove_message(reply_to_id);
                end;
             end;
      URNRC: begin { TYPE1SAP_13 }
             { Station Component received a URNR Command packet }
             index := addr_index(get_src_addr(message));
             type1_busy_status[index] := STATION_BUSY;
             output ip_busy_timer[index].start_busy_timer; {2/3/97 Amer}
             end;
      URNRR: begin { TYPE1SAP_14 }
             { Station Component received a URNR Response packet }
             { Note this does NOT constitute acknowledgment of the
               corresponding UI PDU }
             index := addr_index(get_src_addr(message));
             type1_busy_status[index] := STATION_BUSY;
             output ip_busy_timer[index].start_busy_timer; {2/3/97 Amer}
             end;
      end; {case}
   end; {transition}

{ Ack timer for current outgoing packet (at lowest level) times out,
  but this is not the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout
provided (get_message_transmission_count(message_id) < N4)
name TYPE1SAP_15:
   begin
   increment_trans_count(message_id);
   message := create_updated_type1SAP_message(message_id);
   output ip_station_lower_mux.type1SAP_message(
      message, get_msg_precedence(message_id),
      message_id, get_topology_update_id);
   end;

{ Ack timer for current outgoing packet (at lowest level) times out,
  and this is the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout
provided (get_message_transmission_count(message_id) = N4)
name TYPE1SAP_16:
   begin
   { set ack_failure_ind and connection_status approp. first, then...}
   output ip_station_upper_mux.DL_Status_Ind(
      get_DL_Unitdata_id_from_id(message_id), ack_failure_ind,
      get_nonresponding_addresses(message_id), connection_status);
   non_responding_addresses := get_nonresponding_addresses(message_id);
   number_of_addresses := addr_list_size(non_responding_addresses);
   while number_of_addresses > 0 do
      begin
      temp_addr := get_addr_from_list(non_responding_addresses);
      type1_busy_status[addr_index(temp_addr)] := STATION_BUSY;
      output ip_busy_timer[addr_index(temp_addr)].start_busy_timer;
      number_of_addresses := number_of_addresses - 1;
      end;
   remove_message(message_id);
   end;

{ Any station which was BUSY has its busy_timer timeout }
from ACTIVE_STATE to same
any i: 0..NS_LESS_1 do
when ip_busy_timer[i].busy_timer_timeout
name TYPE1SAP_17:
   begin
   type1_busy_status[i] := STATION_FREE;
   end;

{ Process request at head of queue of requests }
from ACTIVE_STATE to same
priority HIGH
{ Estelle transition firing priority HIGH. This is an "implementor's
  decision" which assures some level of "fairness," ie, that request
  at head of queue gets priority over a failed most-recent-job. }
provided (current_no_queue_entries > 0)
name TYPE1SAP_18:
   begin
   process_queue_entry;
   end;

end; { b_type1SAP_component }

{*******************************}
{    URR/URNR Busy Timer        }
{*******************************}
module m_URR_URNR_busy_timer activity;
   ip ip_type1SAP: type1SAP_busy_timer (busy_timer);
end;
body b_URR_URNR_busy_timer for m_URR_URNR_busy_timer;

   state
      TIMER_OFF, TIMER_RUNNING, TIMER_RESTART;

   initialize to TIMER_OFF
      begin
      end;

   trans
      { Start the busy_timer }
      from TIMER_OFF to TIMER_RUNNING
      when ip_type1SAP.start_busy_timer
      name BT_1:
```

```
         begin
         end;

      { Request to stop a non_running busy_timer }
      from TIMER_OFF to same
      when ip_type1SAP.stop_busy_timer
      name BT_2:
         begin
         end;

      { Reqeust to start an already_running busy timer }
      from TIMER_RUNNING to TIMER_RESTART
      when ip_type1SAP.start_busy_timer
      name BT_3:
         begin
         { Change state to restart the timer }
         end;

      { Stop a running busy_timer }
      from TIMER_RUNNING to TIMER_OFF
      when ip_type1SAP.stop_busy_timer
      name BT_4:
         begin
         end;

      { Busy_timer timeout }
      from TIMER_RUNNING to TIMER_OFF
      delay(BUSY_TIMER_TIMEOUT_PERIOD)
      name BT_5:
         begin
         output ip_type1SAP.busy_timer_timeout;
         end;


      { Spontaneous transition to restart the timer }
      from TIMER_RESTART to TIMER_RUNNING
      name BT_6:
         begin
         end;
end; { b_URR_URNR_busy_timer }



{**************************}
{    Type 1 Ack Timer      }
{**************************}
module m_ack_timer activity;
   ip ip_type1SAP: type1SAP_ack_timer (ack_timer);
      ip_station: station_ack_timer (ack_timer);
end;
body b_ack_timer for m_ack_timer;

   state
      TIMER_OFF, TIMER_RUNNING;

   var
      delay_value          : type_time;
      message_idnum        : type_message_id;

   initialize
      to TIMER_OFF
         begin
         delay_value := 0;
         end;

      trans
         { Start ack timer }
         from TIMER_OFF to TIMER_RUNNING
         when ip_station.start_ack_timer
         name ACK_TIMER_1:
            begin
            message_idnum := message_id;
            delay_value := compute_delay(no_expected_acks);
            end;

         { Stop ack timer which is not running }
         from TIMER_OFF to same
         when ip_type1SAP.stop_ack_timer
         name ACK_TIMER_2:
            begin
            { Ignore }
            end;

         { Stop ack timer which is running }
         from TIMER_RUNNING to TIMER_OFF
         when ip_type1SAP.stop_ack_timer
         name ACK_TIMER_3:
            begin
            end;

         { Start ack timer which is already running }
         from TIMER_RUNNING to same
         when ip_station.start_ack_timer
         name ACK_TIMER_4:
            begin
            { Ignore - allows timer to continue; do not restart }
            end;

         { Timer timeout }
         from TIMER_RUNNING to TIMER_OFF
         delay(delay_value)
         name ACK_TIMER_5:
            begin
            output ip_type1SAP.ack_timer_timeout(message_idnum);
            end;

   end; { b_ack_timer }

   {************************}
   {   Type2SAP Component   }
   {************************}
   module m_type2SAP_component activity;
      ip ip_station_lower_mux: station_type2SAP_lower_mux (type2SAP);
         ip_station_upper_mux: station_type2SAP_upper_mux (type2SAP);
         ip_OPERATOR: OPERATOR_type2SAP (type2SAP);
         ip_station_ack2_timer:
            array [type_station_number_range] of station_ack2_timer (ack_timer);
         ip_station_P_timer  :
            array [type_station_number_range] of station_P_timer   (P_timer);
         ip_station_REJ_timer:
            array [type_station_number_range] of station_REJ_timer (REJ_timer);
   end;

   body b_type2SAP_component for m_type2SAP_component;
      ip ip_connection_component_lower_mux: array [type_station_number_range] of
            type2SAP_connection_component_lower_mux (type2SAP);
         ip_connection_component_upper_mux: array [type_station_number_range] of
            type2SAP_connection_component_upper_mux (type2SAP);

      {******************************}
```

```
{      Type 2 Busy Timer        }
{*****************************}
module m_type2_busy_timer activity;
   ip ip_connection_component: connection_component_busy_timer (busy_timer);
end;
body b_type2_busy_timer for m_type2_busy_timer;

   state
      TIMER_OFF, TIMER_RUNNING, TIMER_RESTART;

   initialize to TIMER_OFF
      begin
      end;

   trans
      { Start the busy timer }
      from TIMER_OFF to TIMER_RUNNING
      when ip_connection_component.start_busy_timer
      name BT_1:
         begin
         end;

      { Request to stop a non-running busy timer }
      from TIMER_OFF to same
      when ip_connection_component.stop_busy_timer
      name BT_2:
         begin
         { Ignore }
         end;

      { Reqeust to start an already running busy timer }
      from TIMER_RUNNING to TIMER_RESTART
      when ip_connection_component.start_busy_timer
      name BT_3:
         begin
         { Change state to restart the timer }
         end;

      { Stop a running busy timer }
      from TIMER_RUNNING to TIMER_OFF
      when ip_connection_component.stop_busy_timer
      name BT_4:
         begin
         end;

      { Busy timer timeout }
      from TIMER_RUNNING to TIMER_OFF
      delay(BUSY_TIMER_TIMEOUT_PERIOD)
      name BT_5:
         begin
         output ip_connection_component.busy_timer_timeout;
         end;

      { Spontaneous transition to restart the timer }
      from TIMER_RESTART to TIMER_RUNNING
      name BT_6:
         begin
         end;
end; { b_type2_busy_timer }

{*************************}
{    Type 2 Ack Timer      }
{*************************}
module m_type2_ack_timer activity;
   ip ip_connection_component: connection_component_ack_timer (ack_timer);
      ip_station: station_ack2_timer (ack_timer);
end;
body b_type2_ack_timer for m_type2_ack_timer;

   state
      TIMER_OFF, TIMER_RUNNING;

   var
      delay_value: type_time;

   initialize
      to TIMER_OFF
         begin
         delay_value := lookup_type2_timeout_value;
         end;

   trans
      { Start timer }
      from TIMER_OFF to TIMER_RUNNING
      when ip_station.start_ack_timer
      name ACK_TIMER_1:
         begin
         end;

      { Stop timer which is not running }
      from TIMER_OFF to same
      when ip_connection_component.stop_ack_timer
      name ACK_TIMER_2:
         begin
         { Ignore }
         end;

      { Stop timer which is running }
      from TIMER_RUNNING to TIMER_OFF
      when ip_connection_component.stop_ack_timer
      name ACK_TIMER_3:
         begin
         end;

      { Start timer which is already running }
      from TIMER_RUNNING to same
      when ip_station.start_ack_timer
      name ACK_TIMER_4:
         begin
         { Starting an ack timer that is already running has no effect.
           That is, the timer continues, it is not restarted. }
         end;

      { Timer timeout }
      from TIMER_RUNNING to TIMER_OFF
      delay(delay_value)
      name ACK_TIMER_5:
         begin
         output ip_connection_component.ack_timer_timeout;
         end;
end; { b_type2_ack_timer }

{*************************}
{    Type 2 P-bit Timer    }
{*************************}
module m_type2_P_timer activity;
   ip ip_connection_component: connection_component_P_timer (P_timer);
      ip_station: station_P_timer (P_timer);
end;
body b_type2_P_timer for m_type2_P_timer;
```

```
   state
      TIMER_OFF, TIMER_RUNNING;

   var
      delay_value: type_time;

   initialize
      to TIMER_OFF
         begin
         delay_value := lookup_type2_timeout_value;
         end;

   trans
      { Start timer }
      from TIMER_OFF to TIMER_RUNNING
      when ip_station.start_P_timer
      name P_TIMER_1:
         begin
         end;

      { Stop timer which is not running }
      from TIMER_OFF to same
      when ip_connection_component.stop_P_timer
      name P_TIMER_2:
         begin
         { Ignore }
         end;

      { Stop timer which is running }
      from TIMER_RUNNING to TIMER_OFF
      when ip_connection_component.stop_P_timer
      name P_TIMER_3:
         begin
         end;

      { Start timer which is already running }
      from TIMER_RUNNING to same
      when ip_station.start_P_timer
      name P_TIMER_4:
         begin
         { Starting an P timer that is already running has no effect.
           That is, the timer continues, it is not restarted. }
         end;

      { Timer timeout }
      from TIMER_RUNNING to TIMER_OFF
      delay(delay_value)
      name P_TIMER_5:
         begin
         output ip_connection_component.P_timer_timeout;
         end;
end; { b_type2_P_timer }

{*************************}
{    Type 2 REJ Timer     }
{*************************}
module m_type2_REJ_timer activity;
   ip ip_connection_component: connection_component_REJ_timer (REJ_timer);
      ip_station: station_REJ_timer (REJ_timer);
end;
body b_type2_REJ_timer for m_type2_REJ_timer;

   state
      TIMER_OFF, TIMER_RUNNING;

   var
      delay_value: type_time;

   initialize
      to TIMER_OFF
         begin
         delay_value := lookup_type2_timeout_value;
         end;

   trans
      { Start timer }
      from TIMER_OFF to TIMER_RUNNING
      when ip_station.start_REJ_timer
      name REJ_TIMER_1:
         begin
         end;

      { Stop timer which is not running }
      from TIMER_OFF to same
      when ip_connection_component.stop_REJ_timer
      name REJ_TIMER_2:
         begin
         { Ignore }
         end;

      { Stop timer which is running }
      from TIMER_RUNNING to TIMER_OFF
      when ip_connection_component.stop_REJ_timer
      name REJ_TIMER_3:
         begin
         end;

      { Start timer which is already running }
      from TIMER_RUNNING to same
      when ip_station.start_REJ_timer
      name REJ_TIMER_4:
         begin
         { Starting an REJ timer that is already running has no effect.
           That is, the timer continues, it is not restarted. }
         end;

      { Timer timeout }
      from TIMER_RUNNING to TIMER_OFF
      delay(delay_value)
      name REJ_TIMER_5:
         begin
         output ip_connection_component.REJ_timer_timeout;
         end;
end; { b_type2_REJ_timer }

{*******************************}
{   Type2 Connection Component   }
{*******************************}
module m_type2_connection_component activity;
   ip ip_type2SAP_lower_mux:
         type2SAP_connection_component_lower_mux (connection_component);
      ip_type2SAP_upper_mux:
         type2SAP_connection_component_upper_mux (connection_component);
      ip_busy_timer: connection_component_busy_timer (connection_component);
      ip_ack2_timer:  connection_component_ack_timer  (connection_component);
      ip_P_timer:    connection_component_P_timer    (connection_component);
      ip_REJ_timer:  connection_component_REJ_timer  (connection_component);
end;
```

```
      body b_type2_connection_component for m_type2_connection_component;
         state
            ADM, SETUP, NORMAL, BUSY, REJECT, AWAIT, AWAIT_BUSY, AWAIT_REJECT, D_CONN,
            RESET, ERROR, CONN, RESET_CHECK, RESET_WAIT, RSET, RSET_EXPECTED;
         stateset
            MULTI_NOT_READY_STATES = [ ADM, SETUP, D_CONN, RESET, ERROR, CONN,
                                       RESET_CHECK, RESET_WAIT, RSET, RSET_EXPECTED ];
            NORMAL_OR_BUSY_OR_REJECT = [ NORMAL, BUSY, REJECT ];
            AWAIT_STATES = [ AWAIT, AWAIT_BUSY, AWAIT_REJECT ];
            ALL_TRANSFER_STATES = [ NORMAL, BUSY, REJECT, AWAIT, AWAIT_BUSY,
                                    AWAIT_REJECT, RSET, RSET_EXPECTED ];
            ISO_TRANSFER_STATES =
               [ NORMAL, BUSY, REJECT, AWAIT, AWAIT_BUSY, AWAIT_REJECT ];
            ISO_TRANSFER_STATES_EXCEPT_NORMAL =
               [ BUSY, REJECT, AWAIT, AWAIT_BUSY, AWAIT_REJECT ];
            NORMAL_OR_BUSY_OR_AWAIT_OR_AWAIT_BUSY = [ NORMAL, BUSY, AWAIT, AWAIT_BUSY ];
         var
            DATA_flag: 0..2;
            F_flag   : 0..1;
            P_flag   : 0..1;
            S_flag   : 0..1;
            local_busy : 0..1;
            remote_busy: 0..1;
            N_R: type_type2_sequence_number;
            V_S: type_type2_sequence_number;
            V_R: type_type2_sequence_number;
            retry_count: integer;
            v_out_of_order_data_buffer: type_type2_out_of_order_data_buffer;
            v_retransmission_data_buffer: type_type2_retransmission_data_buffer;
            v_unitdata_buffer: type_type2_unitdata_buffer;
         initialize to ADM
            begin
            init_out_of_order_data_buffer(v_out_of_order_data_buffer);
            init_retransmission_data_buffer(v_retransmission_data_buffer);
            init_unitdata_buffer(v_unitdata_buffer);
            end;

         trans

{ ***************Original ISO 8802 Table 4 with modifications *************** }

{ Receive CONNECT_REQUEST }
from ADM to SETUP
when ip_type2SAP_upper_mux.connect_request
name ISO_1_OPTION_1:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := 0;
   S_flag := 0;
   end;

{ Receive CONNECT_REQUEST }
from ADM to same
when ip_type2SAP_upper_mux.connect_request
name ISO_1_OPTION_2:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive SABME_CMD(P=X) }
from ADM to CONN
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_2:
   begin
   output ip_type2SAP_upper_mux.connect_indication;
   P_flag := get_PF_bit(message);
   end;

{ Receive DISC_CMD(P=X) }
from ADM to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_3:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive XXX_CMD(P=1) }
from ADM to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_XXX_CMD(message, ADM_state) = TRUE) and (get_PF_bit(message) = 1)
name ISO_4:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive XXX_CMD(P=0) or XXX_RSP(F=X) }
from ADM to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_XXX_CMD(message, ADM_state) = TRUE) and (get_PF_bit(message) = 0)
      or (is_XXX_RSP(message, ADM_state) = TRUE)
name ISO_5:
   begin
   end;

{ Receive CONNECT_RESPONSE }
from CONN to NORMAL
when ip_type2SAP_upper_mux.connect_response
name ISO_6:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   P_flag := 0;
   remote_busy := 0;
   end;

{ Receive DISCONNECT_REQUEST }
from CONN to ADM
when ip_type2SAP_upper_mux.disconnect_request
name ISO_7:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive SABME_CMD(P=X) }
from CONN to same
```

```
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_8:
   begin
   F_flag := get_PF_bit(message);
   end;

{ Receive DM_RSP(F=X) }
from CONN to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_9:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive XXX_YYY }
from CONN to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, CONN_state) = TRUE
name ISO_10:
   begin
   end;

{ Receive RESET_REQUEST and S_FLAG=0 }
from RESET_WAIT to RESET
when ip_type2SAP_upper_mux.reset_request
provided S_flag = 0
name ISO_11:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := 0;
   end;

{ Receive RESET_REQUEST and S_FLAG=1 }
from RESET_WAIT to NORMAL
when ip_type2SAP_upper_mux.reset_request
provided S_flag = 1
name ISO_12:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   P_flag := 0;
   remote_busy := 0;
   output ip_type2SAP_upper_mux.reset_confirm;
   end;

{ Receive DISCONNECT_REQUEST and S_FLAG=0 }
from RESET_WAIT to D_CONN
when ip_type2SAP_upper_mux.disconnect_request
provided S_flag = 0
name ISO_13:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DISC_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := 0;
   end;

{ Receive DISCONNECT_REQUEST and S_FLAG=1 }
from RESET_WAIT to ADM
when ip_type2SAP_upper_mux.disconnect_request
provided S_flag = 1
name ISO_14:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive DM_RSP(F=X) }
from RESET_WAIT to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_15:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive SABME_CMD(P=X) }
from RESET_WAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_16:
   begin
   S_flag := 1;
   F_flag := get_PF_bit(message);
   end;

{ Receive DISC_CMD(P=X) }
from RESET_WAIT to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_17:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive XXX_YYY }
from RESET_WAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, RESET_WAIT_state) = TRUE
name ISO_18:
   begin
   end;

{ Receive RESET_RESPONSE }
from RESET_CHECK to NORMAL
when ip_type2SAP_upper_mux.reset_response
name ISO_19:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   P_flag := 0;
   remote_busy := 0;
```

```
   end;

{ Receive DISCONNECT_REQUEST }
from RESET_CHECK to ADM
when ip_type2SAP_upper_mux.disconnect_request
name ISO_20:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(F_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive DM_RSP(F=X) }
from RESET_CHECK to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_21:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive SABME_CMD(P=X) }
from RESET_CHECK to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_22:
   begin
   F_flag := get_PF_bit(message);
   end;

{ Receive DISC_CMD(P=X) }
from RESET_CHECK to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_23:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;

{ Receive XXX_YYY }
from RESET_CHECK to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, RESET_CHECK_state) = TRUE
name ISO_24:
   begin
   end;

{ Receive SABME_CMD(P=X) }
from SETUP to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_25:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   S_flag := 1;
   end;

{ Receive UA_RSP(F=X) and P_FLAG=F }
from SETUP to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = UA_RSP)
     and (P_flag = get_PF_bit(message))
name ISO_26:
   begin
   output ip_ack2_timer.stop_ack_timer;
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_upper_mux.connect_confirm;
   remote_busy := 0;
   end;

{ Ack timer expired and S_FLAG=1 }
from SETUP to NORMAL
when ip_ack2_timer.ack_timer_timeout
provided S_flag = 1
name ISO_27:
   begin
   P_flag := 0;
   output ip_type2SAP_upper_mux.connect_confirm;
   remote_busy := 0;
   end;

{ Receive DISC_CMD(P=X) }
from SETUP to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_28:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive DM_RSP(F=X) }
from SETUP to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_29:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive XXX_YYY }
from SETUP to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, SETUP_state) = TRUE
name ISO_30:
   begin
   end;

{ Ack timer expired and RETRY_COUNT<N2 and S_FLAG=0 }
from SETUP to same
when ip_ack2_timer.ack_timer_timeout
```

```
provided (retry_count < N2) and (S_flag = 0)
name ISO_31:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and RETRY_COUNT>=N2 and S_FLAG=0 }
from SETUP to ADM
when ip_ack2_timer.ack_timer_timeout
provided (retry_count >= N2) and (S_flag = 0)
name ISO_32:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;


{ Receive SABME_CMD(P=X) }
from RESET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_33:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   S_flag := 1;
   end;

{ Receive UA_RSP(F=X) and P_FLAG=F }
from RESET to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = UA_RSP)
     and (P_flag = get_PF_bit(message))
name ISO_34:
   begin
   output ip_ack2_timer.stop_ack_timer;
   V_S := 0;
   V_R := 0;
   retry_count := 0;
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_upper_mux.reset_confirm;
   remote_busy := 0;
   end;

{ Ack timer expired and S_FLAG=1 }
from RESET to NORMAL
when ip_ack2_timer.ack_timer_timeout
provided S_flag = 1
name ISO_35:
   begin
   P_flag := 0;
   output ip_type2SAP_upper_mux.reset_confirm;
   remote_busy := 0;
   end;


{ Receive DISC_CMD(P=X) }
from RESET to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_36:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive DM_RSP(F=X) }
from RESET to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_37:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive XXX_YYY }
from RESET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, RESET_state) = TRUE
name ISO_38:
   begin
   end;

{ Ack timer expired and RETRY_COUNT<N2 and S_FLAG=0 }
from RESET to same
when ip_ack2_timer.ack_timer_timeout
provided (retry_count < N2) and (S_flag = 0)
name ISO_39:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and RETRY_COUNT>=N2 and S_FLAG=0 }
from RESET to ADM
when ip_ack2_timer.ack_timer_timeout
provided (retry_count >= N2) and (S_flag = 0)
name ISO_40:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   end;


{ Receive SABME_CMD(P=X) }
from D_CONN to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_41:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DM_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_ack2_timer.stop_ack_timer;
```

```
   end;

{ Receive UA_RSP(F=X) and P_FLAG=F }
from D_CONN to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = UA_RSP)
     and (P_flag = get_PF_bit(message))
name ISO_42:
   begin
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive DISC_CMD(P=X) }
from D_CONN to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_43:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive DM_RSP(F=X) }
from D_CONN to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_44:
   begin
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive XXX_YYY }
from D_CONN to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_YYY(message, D_CONN_state) = TRUE
name ISO_45:
   begin
   end;

{ Ack timer expired and RETRY_COUNT<N2 }
from D_CONN to same
when ip_ack2_timer.ack_timer_timeout
provided retry_count < N2
name ISO_46:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DISC_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and RETRY_COUNT>=N2 }
from D_CONN to ADM
when ip_ack2_timer.ack_timer_timeout
provided retry_count >= N2
name ISO_47:
   begin
   end;


{ Receive SABME_CMD(P=X) }
from ERROR to RESET_CHECK
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_48:
   begin
   output ip_type2SAP_upper_mux.reset_indication(REMOTE);
   output ip_ack2_timer.stop_ack_timer;
   F_flag := get_PF_bit(message);
   end;

{ Receive DISC_CMD(P=X) }
from ERROR to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name ISO_49:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive DM_RSP(F=X) }
from ERROR to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_50:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_ack2_timer.stop_ack_timer;
   end;

{ Receive FRMR_RSP(F=X) }
from ERROR to RESET_WAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = FRMR_RSP
name ISO_51:
   begin
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   output ip_ack2_timer.stop_ack_timer;
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   S_flag := 0;
   end;

{ Receive XXX_CMD(P=X) }
from ERROR to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_CMD(message, ERROR_state) = TRUE
name ISO_52:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ Receive XXX_RSP(F=X) }
from ERROR to same
when ip_type2SAP_lower_mux.type2SAP_message
provided is_XXX_RSP(message, ERROR_state) = TRUE
name ISO_53:
   begin
   end;

{ Ack timer expired and RETRY_COUNT<N2 }
from ERROR to same
when ip_ack2_timer.ack_timer_timeout
```

```
provided retry_count < N2
name ISO_54:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and RETRY_COUNT>=N2 }
from ERROR to RESET_WAIT
when ip_ack2_timer.ack_timer_timeout
provided retry_count >= N2
name ISO_55:
   begin
   S_flag := 0;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   end;


{ Receive DISCONNECT_REQUEST }
from ISO_TRANSFER_STATES to D_CONN
when ip_type2SAP_upper_mux.disconnect_request
name ISO_56:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DISC_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive RESET_REQUEST }
from ISO_TRANSFER_STATES to RESET
when ip_type2SAP_upper_mux.reset_request
name ISO_57:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   S_flag := 0;
   end;

{ Receive SABME_CMD(P=X) }
from ISO_TRANSFER_STATES to RESET_CHECK
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name ISO_58:
   begin
   output ip_type2SAP_upper_mux.reset_indication(REMOTE);
   F_flag := get_PF_bit(message);
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive DISC_CMD(P=X) }
from ISO_TRANSFER_STATES to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name RSET_59:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive DM_RSP(F=X) }
from ISO_TRANSFER_STATES to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name ISO_60:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive BAD_PDU }
from ISO_TRANSFER_STATES to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided bad_PDU(message) = TRUE
name ISO_61:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive UA_RSP(F=X) }
from ISO_TRANSFER_STATES to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = UA_RSP
name ISO_62_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive UA_RSP }
from ISO_TRANSFER_STATES to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = UA_RSP
```

```
name ISO_62_OPTION_2:
   begin
   end;

{ Receive RSET_CMD(P=X) }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = RSET_CMD
name ISO_63A:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive RSET_CMD(P=X) }
from AWAIT_REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = RSET_CMD
name ISO_63B:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive RSET_CMD(P=X) }
from NORMAL_OR_BUSY_OR_AWAIT_OR_AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = RSET_CMD
name ISO_63C:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive XXX_RSP(F=1) and P_FLAG=0 }
from ISO_TRANSFER_STATES to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_XXX_RSP(message, ISO_TRANSFER_STATES_states) = TRUE)
     and (get_PF_bit(message) = 1) and (P_flag = 0)
name ISO_64_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive XXX_RSP(F=1) and P_FLAG=0 }
from ISO_TRANSFER_STATES to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_XXX_RSP(message, ISO_TRANSFER_STATES_states) = TRUE)
     and (get_PF_bit(message) = 1) and (P_flag = 0)
name ISO_64_OPTION_2:
   begin
   end;

{ P timer expired and RETRY_COUNT>=N2 }
from ISO_TRANSFER_STATES to RESET_WAIT
when ip_P_timer.P_timer_timeout
provided retry_count >= N2
name ISO_65A:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{ Ack timer expired and RETRY_COUNT>=N2 }
from ISO_TRANSFER_STATES to RESET_WAIT
when ip_ack2_timer.ack_timer_timeout
provided retry_count >= N2
name ISO_65B:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{ REJ timer expired and RETRY_COUNT>=N2 }
from ISO_TRANSFER_STATES to RESET_WAIT
when ip_REJ_timer.REJ_timer_timeout
provided retry_count >= N2
name ISO_65C:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{ Busy timer expired and RETRY_COUNT>=N2 }
from ISO_TRANSFER_STATES to RESET_WAIT
when ip_busy_timer.busy_timer_timeout
provided retry_count >= N2
name ISO_65D:
   begin
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;


{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from NORMAL to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_66_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 1, 0));
   end;
```

```
{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from NORMAL to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_66_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=1 }
from NORMAL to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 1)
name ISO_67:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=0 }
from NORMAL to BUSY
provided local_busy_detected and (P_flag = 0)
name ISO_68_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   DATA_flag := 0;
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=0 }
from NORMAL to BUSY
provided local_busy_detected and (P_flag = 0)
name ISO_68_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 0;
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=1 }
from NORMAL to BUSY
provided local_busy_detected and (P_flag = 1)
name ISO_69:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 or
   Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1
  ) and IS_SREJ_USED=FALSE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_flag = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                                and (P_flag = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
                                                and (P_flag = 1))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_70_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 or
   Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1
  ) and IS_SREJ_USED=FALSE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_flag = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                                and (P_flag = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
                                                and (P_flag = 1))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_70_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=1 or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=1)
   and IS_SREJ_USED=FALSE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_flag = 1)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                                and (P_flag = 1))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
```

```
name ISO_71:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_72:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                               and (P_FLAG = 0)
name ISO_73_OPTION_1:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_CMD(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_CMD(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                               and (P_FLAG = 0)
name ISO_73_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=0) and P_FLAG=1 or
  Receive I_CMD(P=0) and P_FLAG=1 }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                               and (P_FLAG = 1)
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                               and (P_FLAG = 1)
name ISO_74:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_CMD(P=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_75:
   begin
```

```
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or (RR_RSP(F=1) and P_FLAG=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
                                                 and (P_flag = 1))
name ISO_76:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1)
name ISO_77:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) or (RNR_RSP(F=1) and P_FLAG=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
                                                  and (P_flag = 1))
name ISO_78:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name ISO_79:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_80_OPTION_1:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
```

```
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_80_OPTION_2:
   begin
   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;


   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=1 or
  Receive REJ_RSP(F=0) and P_FLAG=1 }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
name ISO_81:
   begin
   V_S := get_N_R(message);


   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;


   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
     and (get_PF_bit(message) = 1)
name ISO_82:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { RESEND_I_RSP(F=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_RSP(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ INITIATE_P/F_CYCLE and P_FLAG=0}
from NORMAL to same
provided initiate_P_F_cycle and (P_flag = 0)
name ISO_83:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0))
   end;

{ P timer expired and RETRY_COUNT<N2 }
from NORMAL to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_84_OPTION_1:
```

```
   begin
   P_flag := 0;
   end;

{ P timer expired and RETRY_COUNT<N2 }
from NORMAL to AWAIT
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_84_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from NORMAL to AWAIT
when ip_ack2_timer.ack_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_85A:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ Busy timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from NORMAL to AWAIT
when ip_busy_timer.busy_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_85B:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;


{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from BUSY to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_86_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 1, 0));
   end;

{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from BUSY to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_86_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=1 }
from BUSY to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 1)
name ISO_87:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ Local busy cleared and DATA_FLAG=1 and P_FLAG=0 and IS_SREJ_USED=FALSE }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 0) and (is_SREJ_used = FALSE)
name ISO_88_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   end;

{ Local busy cleared and DATA_FLAG=1 and P_FLAG=0 and IS_SREJ_USED=FALSE }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 0) and (is_SREJ_used = FALSE)
name ISO_88_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Local busy cleared and DATA_FLAG=1 and P_FLAG=1 and IS_SREJ_USED=FALSE }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 1) and (is_SREJ_used = FALSE)
name ISO_89:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Local busy cleared and DATA_FLAG=0 and P_FLAG=0 }
from BUSY to NORMAL
provided local_busy_cleared and (DATA_flag = 0) and (P_flag = 0)
name ISO_90_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   end;

{ Local busy cleared and DATA_FLAG=0 and P_FLAG=0 }
from BUSY to NORMAL
provided local_busy_cleared and (DATA_flag = 0) and (P_flag = 0)
name ISO_90_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Local busy cleared and DATA_FLAG=0 and P_FLAG=1 }
from BUSY to NORMAL
provided local_busy_cleared and (DATA_flag = 0) and (P_flag = 1)
```

```
name ISO_91:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Local busy cleared and DATA_FLAG=2 and P_FLAG=0 }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 2) and (P_flag = 0)
name ISO_92_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   end;

{ Local busy cleared and DATA_FLAG=2 and P_FLAG=0 }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 2) and (P_flag = 0)
name ISO_92_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Local busy cleared and DATA_FLAG=2 and P_FLAG=1 }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 2) and (P_flag = 1)
name ISO_93:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive I_RSP(F=X)_WITH_UNEXPECTED_N(S) and P_FLAG=F or
  Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and with_unexpected_N_S(message) and (P_flag = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
name ISO_94_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 0 then
      DATA_flag := 1;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=X)_WITH_UNEXPECTED_N(S) and P_FLAG=F or
  Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and with_unexpected_N_S(message) and (P_flag = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
name ISO_94_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 0 then
      DATA_flag := 1;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=1 or
  Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=1 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 1)
      or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 1)
name ISO_95:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 0 then
      DATA_flag := 1;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message)
name ISO_96:
   begin
```

```
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 0 then
      DATA_flag := 1;
   end;

{ Receive I_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_97_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 1;
   end;

{ Receive I_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_97_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 0;
   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 0)
name ISO_98_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 1;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 0)
name ISO_98_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 1;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from BUSY to same
```

```
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 0)
name ISO_98_OPTION_3:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 0;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=X) and P_FLAG=F or
  Receive I_CMD(P=0) and P_FLAG=0 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (P_FLAG = get_PF_bit(message))
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 0)
name ISO_98_OPTION_4:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 0;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_RSP(F=0) and P_FLAG=1 or
  Receive I_CMD(P=0) and P_FLAG=1 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
name ISO_99_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 1;
   end;

{ Receive I_RSP(F=0) and P_FLAG=1 or
  Receive I_CMD(P=0) and P_FLAG=1 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
name ISO_99_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if DATA_flag = 2 then
      output ip_REJ_timer.stop_REJ_timer;
   DATA_flag := 0;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or (RR_RSP(F=1) and P_FLAG=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
```

```
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
                                                 and (P_flag = 1))
name ISO_100:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1)
name ISO_101:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) or (RNR_RSP(F=1) and P_FLAG=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
                                                  and (P_flag = 1))
name ISO_102:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name ISO_103:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_104_OPTION_1:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
```

```
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_104_OPTION_2:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=1 or
  Receive REJ_RSP(F=0) and P_FLAG=1 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
name ISO_105:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;


   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
     and (get_PF_bit(message) = 1)
name ISO_106:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ INITIATE_P/F_CYCLE and P_FLAG=0 }
from BUSY to same
provided initiate_P_F_cycle and (P_flag = 0)
name ISO_107:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0))
   end;

{ P timer expired and RETRY_COUNT<N2 }
from BUSY to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_108_OPTION_1:
   begin
   P_flag := 0;
   end;

{ P timer expired and RETRY_COUNT<N2 }
from BUSY to AWAIT_BUSY
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_108_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from BUSY to AWAIT_BUSY
when ip_ack2_timer.ack_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_109A:
   begin
```

```
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ Busy timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from BUSY to AWAIT_BUSY
when ip_busy_timer.busy_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_109B:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ REJ timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from BUSY to same
when ip_REJ_timer.REJ_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_110_OPTION_1:
   begin
   DATA_flag := 1;
   end;

{ REJ timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from BUSY to AWAIT_BUSY
when ip_REJ_timer.REJ_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_110_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   DATA_flag := 1;
   end;

{ REJ timer expired and P_FLAG=1 and RETRY_COUNT<N2 }
from BUSY to same
when ip_REJ_timer.REJ_timer_timeout
provided (P_flag = 1) and (retry_count < N2)
name ISO_111:
   begin
   DATA_flag := 1;
   end;


{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from REJECT to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_112_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 1, 0));
   end;

{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=0 }
from REJECT to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 0)
name ISO_112_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ Receive DATA_REQUEST and REMOTE_BUSY=0 and P_FLAG=1 }
from REJECT to same
provided exist_buffered_data_request(v_unitdata_buffer)
     and (remote_busy = 0) and (P_flag = 1)
name ISO_113:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_1_I_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=0 }
from REJECT to BUSY
provided local_busy_detected and (P_flag = 0)
name ISO_114_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   DATA_flag := 2;
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=0 }
from REJECT to BUSY
provided local_busy_detected and (P_flag = 0)
name ISO_114_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 2;
   end;

{ LOCAL_BUSY_DETECTED and P_FLAG=1 }
from REJECT to BUSY
provided local_busy_detected and (P_flag = 1)
name ISO_115:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 2;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1
  ) and IS_SREJ_USED=FALSE }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
                                                and (P_flag = 1))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_116:
   begin
```

```
   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_117:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ ( Receive I_RSP(F=X) and P_FLAG=F or Receive I_CMD(P=0) and P_FLAG=0 )
  and IS_SREJ_USED=FALSE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 0))
      and (is_SREJ_used = FALSE)
name ISO_118_OPTION_1:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_CMD(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_CMD(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ ( Receive I_RSP(F=X) and P_FLAG=F or Receive I_CMD(P=0) and P_FLAG=0 )
  and IS_SREJ_USED=FALSE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 0))
      and (is_SREJ_used = FALSE)
name ISO_118_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ ( Receive I_RSP(F=0) and P_FLAG=1 or Receive I_CMD(P=0) and P_FLAG=1 )
  and IS_SREJ_USED=FALSE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 1)
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 1))
      and (is_SREJ_used = FALSE)
name ISO_119:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_REJ_timer.stop_REJ_timer;
   end;
```

```
{ Receive I_CMD(P=1) }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1) and (is_SREJ_used = FALSE)
name ISO_120:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or (RR_RSP(F=1) and P_FLAG=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
                                                 and (P_flag = 1))
name ISO_121:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1)
name ISO_122:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) or (RNR_RSP(F=1) and P_FLAG=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
                                                  and (P_flag = 1))
name ISO_123:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name ISO_124:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_125_OPTION_1:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
```

```
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=0 or
  Receive REJ_RSP(F=X) and P_FLAG=F }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = REJ_RSP)
          and (P_flag = get_PF_bit(message))
name ISO_125_OPTION_2:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=0) and P_FLAG=1 or
  Receive REJ_RSP(F=0) and P_FLAG=1 }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0) and (P_FLAG = 1)
name ISO_126:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;


   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive REJ_CMD(P=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_CMD)
     and (get_PF_bit(message) = 1)
name ISO_127:
   begin
   V_S := get_N_R(message);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { RESEND_I_RSP(F=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_RSP(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ INITIATE_P/F_CYCLE and P_FLAG=0}
from REJECT to same
provided initiate_P_F_cycle and (P_flag = 0)
name ISO_128:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0))
   end;

{ REJ timer expired and RETRY_COUNT<N2 and P_FLAG=0 and IS_SREJ_USED=FALSE }
from REJECT to same
when ip_REJ_timer.REJ_timer_timeout
provided (retry_count < N2) and (P_flag = 0) and (is_SREJ_used = FALSE)
name ISO_129_OPTION_1:
   begin
```

```
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count+1;
   end;

{ REJ timer expired and RETRY_COUNT<N2 and P_FLAG=0 and IS_SREJ_USED=FALSE }
from REJECT to NORMAL
when ip_REJ_timer.REJ_timer_timeout
provided (retry_count < N2) and (P_flag = 0) and (is_SREJ_used = FALSE)
name ISO_129_OPTION_2:
   begin
   end;

{ P timer expired and RETRY_COUNT<N2 }
from REJECT to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_130_OPTION_1:
   begin
   P_flag := 0;
   end;

{ P timer expired and RETRY_COUNT<N2 }
from REJECT to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_130_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count + 1;
   end;

{ Ack timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from REJECT to AWAIT_REJECT
when ip_ack2_timer.ack_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_131A:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count + 1;
   end;

{ Busy timer expired and P_FLAG=0 and RETRY_COUNT<N2 }
from REJECT to AWAIT_REJECT
when ip_busy_timer.busy_timer_timeout
provided (P_flag = 0) and (retry_count < N2)
name ISO_131B:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count + 1;
   end;

{ LOCAL_BUSY_DETECTED }
from AWAIT to AWAIT_BUSY
provided local_busy_detected
name ISO_132:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 0;
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from AWAIT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_133_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from AWAIT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_133_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;
```

```
   remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) or
  ) and IS_SREJ_USED=FALSE }
from AWAIT to AWAIT_REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_134:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from AWAIT to AWAIT_REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_135:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
     and (get_PF_bit(message) = 1)
name ISO_136_OPTION_1:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_CMD(P=1) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0));
      end
   else if resend_I_PDU_finished = FALSE then
      begin         { Resend the first unacked I_PDU as I_CMD(P=1) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the rest unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive I_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
     and (get_PF_bit(message) = 1)
name ISO_136_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_P_timer.stop_P_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_XXX(X=0)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_XXX(X=0) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_XXX(0), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 0, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive I_RSP(F=0) or Receive I_CMD(P=0) }
```

```
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0)
name ISO_137:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive I_CMD(P=1) }
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_138:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1))
name ISO_139_OPTION_1:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1))
name ISO_139_OPTION_2:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or REJ_CMD(P=0) or REJ_RSP(F=0) }
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 0))
name ISO_140:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) or REJ_CMD(P=1) }
```

```
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 1))
name ISO_141:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1))
name ISO_142:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;
   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) }
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
name ISO_143:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name ISO_144:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ P timer expired and RETRY_COUNT<N2 }
from AWAIT to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_145:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ Local busy cleared and DATA_FLAG=1 and IS_SREJ_USED=FALSE }
from AWAIT_BUSY to AWAIT_REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (is_SREJ_used = FALSE)
name ISO_146:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Local busy cleared and DATA_FLAG=0 }
from AWAIT_BUSY to AWAIT
provided local_busy_cleared and (DATA_flag = 0)
name ISO_147:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Local busy cleared and DATA_FLAG=2 }
from AWAIT_BUSY to AWAIT_REJECT
provided local_busy_cleared and (DATA_flag = 2)
name ISO_148:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message)
name ISO_149_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
```

```
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;
   DATA_flag := 1;
   remote_busy := 0;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message)
name ISO_149_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   DATA_flag := 1;
   remote_busy := 0;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   end;

{ Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
  Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message)
name ISO_150:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 1;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message)
name ISO_151:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 1;
   end;

{ Receive I_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
     and (get_PF_bit(message) = 1)
name ISO_152_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   DATA_flag := 1;
   output ip_P_timer.stop_P_timer;
   remote_busy := 0;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
```

```
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   end;

{ Receive I_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
     and (get_PF_bit(message) = 1)
name ISO_152_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   DATA_flag := 0;
   remote_busy := 0;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   end;

{ Receive I_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
     and (get_PF_bit(message) = 1)
name ISO_152_OPTION_3:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_P_timer.stop_P_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   DATA_flag := 0;
   remote_busy := 0;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   end;

{ Receive I_RSP(F=0) or Receive I_CMD(P=0) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0)
name ISO_153_OPTION_1:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 1;
   end;

{ Receive I_RSP(F=0) or Receive I_CMD(P=0) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0)
name ISO_153_OPTION_2:
   begin
   if choose_to_send_RNR_XXX = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_XXX(0), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 0;
```

```
   end;

{ Receive I_CMD(P=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_154_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 1;
   end;

{ Receive I_CMD(P=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1)
name ISO_154_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   DATA_flag := 0;
   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1))
name ISO_155_OPTION_1:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1))
name ISO_155_OPTION_2:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or REJ_CMD(P=0) or REJ_RSP(F=0) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 0))
name ISO_156:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) or REJ_CMD(P=1) }
from AWAIT_BUSY to same
```

```
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 1))
name ISO_157:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_RSP(F=1) }
from AWAIT_BUSY to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
name ISO_158:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;
   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
name ISO_159:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name ISO_160:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ P timer expired and RETRY_COUNT<N2 }
from AWAIT_BUSY to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name ISO_161:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;

{ LOCAL_BUSY_DETECTED }
from AWAIT_REJECT to AWAIT_BUSY
provided local_busy_detected
name ISO_162:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 2;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) or
  ) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0))
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_163:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = FALSE)
name ISO_164:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
```

```
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;
   end;

{ Receive I_RSP(F=1) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
      and (get_PF_bit(message) = 1) and (is_SREJ_used = FALSE)
name ISO_165_OPTION_1:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_CMD(P=1) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0));
      end
   else if resend_I_PDU_finished = FALSE then
      begin         { Resend the first unacked I_PDU as I_CMD(P=1) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the rest unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   output ip_REJ_timer.stop_REJ_timer;
   remote_busy := 0;
   end;

{ Receive I_RSP(F=1) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
      and (get_PF_bit(message) = 1) and (is_SREJ_used = FALSE)
name ISO_165_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_XXX(X=0)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_XXX(X=0) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_XXX(0), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 0, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ (Receive I_RSP(F=0) or Receive I_CMD(P=0)) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to AWAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0))
      and (is_SREJ_used = FALSE)
name ISO_166A:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   output ip_REJ_timer.stop_REJ_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;
   end;

{ Receive I_CMD(P=1) and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to AWAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1) and (is_SREJ_used = FALSE)
name ISO_167:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
```

```
      ack_P_REJ_timers_needed(0, 0, 0));
   output ip_REJ_timer.stop_REJ_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) or
  ( I_RSP(F=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE ) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
      or (get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
          and (with_unexpected_N_S(message) = TRUE) and (is_SREJ_used = FALSE)
name ISO_168_OPTION_1:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;

   { RESEND_I_XXX(X=0) }
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_RSP(F=1) or REJ_RSP(F=1) or
  ( I_RSP(F=1)_WITH_UNEXPECTED_N(S) and IS_SREJ_USED=FALSE ) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
      or (get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 1)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
          and (with_unexpected_N_S(message) = TRUE) and (is_SREJ_used = FALSE)
name ISO_168_OPTION_2:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);

   { RESEND_I_CMD(P=1) }
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   while resend_I_PDU_finished = FALSE do
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or REJ_CMD(P=0) or REJ_RSP(F=0) }
from AWAIT_REJECT to AWAIT_REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = REJ_RSP) and (get_PF_bit(message) = 0)
name ISO_169:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) or REJ_CMD(P=1) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1))
      or ((get_type2_PDU_type(message) = REJ_CMD) and (get_PF_bit(message) = 1))
name ISO_170:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 0;
   end;

{ Receive RNR_RSP(F=1) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
name ISO_171:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
```

```
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   V_S := get_N_R(message);
   output ip_P_timer.stop_P_timer;
   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
name ISO_172:
   begin

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1))
name ISO_173:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   remote_busy := 1;
   end;

{ P timer expired and RETRY_COUNT<N2 and IS_SREJ_USED=FALSE }
from AWAIT_REJECT to same
when ip_P_timer.P_timer_timeout
provided (retry_count < N2) and (is_SREJ_used = FALSE)
name ISO_174:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_REJ_CMD(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 1, 0));
   retry_count := retry_count + 1;
   end;


{********** Modification of ISO 8802-2 Table 4 ''Modification Table'' **********}

{ Receive FRMR_RSP and REASON=N(S)_ERROR }
from NORMAL to RSET
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message) = N_S_ERROR)
name MOD_1:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RSET_CMD, get_precedence, get_topology_update_id, ack_P_REJ_timers_needed
(1, 0, 0));
   retry_count := 0;
   end;

{ Receive FRMR_RSP and REASON<>N(S)_ERROR }
from NORMAL to RESET_WAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message) <> N_S_ERROR)
name MOD_2:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   S_flag := 0;
   end;

{ Receive FRMR_RSP }
from ISO_TRANSFER_STATES_EXCEPT_NORMAL to RESET_WAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = FRMR_RSP
name MOD_3:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   S_flag := 0;
   end;

{ Receive ZZZ_CMD_WITH_INVALID_N(R) }
from NORMAL to RSET
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_ZZZ_CMD(message) = TRUE) and with_invalid_N_R(message)
name MOD_4:
   begin
   if is_P_bit_1(message) = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RSET_CMD, get_precedence, get_topology_update_id, ack_P_REJ_timers_needed
(1, 0, 0));
   retry_count := 0;
   end;

{ Receive ZZZ_CMD_WITH_INVALID_N(R) }
from NORMAL to RSET
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_ZZZ_RSP(message) = TRUE) and with_invalid_N_R(message)
```

```
name MOD_5:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RSET_CMD, get_precedence, get_topology_update_id, ack_P_REJ_timers_neede
d(1, 0, 0));
   retry_count := 0;
   end;

{ Receive I_CMD_WITH_INVALID_N(S) }
from NORMAL to RSET_EXPECTED
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and with_invalid_N_S(message)
name MOD_6:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive I_RSP_WITH_INVALID_N(S) }
from NORMAL to RSET_EXPECTED
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and with_invalid_N_S(message)
name MOD_7:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive ZZZ_CMD_WITH_INVALID_N(R) or I_CMD_WITH_INVALID_N(S) }
from ISO_TRANSFER_STATES_EXCEPT_NORMAL to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_ZZZ_CMD(message) = TRUE) and with_invalid_N_R(message)
      or (get_type2_PDU_type(message) = I_CMD) and with_invalid_N_S(message)
name MOD_8:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive ZZZ_RSP_WITH_INVALID_N(R) or I_RSP_WITH_INVALID_N(S) }
from ISO_TRANSFER_STATES_EXCEPT_NORMAL to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (is_ZZZ_RSP(message) = TRUE) and with_invalid_N_R(message)
      or (get_type2_PDU_type(message) = I_RSP) and with_invalid_N_S(message)
name MOD_9:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0
   or Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0
   or Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1)
  ) and IS_SREJ_USED=TRUE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
          and with_unexpected_N_S(message) and (P_flag = 1))
     and (is_SREJ_used = TRUE)
name MOD_10_OPTION_1:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0
   or Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) and P_FLAG=0
   or Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1)
  ) and IS_SREJ_USED=TRUE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 0)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
          and with_unexpected_N_S(message) and (P_flag = 1))
     and (is_SREJ_used = TRUE)
name MOD_10_OPTION_2:
   begin
```

```
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) and P_FLAG=1 or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) P_FLAG=1
  ) and IS_SREJ_USED=TRUE }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 1)
      or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
          and with_unexpected_N_S(message) and (P_flag = 1))
     and (is_SREJ_used = TRUE)
name MOD_11:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from NORMAL to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_12:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_13_OPTION_1:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_13_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=1 or
  Receive SREJ_RSP(F=0) and P_FLAG=1 }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
name MOD_14:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
from NORMAL to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_15:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_RSP(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Local busy cleared }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 0) and (is_SREJ_used = TRUE)
```

```
name MOD_16_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   end;

{ Local busy cleared }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 0) and (is_SREJ_used = TRUE)
name MOD_16_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Local busy cleared }
from BUSY to REJECT
provided local_busy_cleared and (DATA_flag = 1)
     and (P_flag = 1) and (is_SREJ_used = TRUE)
name MOD_17:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_18_OPTION_1:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_18_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=1 or
  Receive SREJ_RSP(F=0) and P_FLAG=1 }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
name MOD_19:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
from BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_20:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) and P_FLAG=1
  ) and IS_SREJ_USED=TRUE }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
                                                and (P_flag = 1))
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_21:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
```

```
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_22:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ ( Receive I_RSP(F=X) and P_FLAG=F or Receive I_CMD(P=0) and P_FLAG=0 )
  and IS_SREJ_USED=TRUE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 0))
      and (is_SREJ_used = TRUE)
name MOD_23_OPTION_1:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_CMD(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_CMD(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   if no_more_buffered_out_of_order_data(v_out_of_order_data_buffer) then
      output ip_REJ_timer.stop_REJ_timer;
   end;

{ ( Receive I_RSP(F=X) and P_FLAG=F or Receive I_CMD(P=0) and P_FLAG=0 )
  and IS_SREJ_USED=TRUE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (P_FLAG = get_PF_bit(message))
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 0))
      and (is_SREJ_used = TRUE)
name MOD_23_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   if no_more_buffered_out_of_order_data(v_out_of_order_data_buffer) then
      output ip_REJ_timer.stop_REJ_timer;
   end;

{ If there are no more buffered out of order data, then change state }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided no_more_buffered_out_of_order_data(v_out_of_order_data_buffer)
     and (is_SREJ_used = TRUE)
name MOD_23_24_25_CHANGE_STATE:
   begin
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ ( Receive I_RSP(F=0) and P_FLAG=1 or Receive I_CMD(P=0) and P_FLAG=1 )
  and IS_SREJ_USED=TRUE }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 1)
       or (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
                                                and (P_FLAG = 1))
      and (is_SREJ_used = TRUE)
name MOD_24:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_XXX(0), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_XXX(0)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   if no_more_buffered_out_of_order_data(v_out_of_order_data_buffer) then
```

```
      output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive I_CMD(P=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1) and (is_SREJ_used = TRUE)
name MOD_25:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   if no_more_buffered_out_of_order_data(v_out_of_order_data_buffer) then
      output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_26_OPTION_1:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=0 or
  Receive SREJ_RSP(F=X) and P_FLAG=F }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (P_flag = get_PF_bit(message))
name MOD_26_OPTION_2:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) and P_FLAG=1 or
  Receive SREJ_RSP(F=0) and P_FLAG=1 }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0) and (P_flag = 1)
name MOD_27:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
from REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_28:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_RSP(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ REJ timer expired and RETRY_COUNT < N2 }
from REJECT to same
when ip_REJ_timer.REJ_timer_timeout
provided (P_flag = 0) and (retry_count < N2) and (is_SREJ_used = TRUE)
name MOD_29_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count + 1;
   end;

{ REJ timer expired and RETRY_COUNT < N2 }
from REJECT to NORMAL
when ip_REJ_timer.REJ_timer_timeout
provided (P_flag = 0) and (retry_count < N2) and (is_SREJ_used = TRUE)
name MOD_29_OPTION_2:
   begin
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_30_OPTION_1:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);
```

```
   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   output ip_P_timer.stop_P_timer;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_30_OPTION_2:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S)
  ) and IS_SREJ_USED=TRUE }
from AWAIT to AWAIT_REJECT   {amer 3/7/98}
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0))
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_31:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from AWAIT to AWAIT_REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_32:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_33A_OPTION_1:
   begin
   output ip_P_timer.stop_P_timer;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_33A_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) or SREJ_RSP(F=0) }
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0)
name MOD_33B:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
```

```
from AWAIT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_33C:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Local busy cleared }
from AWAIT_BUSY to AWAIT_REJECT
provided local_busy_cleared and (DATA_flag = 1) and (is_SREJ_used = TRUE)
name MOD_34:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_35_OPTION_1:
   begin
   output ip_P_timer.stop_P_timer;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_35_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) or SREJ_RSP(F=0) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0)
name MOD_36:
   begin
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
from AWAIT_BUSY to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_37:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ (Receive I_CMD(P=0)_WITH_UNEXPECTED_N(S) or
   Receive I_RSP(F=0)_WITH_UNEXPECTED_N(S)
  ) and IS_SREJ_USED=TRUE }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 0)
       or (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 0))
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_38:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_39:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   end;

{ Receive I_RSP(F=1) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
      and (get_PF_bit(message) = 1) and (is_SREJ_used = TRUE)
name MOD_40_OPTION_1:
   begin
   V_R := V_R + 1;
```

```
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;


   { RESEND_I_CMD(P=1)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_CMD(P=1) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0));
      end
   else if resend_I_PDU_finished = FALSE then
      begin         { Resend the first unacked I_PDU as I_CMD(P=1) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_CMD(1, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the rest unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   remote_busy := 0;
   end;

{ Receive I_RSP(F=1) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP)
      and (get_PF_bit(message) = 1) and (is_SREJ_used = TRUE)
name MOD_40_OPTION_2:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_P_timer.stop_P_timer;

   { RESEND_I_XXX(X=0)_OR_SEND_RR }
   if choose_to_send_RR then
      begin         { Send RR_XXX(X=0) first }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_XXX(0), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 0, 0));
      end;
   while resend_I_PDU_finished = FALSE do
      begin         { Resend the unacked I_PDUs as I_XXX(X=0) }
      output ip_type2SAP_lower_mux.connection_component_message(
         reconstruct_1_I_XXX(0, next_unacked_seqno), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
      end;

   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   remote_busy := 0;
   end;

{ If there are no more buffered out of order data, then change state }
from REJECT to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided no_more_buffered_out_of_order_data(v_out_of_order_data_buffer)
     and (is_SREJ_used = TRUE)
name MOD_40_CHANGE_STATE:
   begin
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ (Receive I_RSP(F=0) or I_CMD(P=0)) and IS_SREJ_USED=TRUE }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_RSP)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = I_CMD)
          and (get_PF_bit(message) = 0))
     and (is_SREJ_used = TRUE)
name MOD_41:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_XXX(0), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   end;

{ Receive I_CMD(P=1) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1) and (is_SREJ_used = TRUE)
name MOD_42:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
```

```
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   process_buffered_out_of_order_data(v_out_of_order_data_buffer);
   end;

{ If there are no more buffered out of order data, then change state }
from AWAIT_REJECT to AWAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided no_more_buffered_out_of_order_data(v_out_of_order_data_buffer)
     and (is_SREJ_used = TRUE)
name MOD_41_42_CHANGE_STATE:
   begin
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_43_OPTION_1:
   begin
   output ip_P_timer.stop_P_timer;
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_RSP(F=1) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_RSP)
     and (get_PF_bit(message) = 1)
name MOD_43_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=0) or SREJ_RSP(F=0) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
          and (get_PF_bit(message) = 0)
      or (get_type2_PDU_type(message) = SREJ_RSP)
          and (get_PF_bit(message) = 0)
name MOD_44:
   begin
   remote_busy := 0;
   end;

{ Receive SREJ_CMD(P=1) }
from AWAIT_REJECT to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = SREJ_CMD)
     and (get_PF_bit(message) = 1)
name MOD_45:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR_RSP(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_46_OPTION_1:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_XXX(0, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   output ip_P_timer.stop_P_timer;
   end;

{ Receive I_RSP(F=1)_WITH_UNEXPECTED_N(S) }
from AWAIT_REJECT to REJECT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and (get_PF_bit(message) = 1)
      and with_unexpected_N_S(message) and (is_SREJ_used = TRUE)
name MOD_46_OPTION_2:
   begin
   process_and_buffer_out_of_order_data(message, v_out_of_order_data_buffer);

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 1));
   output ip_type2SAP_lower_mux.connection_component_message(
      reconstruct_1_I_CMD(1, get_N_R(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   remote_busy := 0;
   end;
```

```
{ P timer expired and RETRY_COUNT < N2 }
from AWAIT_REJECT to same
when ip_P_timer.P_timer_timeout
provided (retry_count < N2) and (is_SREJ_used = TRUE)
name MOD_47:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SREJ_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 1));
   retry_count := retry_count + 1;
   end;

{********** Multi-destination-PDU-related transitions ''Multi'' table **********}

{ Receive MULTI_DATA_REQUEST }
from MULTI_NOT_READY_STATES to same
when ip_type2SAP_upper_mux.multidata_request
name MULTI_1:
   begin
   output ip_type2SAP_lower_mux.multi_not_ready;
   end;

{ Receive MULTI_DATA_REQUEST }
from NORMAL_OR_BUSY_OR_REJECT to same
when ip_type2SAP_upper_mux.multidata_request
provided (remote_busy = 0) and (is_send_window_open = TRUE)
     and (NOT exist_buffered_data_request(v_unitdata_buffer)) and (P_flag = 0)
name MULTI_2_4_6_OPTION_1:
   begin
   output ip_type2SAP_lower_mux.multi_ready(
      construct_PDU_control(1), ack_P_REJ_timers_needed(1, 1, 0));
   buffer_multidata_request(v_unitdata_buffer);
   end;

{ Receive MULTI_DATA_REQUEST }
from NORMAL_OR_BUSY_OR_REJECT to same
when ip_type2SAP_upper_mux.multidata_request
provided (remote_busy = 0) and (is_send_window_open = TRUE)
     and (NOT exist_buffered_data_request(v_unitdata_buffer)) and (P_flag = 0)
name MULTI_2_4_6_OPTION_2:
   begin
   output ip_type2SAP_lower_mux.multi_ready(
      construct_PDU_control(0), ack_P_REJ_timers_needed(1, 0, 0));
   buffer_multidata_request(v_unitdata_buffer);
   end;

{ Receive MULTI_DATA_REQUEST }
from NORMAL_OR_BUSY_OR_REJECT to same
when ip_type2SAP_upper_mux.multidata_request
provided (remote_busy = 0) and (is_send_window_open = TRUE)
     and (NOT exist_buffered_data_request(v_unitdata_buffer)) and (P_flag = 1)
name MULTI_3_5_7:
   begin
   output ip_type2SAP_lower_mux.multi_ready(
      construct_PDU_control(0), ack_P_REJ_timers_needed(1, 0, 0));
   buffer_multidata_request(v_unitdata_buffer);
   end;

{ Receive MULTI_DATA_REQUEST }
from NORMAL_OR_BUSY_OR_REJECT to same
when ip_type2SAP_upper_mux.multidata_request
provided (remote_busy = 1) or (is_send_window_open = FALSE)
      or exist_buffered_data_request(v_unitdata_buffer)
name MULTI_8:
   begin
   output ip_type2SAP_lower_mux.multi_not_ready;
   buffer_multidata_request(v_unitdata_buffer);
   end;

{ Receive MULTI_DATA_REQUEST }
from AWAIT_STATES to same
when ip_type2SAP_upper_mux.multidata_request
name MULTI_9:
   begin
   output ip_type2SAP_lower_mux.multi_not_ready;
   buffer_multidata_request(v_unitdata_buffer);
   end;

{ Receive UNIT_DATA_REQUEST }
from ALL_TRANSFER_STATES to same
when ip_type2SAP_upper_mux.unitdata_request
name MULTI_10:
   begin
   buffer_unitdata_request(v_unitdata_buffer);
   end;


{********************************* RSET Transitions *********************************}

{ Receive UA_RSP and local station is NOT busy }
from RSET to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = UA_RSP) and (local_busy = 0)
name RSET_1A:
   begin
   V_S := 0;
   output ip_ack2_timer.stop_ack_timer;
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   end;

{ Receive UA_RSP and local station is busy }
from RSET to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = UA_RSP) and (local_busy = 1)
name RSET_1B:
   begin
   V_S := 0;
   output ip_ack2_timer.stop_ack_timer;
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   end;

{ Ack timer expired and RETRY_COUNT < N2 }
from RSET to same
when ip_ack2_timer.ack_timer_timeout
provided retry_count < N2
name RSET_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RSET_CMD, get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
```

```
   end;

{ Receive DISCONNECT_REQUEST }
from RSET to D_CONN
when ip_type2SAP_upper_mux.disconnect_request
name RSET_3:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DISC_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive RESET_REQUEST }
from RSET to RESET
when ip_type2SAP_upper_mux.reset_request
name RSET_4:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   S_flag := 0;
   end;

{ Receive SABME_CMD }
from RSET to RESET_CHECK
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name RSET_5:
   begin
   output ip_type2SAP_upper_mux.reset_indication(REMOTE);
   F_flag := get_PF_bit(message);
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive DISC_CMD }
from RSET to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name RSET_6:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive FRMR_RSP and REASON=N(S)_ERROR }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message) = N_S_ERROR)
name RSET_7:
   begin
   { RSET has already been sent; simply report FRMR_RECEIVED }
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   end;

{ Receive FRMR_RSP and REASON<>N(S)_ERROR }
from RSET to RESET_WAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message)<>N_S_ERROR)
name RSET_8:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   S_flag := 0;
   end;

{ Receive DM_RSP }
from RSET to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name RSET_9:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive I_CMD_WITH_INVALID_N(S) }
from RSET to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and with_invalid_N_S(message)
name RSET_10:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive I_RSP_WITH_INVALID_N(S) or BAD_PDU }
from RSET to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and with_invalid_N_S(message)
      or (bad_PDU(message) = TRUE)
name RSET_11:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_type2SAP_upper_mux.report_status(FRMR_SENT);
```

```
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive RSET_CMD }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = RSET_CMD
name RSET_12:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ LOCAL_BUSY_DETECTED }
from RSET to same
provided local_busy_detected and (P_flag = 0)
name RSET_13:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   DATA_flag := 0;
   local_busy := 1;
   end;

{ LOCAL_BUSY_DETECTED }
from RSET to same
provided local_busy_detected and (P_flag = 1)
name RSET_14:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 0;
   local_busy := 1;
   end;

{ Receive I_CMD_WITH_UNEXPECTED_N(S) }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and with_unexpected_N_S(message)
name RSET_15:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   end;

{ Receive I_XXX and local station is busy }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD)
         or (get_type2_PDU_type(message) = I_RSP))
      and (local_busy = 1)
name RSET_16:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR, get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_RNR));
   DATA_flag := 1;
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   { Data could be either discarded or delivered according to local busy situation }
   end;

{ Receive I_XXX and local station is NOT busy }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD)
         or (get_type2_PDU_type(message) = I_RSP))
      and (local_busy = 0)
name RSET_17:
   begin
   V_R := V_R + 1;
   output ip_type2SAP_upper_mux.data_indication(
      get_dest_addr_list(message), get_src_addr(message), topology_update_id,
      get_data(message), get_data_length(message));
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RR, get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_RR));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   if is_F_bit_1(message) = TRUE then
      remote_busy := 0;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or (RR_RSP(F=1) and P_FLAG=1) }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
                                                 and (P_flag = 1))
name RSET_18:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1)
name RSET_19:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_RSP(1), get_precedence,
```

```
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) or (RNR_RSP(F=1) and P_FLAG=1) }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
                                                  and (P_flag = 1))
name RSET_20:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name RSET_21:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   remote_busy := 1;
   end;

{ Receive REJ_XXX or SREJ_XXX }
from RSET to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_XXX)
      or (get_type2_PDU_type(message) = SREJ_XXX)
name RSET_22:
   begin
   { RSET has already been sent }
   end;

{ INITIATE_P/F_CYCLE and P_FLAG=0}
from RSET to same
provided initiate_P_F_cycle and (P_flag = 0)
name RSET_23:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0));
   end;

{ P timer expired and RETRY_COUNT < N2 }
from RSET to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name RSET_24:
   begin
   P_flag := 0;
   end;

{ P timer expired and RETRY_COUNT >= N2 }
from RSET to RESET_WAIT
when ip_P_timer.P_timer_timeout
provided retry_count >= N2
name RSET_25A:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{ Ack timer expired and RETRY_COUNT >= N2 }
from RSET to RESET_WAIT
when ip_ack2_timer.ack_timer_timeout
provided retry_count >= N2
name RSET_25B:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{****************************** RSET_EXPECTED Transitions ****************************}

{ Receive RSET_CMD and local station is NOT busy }
from RSET_EXPECTED to NORMAL
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RSET_CMD) and (local_busy = 0)
name RSET_EXP_1A:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   end;


{ Receive RSET_CMD and local station is busy }
from RSET_EXPECTED to BUSY
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RSET_CMD) and (local_busy = 1)
name RSET_EXP_1B:
   begin
   V_R := 0;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence, get_topology_update_id,
      ack_P_REJ_timers_needed(0, 0, 0));
   end;

{ Ack timer expired and RETRY_COUNT<N2 }
```

```
from RSET_EXPECTED to same
when ip_ack2_timer.ack_timer_timeout
provided retry_count < N2
name RSET_EXP_2:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_FRMR_RSP(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   retry_count := retry_count + 1;
   end;

{ Receive DISCONNECT_REQUEST }
from RSET_EXPECTED to D_CONN
when ip_type2SAP_upper_mux.disconnect_request
name RSET_EXP_3:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_DISC_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   end;

{ Receive RESET_REQUEST }
from RSET_EXPECTED to RESET
when ip_type2SAP_upper_mux.reset_request
name RSET_EXP_4:
   begin
   P_flag := select_P_bit;
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_SABME_CMD(P_flag), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(1, 0, 0));
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   retry_count := 0;
   S_flag := 0;
   end;

{ Receive SABME_CMD }
from RSET_EXPECTED to RESET_CHECK
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = SABME_CMD
name RSET_EXP_5:
   begin
   output ip_type2SAP_upper_mux.reset_indication(REMOTE);
   F_flag := get_PF_bit(message);
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive DISC_CMD }
from RSET_EXPECTED to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DISC_CMD
name RSET_EXP_6:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_UA_RSP(get_PF_bit(message)), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive FRMR_RSP and REASON=N(S)_ERROR }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message) = N_S_ERROR)
name RSET_EXP_7:
   begin
   { RSET has already been sent; simply report FRMR_RECEIVED }
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   end;

{ Receive FRMR_RSP and REASON<>N(S)_ERROR }
from RSET_EXPECTED to RESET_WAIT
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = FRMR_RSP)
     and (FRMR_reason(message)<>N_S_ERROR)
name RSET_EXP_8:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   output ip_type2SAP_upper_mux.report_status(FRMR_RECEIVED);
   S_flag := 0;
   end;

{ Receive DM_RSP }
from RSET_EXPECTED to ADM
when ip_type2SAP_lower_mux.type2SAP_message
provided get_type2_PDU_type(message) = DM_RSP
name RSET_EXP_9:
   begin
   output ip_type2SAP_upper_mux.disconnect_indication;
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   end;

{ Receive I_CMD_WITH_INVALID_N(S) }
from RSET_EXPECTED to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD) and with_invalid_N_S(message)
name RSET_EXP_10:
   begin
   if is_P_bit_1(message) = TRUE then
      begin
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
         timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));
      end;
   end;

{ Receive I_RSP_WITH_INVALID_N(S) or BAD_PDU }
from RSET_EXPECTED to ERROR
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_RSP) and with_invalid_N_S(message)
```

```
      or (bad_PDU(message) = TRUE)
name RSET_EXP_11:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   end;

{ LOCAL_BUSY_DETECTED }
from RSET_EXPECTED to same
provided local_busy_detected and (P_flag = 0)
name RSET_EXP_13:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_CMD(1), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 1, 0));
   DATA_flag := 0;
   local_busy := 1;
   end;

{ LOCAL_BUSY_DETECTED }
from RSET_EXPECTED to same
provided local_busy_detected and (P_flag = 1)
name RSET_EXP_14:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_RNR_XXX(0), get_precedence,
      get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0));
   DATA_flag := 0;
   local_busy := 1;
   end;

{ Receive I_CMD(P=1)_WITH_UNEXPECTED_N(S) }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = I_CMD)
     and (get_PF_bit(message) = 1) and with_unexpected_N_S(message)
name RSET_EXP_15:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence,
         get_topology_update_id, ack_P_REJ_timers_needed(0, 0, 0))
   end;

{ Receive I_XXX and local station is busy }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD)
         or (get_type2_PDU_type(message) = I_RSP))
     and (local_busy = 1)
name RSET_EXP_16:
   begin
   if is_P_bit_1(message) = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_RSP(1), get_precedence, get_topology_update_id,
         timers_needed_according_to_PDU_sent(construct_RNR));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;

   { Data could be either discarded or delivered according to local busy situation }
   end;

{ Receive I_XXX and local station is NOT busy }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = I_CMD)
         or (get_type2_PDU_type(message) = I_RSP))
     and (local_busy = 0)
name RSET_EXP_17:
   begin
   if is_P_bit_1(message) = TRUE then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_RSP(1), get_precedence, get_topology_update_id,
         timers_needed_according_to_PDU_sent(construct_RNR));
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;

   { UPDATE_N(R)_RECEIVED }
   update_N_R;
   retry_count := 0;
   output ip_ack2_timer.stop_ack_timer;
   if exist_unacked_PDU then
      output ip_ack2_timer.start_ack_timer;
   end;

{ Receive RR_CMD(P=0) or RR_RSP(F=0) or (RR_RSP(F=1) and P_FLAG=1) }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RR_RSP) and (get_PF_bit(message) = 1)
                                                 and (P_flag = 1))
name RSET_EXP_18:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   remote_busy := 0;
   end;

{ Receive RR_CMD(P=1) }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RR_CMD) and (get_PF_bit(message) = 1)
name RSET_EXP_19:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
```

```
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));
   remote_busy := 0;
   end;

{ Receive RNR_CMD(P=0) or RNR_RSP(F=0) or (RNR_RSP(F=1) and P_FLAG=1) }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided ((get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 0))
      or ((get_type2_PDU_type(message) = RNR_RSP) and (get_PF_bit(message) = 1)
                                                  and (P_flag = 1))
name RSET_EXP_20:
   begin
   if is_F_bit_1(message) = TRUE then    { UPDATE_P_FLAG }
      begin
      P_flag := 0;
      output ip_P_timer.stop_P_timer;
      end;
   remote_busy := 1;
   end;

{ Receive RNR_CMD(P=1) }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = RNR_CMD) and (get_PF_bit(message) = 1)
name RSET_EXP_21:
   begin
   output ip_type2SAP_lower_mux.connection_component_message(
      construct_ACKNOWLEDGE_RSP(1), get_precedence, get_topology_update_id,
      timers_needed_according_to_PDU_sent(construct_ACKNOWLEDGE_RSP(1)));
   remote_busy := 1;
   end;

{ Receive REJ_XXX or SREJ_XXX }
from RSET_EXPECTED to same
when ip_type2SAP_lower_mux.type2SAP_message
provided (get_type2_PDU_type(message) = REJ_XXX)
      or (get_type2_PDU_type(message) = SREJ_XXX)
name RSET_EXP_22:
   begin
   { RSET has already been sent }
   end;

{ INITIATE_P/F_CYCLE and P_FLAG=0}
from RSET_EXPECTED to same
provided initiate_P_F_cycle and (P_flag = 0)
name RSET_EXP_23:
   begin
   if local_busy = 1 then
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RNR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0))
   else { local_busy = 0 }
      output ip_type2SAP_lower_mux.connection_component_message(
         construct_RR_CMD(1), get_precedence, get_topology_update_id,
         ack_P_REJ_timers_needed(0, 1, 0));
   end;

{ P timer expired and RETRY_COUNT < N2 }
from RSET_EXPECTED to same
when ip_P_timer.P_timer_timeout
provided retry_count < N2
name RSET_EXP_24:
   begin
   P_flag := 0;
   end;

{ P timer expired and RETRY_COUNT >= N2 }
from RSET_EXPECTED to RESET_WAIT
when ip_P_timer.P_timer_timeout
provided retry_count >= N2
name RSET_EXP_25A:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_ack2_timer.stop_ack_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

{ Ack timer expired and RETRY_COUNT >= N2 }
from RSET_EXPECTED to RESET_WAIT
when ip_ack2_timer.ack_timer_timeout
provided retry_count >= N2
name RSET_EXP_25B:
   begin
   output ip_busy_timer.stop_busy_timer;
   output ip_P_timer.stop_P_timer;
   output ip_REJ_timer.stop_REJ_timer;
   output ip_type2SAP_upper_mux.reset_indication(LOCAL);
   S_flag := 0;
   end;

      end; { b_type2_connection_component }

      {*****************************************}
      {   Type2 SAP Component Body: Transitions   }
      {*****************************************}
      state
         NORMAL, MULTI_DESTINATION_I_PDU_OPERATION;

      var
         v_ack_failure: type_ack_failure;
         v_multi_info: type_multi_info;
         v_null_DL_Unitdata_id: type_DL_Unitdata_id;
         v_null_ack_failure: type_ack_failure;
         v_null_address_list: type_addr_list;
         v_timer_list: type_type2_timer_list;

      modvar
         connection_component: array [type_station_number_range]
            of m_type2_connection_component;
         busy_timer  : array [type_station_number_range] of m_type2_busy_timer;
         ack2_timer   : array [type_station_number_range] of m_type2_ack_timer;
         P_timer     : array [type_station_number_range] of m_type2_P_timer;
         REJ_timer   : array [type_station_number_range] of m_type2_REJ_timer;

      initialize to NORMAL
         begin

         { Initiate Type 2 connection components, busy, ack, P and REJ timers;}
         all i: type_station_number_range do
            begin
            init connection_component[i] with b_type2_connection_component;
            init busy_timer[i]           with b_type2_busy_timer;
            init ack2_timer[i]            with b_type2_ack_timer;
            init P_timer[i]              with b_type2_P_timer;
            init REJ_timer[i]            with b_type2_REJ_timer;

            connect ip_connection_component_upper_mux[i] to
```

```
                connection_component[i].ip_type2SAP_upper_mux;
        connect ip_connection_component_lower_mux[i] to
                connection_component[i].ip_type2SAP_lower_mux;
        connect connection_component[i].ip_busy_timer to
                busy_timer[i].ip_connection_component;
        connect connection_component[i].ip_ack2_timer to
                ack2_timer[i].ip_connection_component;
        connect connection_component[i].ip_P_timer to
                P_timer[i].ip_connection_component;
        connect connection_component[i].ip_REJ_timer to
                REJ_timer[i].ip_connection_component;

        attach ip_station_ack2_timer[i] to
               ack2_timer[i].ip_station;
        attach ip_station_P_timer[i] to
               P_timer[i].ip_station;
        attach ip_station_REJ_timer[i] to
               REJ_timer[i].ip_station;
        end;

    { Release the connection component and timers for this station }
    release connection_component[whoami];
    release busy_timer[whoami];
    release ack2_timer[whoami];
    release P_timer[whoami];
    release REJ_timer[whoami];

    { The following variables are initialized and used as constants for
      constructing DL_Status_Ind purpose }
    init_null_DL_Unitdata_id(v_null_DL_Unitdata_id);
    init_null_ack_failure(v_null_ack_failure);
    init_null_address_list(v_null_address_list);

    end;

trans

{** Transition TYPE2SAP_1 - TYPE2SAP_4: multi-destination I PDU operations **}

{ Receive DL_Unitdata_Req from Station component }
from NORMAL to MULTI_DESTINATION_I_PDU_OPERATION
when ip_station_upper_mux.DL_Unitdata_Req
provided has_multi_destinations(dest_addr_list) = TRUE
name TYPE2SAP_1:
   begin
   record_DL_Unitdata_Req_for_multi_processing(v_multi_info);
   init_timer_list(v_timer_list);
   all i: 1..NS_LESS_1 do
      if is_destination_in_dest_addr_list(i, dest_addr_list) then
         output ip_connection_component_upper_mux[i].multidata_request(
            DL_Unitdata_id, dest_addr_list,src_addr, topology_update_id,
            precedence, data, data_length);
   end;

{ Receive multi_ready response from connection component }
from MULTI_DESTINATION_I_PDU_OPERATION to NORMAL
any i: type_station_number_range do
when ip_connection_component_lower_mux[i].multi_ready
name TYPE2SAP_2:
   begin
   record_multi_ready(i, PDU_control, timers_needed, v_multi_info, v_timer_list);
   end;

{ Receive multi_not_ready response from connection component }
from MULTI_DESTINATION_I_PDU_OPERATION to NORMAL
any i: type_station_number_range do
when ip_connection_component_lower_mux[i].multi_not_ready
name TYPE2SAP_3:
   begin
   record_multi_not_ready(i, v_multi_info);
   end;

{ All multi-responses are collected }
from MULTI_DESTINATION_I_PDU_OPERATION to same
provided are_all_multi_responses_received(v_multi_info) = TRUE
name TYPE2SAP_4:
   begin
   output ip_station_lower_mux.type2SAP_message(
      construct_DL_PDU_from_multi_info(v_multi_info),
      get_precedence_from_multi_info(v_multi_info),
      get_topology_update_id_from_multi_info(v_multi_info),
      v_timer_list);
   end;

{ Receive connection_component_message }
from MULTI_DESTINATION_I_PDU_OPERATION to same
any i: type_station_number_range do
when ip_connection_component_lower_mux[i].connection_component_message
provided is_waiting_multi_response(i) = TRUE
name TYPE2SAP_5:
   begin
   output ip_station_lower_mux.type2SAP_message(
       message, precedence, topology_update_id,
       construct_timer_list_from_timers_needed(timers_needed));
   end;

{** Transition TYPE2SAP_11 - TYPE2SAP_13: Forwarding service **}

{ Receive connection_component_message }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_lower_mux[i].connection_component_message
name TYPE2SAP_11:
   begin
   output ip_station_lower_mux.type2SAP_message(
      message, precedence, topology_update_id,
      construct_timer_list_from_timers_needed(timers_needed));
   end;

{ Receive DL_Unitdata_Req from station component }
from NORMAL to same
when ip_station_upper_mux.DL_Unitdata_Req
provided has_multi_destinations(dest_addr_list) = FALSE
name TYPE2SAP_12:
   begin
   output ip_connection_component_upper_mux[get_destination(dest_addr_list)].
      unitdata_request(DL_Unitdata_id, dest_addr_list, src_addr,
      topology_update_id, precedence, data, data_length);
   end;

{ Receive DL_PDU from station component }
from NORMAL to same
when ip_station_lower_mux.station_message
name TYPE2SAP_13:
   begin
   output ip_connection_component_lower_mux
      [get_connection_component_number(message)].
      type2SAP_message(message, topology_update_id);
   end;
```

```
{** Transition TYPE2SAP_21 - TYPE2SAP_23: SAP component initiated operations **}

{ SAP should initiate CONNECT_REQUEST }
from NORMAL to same
any i: type_station_number_range do
provided should_init_connect(i) = TRUE
name TYPE2SAP_21:
   begin
   output ip_connection_component_upper_mux[i].connect_request;
   end;

{ SAP should initiate DISCONNECT_REQUEST }
from NORMAL to same
any i: type_station_number_range do
provided should_init_disconnect(i) = TRUE
name TYPE2SAP_22:
   begin
   output ip_connection_component_upper_mux[i].disconnect_request;
   output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
      v_null_ack_failure, v_null_address_list, DISCONNECTED);
   end;

{ SAP should initiate RESET_REQUEST }
from NORMAL to same
any i: type_station_number_range do
provided should_init_reset(i) = TRUE
name TYPE2SAP_23:
   begin
   output ip_connection_component_upper_mux[i].reset_request;
   end;

{** Transition TYPE2SAP_31 - TYPE2SAP_39:
        Process connection component Indications, Confirms and Reports **}

{ Receive DATA_INDICATION from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].data_indication
name TYPE2SAP_31:
   begin
   output ip_station_upper_mux.DL_Unitdata_Ind(
      dest_addr_list, src_addr, topology_update_id, data, data_length);
   end;

{ Receive CONNECT_INDICATION from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].connect_indication
name TYPE2SAP_32:
   begin
   if is_connect_acceptable(i) = TRUE then
      begin
      output ip_connection_component_upper_mux[i].connect_response;
      output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
         v_null_ack_failure, v_null_address_list, ESTABLISHED);
      end
   else
      begin
      output ip_connection_component_upper_mux[i].disconnect_request;
      output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
         v_null_ack_failure, v_null_address_list, DISCONNECTED);
      end;
   record_connection_status(i);
   end;

{ Receive DISCONNECT_INDICATION from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].disconnect_indication
name TYPE2SAP_33:
   begin
   record_connection_status(i);
   output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
      v_null_ack_failure, v_null_address_list, DISCONNECTED);
   end;

{ Receive RESET_INDICATION from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].reset_indication
name TYPE2SAP_34_AND_35:
   begin
   if reset_originator = REMOTE then
      begin
      if is_reset_acceptable(i) = TRUE then
         begin
         output ip_connection_component_upper_mux[i].reset_response;
         output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
            v_null_ack_failure, v_null_address_list, RESET);
         end
      else
         begin
         output ip_connection_component_upper_mux[i].disconnect_request;
         output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
            v_null_ack_failure, v_null_address_list, DISCONNECTED);
         end;
      end
   else if reset_originator = LOCAL then
      begin
      if is_reset_acceptable(i) = TRUE then
         begin
         output ip_connection_component_upper_mux[i].reset_request;
         end
      else
         begin
         output ip_connection_component_upper_mux[i].disconnect_request;
         output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
            v_null_ack_failure, v_null_address_list, DISCONNECTED);
         end;
      end;
   record_connection_status(i);
   end;

{ Receive CONNECT_CONFIRM from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].connect_confirm
name TYPE2SAP_36:
   begin
   record_connection_status(i);
   output ip_station_upper_mux.DL_Status_Ind(v_null_DL_Unitdata_id,
      v_null_ack_failure, v_null_address_list, ESTABLISHED);
   end;

{ Receive RESET_CONFIRM from connection component }
from NORMAL to same
any i: type_station_number_range do
when ip_connection_component_upper_mux[i].reset_confirm
name TYPE2SAP_37:
   begin
```

```
         record_connection_status(i);
         end;

      { Receive REPORT_STATUS from connection component }
      from NORMAL to same
      any i: type_station_number_range do
      when ip_connection_component_upper_mux[i].report_status
      name TYPE2SAP_38:
         begin
         record_report_status(i, status_reported);
         end;

      { Receive ACK_FAILURE_INDICATION from connection component }
      from NORMAL to same
      any i: type_station_number_range do
      when ip_connection_component_upper_mux[i].ack_failure_indication
      name TYPE2SAP_39:
         begin
         output ip_station_upper_mux.DL_Status_Ind(DL_Unitdata_id, v_ack_failure,
            address_list, get_connection_status(i));
         end;

   end; { b_type2SAP_component }

{*****************************************************}
{ Station Component body definition: 1 phase -- Active }
{*****************************************************}
state
   ACTIVE,
   WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION;

stateset
   JUST_ACTIVE = [ ACTIVE ];
   JUST_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION =
                [ WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION];
   ALL_STATES = [
                ACTIVE,
                WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION
                ];

var
   { Main station component parameters and data structures }
   v_FEC_TDC_scrambling: type_FEC_TDC_scrambling;
   v_link_parameter_table: type_link_parameter_table;
   v_multidwell: type_multidwell;
   v_my_address: type_address;
   v_net_activity: type_net_activity;
   v_quiet_mode: type_quiet_mode;
   v_rexmis_count: type_rexmis_count;
   v_station_message_table: type_station_message_table;
   v_topology_update_id_received: type_topology_update_id;
   v_unique_id: type_unique_id;

   { Other station component variables and data structures }
   v_DL_PDU_received: type_DL_PDU;
   v_DL_PDU_retrieved: type_DL_PDU;
   v_DL_PDU_addressee_list: type_DL_PDU_addressee_list;
   v_DL_PDU_type: type_service_type;
   v_all_addressee_list: type_all_addressee_list;
   v_coupled_ack: type_DL_PDU;
   v_from_component: type_from_component;
   v_message_id: type_message_id;
   v_need_coupled_ack: boolean;
   v_n_th: integer;
   v_no_of_expected_acks: integer;
   v_service: type_service_type;
   v_timer: integer;
   v_transmission_to_be_sent: type_data;
   v_transmission_length: type_data_length;
   v_transmission_type2_timer_list: type_transmission_type2_timer_list;

modvar
   type1SAP_component: m_type1SAP_component;
   URR_URNR_busy_timer: array [0..NS_LESS_1] of m_URR_URNR_busy_timer;
   ack_timer: m_ack_timer;
   type2SAP_component: m_type2SAP_component;

initialize to ACTIVE
   begin
      {* Init Type 1 SAP component *}
      init type1SAP_component with b_type1SAP_component;
      init ack_timer     with b_ack_timer;
      all i: 0..NS_LESS_1 do
         init URR_URNR_busy_timer[i] with b_URR_URNR_busy_timer;
      connect ip_type1SAP_lower_mux to type1SAP_component.ip_station_lower_mux;
      connect ip_type1SAP_upper_mux to type1SAP_component.ip_station_upper_mux;
      connect ip_ack_timer     to ack_timer.ip_station;
      connect type1SAP_component.ip_ack_timer to ack_timer.ip_type1SAP;
      all i: 0..NS_LESS_1 do
         connect type1SAP_component.ip_busy_timer[i] to
                 URR_URNR_busy_timer[i].ip_type1SAP;

      {* Init Type2 SAP component *}
      init type2SAP_component with b_type2SAP_component;
      connect ip_type2SAP_lower_mux to type2SAP_component.ip_station_lower_mux;
      connect ip_type2SAP_upper_mux to type2SAP_component.ip_station_upper_mux;
      all i: 0..NS_LESS_1 do
         begin
         connect ip_ack2_timer[i] to type2SAP_component.ip_station_ack2_timer[i];
         connect ip_P_timer[i]   to type2SAP_component.ip_station_P_timer[i];
         connect ip_REJ_timer[i] to type2SAP_component.ip_station_REJ_timer[i];
         end;

      connect ip_transmission to ip_DL_PDU;

      attach ip_OPERATOR_type1SAP to type1SAP_component.ip_OPERATOR;
      attach ip_OPERATOR_type2SAP to type2SAP_component.ip_OPERATOR;

      v_multidwell := SIX; { 6 segments per packet * }
      v_my_address := NET_ENTRY_ADDRESS;
      v_net_activity := BUSY;
      v_quiet_mode := OFF;

      init_unique_id(v_unique_id);
      init_link_parameter_table(v_link_parameter_table);
      init_station_message_table(v_station_message_table);
      init_FEC_TDC_scrambling(v_FEC_TDC_scrambling); { * }

   end;

trans

{*************************}
{ Active Phase Transitions }
{*************************}
{ TYPE1SAP Component provide a new message }
from ACTIVE to same
when ip_type1SAP_lower_mux.type1SAP_message
priority MEDIUM
name A_1:
```

```
   begin
   buffer_message_in_station_message_table(
      v_station_message_table, message_id, message, FROM_TYPE1SAP, precedence,
      topology_update_id);
   end;

{ PL_Unitdata_Ind arrives from Physical Layer }
from ACTIVE to same
when ip_DL_PDU.DL_PDU_form_PL_Unitdata_Ind
priority MEDIUM
name A_3:
   begin
   v_DL_PDU_received := DL_PDU;
   if require_coupled_ack(v_DL_PDU_received)
         and (v_quiet_mode = OFF) and (v_net_activity = CLEAR) then
      begin
      construct_coupled_ack(v_coupled_ack, v_my_address, v_DL_PDU_received);
         { could be TEST, URR or URNR response.  In the case of URR/URNR
           response, station_busy variable is checked so that correct response
           could be constructed }
      construct_transmission(v_transmission_to_be_sent, v_transmission_length,
         v_coupled_ack, v_FEC_TDC_scrambling, v_multidwell);
      output ip_PSAP.PL_Unitdata_Req(v_transmission_to_be_sent, v_transmission_length,
         v_FEC_TDC_scrambling, v_multidwell, COUPLED);
      end;

   if not is_coupled_ack(v_DL_PDU_received) then
      v_message_id := NULL_ID; { Don't wait for coupled ack anymore }

   if is_for_me(v_DL_PDU_received, v_my_address) then
      begin
      v_DL_PDU_type := get_typeSAP(v_DL_PDU_received);
      if (v_DL_PDU_type = TYPE1ACK) or (v_DL_PDU_type = TYPE1NOACK) then
         begin
         output ip_type1SAP_lower_mux.station_message(
            v_message_id, v_DL_PDU_received, topology_update_id);
         if URNR_message(v_DL_PDU_received) then
            output ip_type2SAP_lower_mux.station_message(
               v_DL_PDU_received, topology_update_id);
         end
      else if (v_DL_PDU_type = TYPE2) then
         begin
         output ip_type2SAP_lower_mux.station_message(
            v_DL_PDU_received, topology_update_id);
         if is_RNRC(v_DL_PDU_received) = TRUE then
            begin
            purge_station_message_table(v_DL_PDU_received);
            end;
         end
      end
   end;

{ Station has buffered message to be sent }
from ACTIVE to WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION
provided ((not station_message_table_empty(v_station_message_table))
   and (v_quiet_mode = OFF) and (v_net_activity = CLEAR))
priority MEDIUM
name A_6:
   begin
   construct_and_concatenate_DL_PDU_of_highest_priority(v_transmission_to_be_sent,
      v_transmission_length, v_need_coupled_ack, v_message_id, v_from_component,
      v_no_of_expected_acks,v_station_message_table, v_FEC_TDC_scrambling,v_multidwell);
   output ip_PSAP.PL_Unitdata_Req(v_transmission_to_be_sent, v_transmission_length,
      v_FEC_TDC_scrambling, v_multidwell, NOT_COUPLED);
   record_type2_transmission_timer_information(v_transmission_type2_timer_list);
   end;

{ Wait for Physical Layer transimission until transmit is COMPLETE, IDLE OR ABORTED }
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to ACTIVE
when ip_PSAP.PL_Status_Ind
provided (transmission_status = TRANSMIT_COMPLETE_OR_IDLE)
   or (transmission_status = TRANSMIT_ABORTED)
priority MEDIUM
name A_7:
   begin
   if v_need_coupled_ack = TRUE then
      output ip_ack_timer.start_ack_timer(
         v_message_id, v_from_component, v_no_of_expected_acks)
   else
      begin
      { According to the timer list recorded when the last transmission was sent, start
        all type 2 timers in the list now.  The timers could be Ack, P, REJ timers, and
        could be for > 1 destinations }
      all i: 0..TRANSMISSION_TIMER_LIST_SIZE do
         all j: type_station_number_range do
            begin
            if (v_transmission_type2_timer_list[i,j].ack2_timer_needed = 1) then
               output ip_ack2_timer[j].start_ack_timer;
            if (v_transmission_type2_timer_list[i,j].P_timer_needed = 1) then
               output ip_P_timer[j].start_P_timer;
            if (v_transmission_type2_timer_list[i,j].REJ_timer_needed = 1) then
               output ip_REJ_timer[j].start_REJ_timer;
            end;
      end;
   v_net_activity := net_activity;
   end;

{ Wait for Physical Layer transmission if TRANSMIT_IN_PROCESS }
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same
when ip_PSAP.PL_Status_Ind
provided transmission_status = TRANSMIT_IN_PROCESS
priority MEDIUM
name A_8:
   begin
   v_net_activity := net_activity;
   end;

{ TYPE2SAP Component provide a new message }
from ACTIVE to same
when ip_type2SAP_lower_mux.type2SAP_message
priority MEDIUM
name A_9:
   begin
   buffer_message_in_station_message_table(
      v_station_message_table, NULL_ID, message, FROM_TYPE2SAP, precedence,
      topology_update_id);
   end;
{ End of Active Phase Transitions }


{**********************}
{  Common Transitions  }
{**********************}
{ Receive PL_Unitdata_Ind from Physical Layer; enqueue valid DL_PDU(s)
  destined for this station }
from ALL_STATES to same
when ip_PSAP.PL_Unitdata_Ind
priority MEDIUM
name C_1:
   begin
```

```
   v_n_th := 1;
   while more_DL_PDU_in_transmission(data, data_length, v_n_th) do
      begin
      v_DL_PDU_retrieved := get_DL_PDU_from_transmission(
         data, data_length, v_n_th, v_FEC_TDC_scrambling, v_multidwell);
      v_topology_update_id_received :=
         get_topology_update_id_from_transmission(data);
      v_n_th := v_n_th + 1;
      if (DL_PDU_valid(v_DL_PDU_retrieved) and
         is_for_me(v_DL_PDU_received,v_my_address))
         then output ip_transmission.DL_PDU_form_PL_Unitdata_Ind(
                v_DL_PDU_received, v_topology_update_id_received);
      end;
   end;

{ Set v_net_activity according to PL_Status_Ind from Physical Layer if
  not in WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION }
from JUST_ACTIVE to same
when ip_PSAP.PL_Status_Ind
priority MEDIUM
name C_4:
   begin
   v_net_activity := net_activity;
   end;

{ Operator ask to change quiet mode }
from ALL_STATES to same
when ip_OSAP.OP_quiet_mode_Req
priority HIGH
name C_5:
   begin
   v_quiet_mode := quiet_mode;
   end;

{ Ignore all other messages from Physical Layer execpt those already specified.
  Note: transition priority is LOW }
from JUST_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same
when ip_DL_PDU.DL_PDU_form_PL_Unitdata_Ind
priority LOW
name C_11:
   begin
   end;

{ Forward DL_Unitdata_Req from NL to appropriate SAP_component}
from ALL_STATES to same
when ip_LSAP.DL_Unitdata_Req
priority HIGH
name C_12:
   begin
   { check QOS information (delay, throughput, reliability) in DL_Unitdata_Req
     to determine which type service is required. From that, determine which
     type of service is to be used (based on what is available). Forward
     request to appropriate SAP component. }

   v_service := service_type_to_be_used(delay_req, throughput, reliability);
   case v_service of
      TYPE1ACK:
          output ip_type1SAP_upper_mux.DL_Unitdata_Req(
             DL_Unitdata_id, dest_addr_list, src_addr,
             topology_update_id, precedence, NOACK,
             data, data_length);
      TYPE1NOACK:
          output ip_type1SAP_upper_mux.DL_Unitdata_Req(
             DL_Unitdata_id, dest_addr_list, src_addr,
             topology_update_id, precedence, YESACK,
             data, data_length);
      TYPE2:
          output ip_type2SAP_upper_mux.DL_Unitdata_Req(
             DL_Unitdata_id, dest_addr_list, src_addr,
             topology_update_id, precedence,
             data, data_length);
      end; {case}
   end;

{ Forward DL_Unitdata_Ind from type1SAP_component to NL}
from ALL_STATES to same
when ip_type1SAP_upper_mux.DL_Unitdata_Ind
priority HIGH
name C_13:
   begin
   output ip_LSAP.DL_Unitdata_Ind(
      dest_addr_list, src_addr, topology_update_id, data, data_length);
   end;

{ Forward DL_Status_Ind from type1SAP_component to NL}
from ALL_STATES to same
when ip_type1SAP_upper_mux.DL_Status_Ind
priority HIGH
name C_14:
   begin
   output ip_LSAP.DL_Status_Ind(
      DL_Unitdata_id, ack_failure, address_list,
      type2_connection_status);
   end;

{ Forward DL_Unitdata_Ind from type2SAP_component to NL}
from ALL_STATES to same
when ip_type2SAP_upper_mux.DL_Unitdata_Ind
priority HIGH
name C_15:
   begin
   output ip_LSAP.DL_Unitdata_Ind(
      dest_addr_list, src_addr, topology_update_id, data, data_length);
   end;

{ Forward DL_Status_Ind from type2SAP_component to NL}
from ALL_STATES to same
when ip_type2SAP_upper_mux.DL_Status_Ind
priority HIGH
name C_16:
   begin
   output ip_LSAP.DL_Status_Ind(
      DL_Unitdata_id, ack_failure, address_list,
      type2_connection_status);
   end;

end;  { b_station_component }

{*****************************************************}
{             Main Specification body                 }
{*****************************************************}
modvar
   NL               : array [0..NS_LESS_1] of m_NL;
   station_component: array [0..NS_LESS_1] of m_station_component;
   OPERATOR         : array [0..NS_LESS_1] of m_OPERATOR;
   PL               : array [0..NS_LESS_1] of m_PL;

initialize
   begin
   all i : 0..NS_LESS_1 do
```

```
      begin
      init NL[i] with b_NL;
      init station_component[i] with b_station_component;
      init OPERATOR[i] with b_OPERATOR;
      init PL[i] with b_PL;

      connect NL[i].ip_datalink to station_component[i].ip_LSAP;
      connect station_component[i].ip_PSAP to PL[i].ip_datalink;
      connect OPERATOR[i].ip_datalink to station_component[i].ip_OSAP;
      connect OPERATOR[i].ip_type1SAP to station_component[i].ip_OPERATOR_type1SAP;
      connect OPERATOR[i].ip_type2SAP to station_component[i].ip_OPERATOR_type2SAP;
      end;
   end;

end.  { end of entire specification }
```