# EIBA Handbook Series

## Release 3.0

# Volume 3:
# System Specifications

### Part 3: Medium Independent Layers

*Chapter 4: Transport Layer*

26.03.1999

# Table of Contents

# 1. Overview: Communication Relationships

The transport layer (layer-4) provides a reliable data transmission over communication relationships. Communication relationships are logical channels that connect users of layer-4 with each other. Layer-4 provides four different types of communication relationships:

1. one-to-many connection-less (multicast)
2. one-to-all connection-less (broadcast)
3. one-to-one connection-less
4. one-to-one connection-oriented

Communication relationships are identified by a local communication relationship identifier (cr_id) which shall be unique for all communication relationships of this EIB end device. The layer-4 converts the cr_id into the destination_address and vice versa using a communication relationship list. Every communication relationship type provides specific layer-4 services.

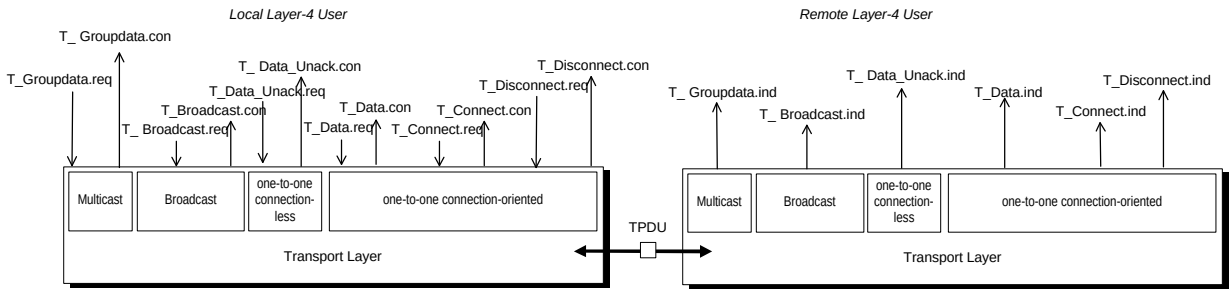


**Fig. 3/3/4-1: Interactivity of the Transport Layer**

Note: In a following update of these EIBA Handbook Series, a one-to-domain communication relationship will be introduced. Specifically on open media it connects one device with all other devices sharing the same Domain Address.

## 1.1 One-to-Many Connection-less (Multicast) Communication Relationship

A multicast communication relationship connects group-objects that belong to the same group (see Chapter 3/3/7 “Application Layer” paragraph "Services on Multicast Communication Relationships"). Group-objects may be distributed to a number of EIB end devices. Each EIB end device may be a transmitter. More than one group-object may exist in an EIB end device. The group-objects of an EIB end device may belong to the same or to different groups.

The following layer-4 service can only be used on a multicast communication relationship:

– T_Groupdata

## 1.2 One-to-All Connection-less (Broadcast) Communication Relationship

The broadcast communication relationship connects a single EIB end device with all communication partners. The single EIB end device is always a transmitter, the communication partners are always receiver.

The following layer-4 service can only be used on a broadcast communication relationship:

- T_Broadcast.

## 1.3 One-to-One Connection-less Communication Relationship

Every EIB end device has a one-to-one connection-less communication relationship with every other EIB end device. A one-to-one connection-less communication relationships shall not be used if the connection-oriented communication relationship is established to the same partner at the same time. The following layer-4 service can only be used on a one-to-one connection-less communication relationship:

- T_Data_Unack.

## 1.4 One-to-One Connection-oriented Communication Relationship

An EIB end device only has a single one-to-one connection-oriented communication relationship. The following layer-4 services can only be used on a connection-oriented communication relationship:

- T_Connect
- T_Data
- T_Disconnect

The user of this type shall establish the connection before it can be used. The user may release the connection if it isn't needed any more. The layer-4 provides a supervision of the connection with a connection-time-out-timer. If the timer expires or if an unrecoverable error occurs, the layer-4 will release the connection immediately. Layer-4 also provides a reliable end-to-end transmission over bridges and routers on the connection-oriented communication relationship. T_Data services are repeated up to three times if the T_Data.req is not acknowledged from the remote layer-4 entity within an acknowledgment-time-out-time. Repetitions of T_Data services are detected using a sequence number. Parallel services are not allowed on a connection-oriented communication relationship. The connection-oriented communication relationship is processed according to the layer-4 state machine described in paragraph 4.

# 2. TPDU

The TPDU is shown in the following figure (Fig. 3/3/4-2).

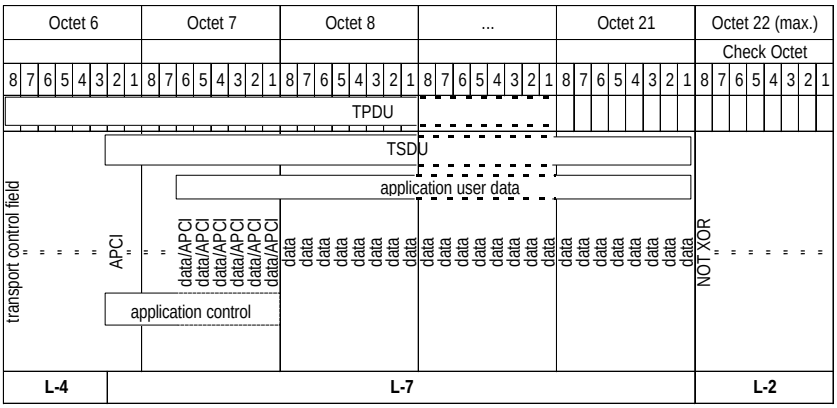


**Fig. 3/3/4-2: Format of the TPDU**

The TPDU corresponds to the NPDU, but reduced by the network control field. The transport control field is encoded and decoded by layer-4 and contains the layer-4 service codes and the sequence_number:

| Octet 5 | | Octet 6 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | transport control field | | | | | | | |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | |
| destination_address_ flag (DA | | | | | | | | | |
| 1 | | 0 | 0 | 0 | 0 | 0 | 0 | | T_BROADCAST_DATA_REQ_PDU (Destination_Address=( |
| 1 | | 0 | 0 | 0 | 0 | 0 | 0 | | T_GROUPDATA_REQ_PDU (Destination_Address<>0) |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | | T_DATA_UNACK_REQ_PDU |
| 0 | | 0 | 1 | SeqNo | SeqNo | SeqNo | SeqNo | | T_DATA_REQ_PDU |
| 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | T_CONNECT_REQ_PDU |
| 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | T_DISCONNECT_REQ_PDU |
| 0 | | 1 | 1 | SeqNo | SeqNo | SeqNo | SeqNo | 1 | 0 T_DATA_ACK_PDU |
| 0 | | 1 | 1 | SeqNo | SeqNo | SeqNo | SeqNo | 1 | 1 T_DATA_NAK_PDU |

**Fig. 3/3/4-3: Transport Control Field**

Note: The encoding 0x83 is reserved and shall not be used for future extensions.

# 3. Transport Layer Services

All the layer-4 services provide a confirmation to the requester (user of layer-4). The confirmation of the T_Data service indicates that the remote layer-4 entity did acknowledge the T_Data.req. The confirmation of the other layer-4 services is caused by the local confirmation of layer-2.

The user of layer-4 shall not request a service primitive before the preceding request is confirmed by layer-4, i.e. no parallel services are allowed.

## 3.1 T_Groupdata Service

The T_Groupdata service is applied by the user of layer-4, to transmit a TSDU over a multicast communication relationship to one or more remote partners. The T_Groupdata service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 prepares a TSDU for the remote user. The local user of layer-4 applies the T_Groupdata.req primitive to pass the TSDU to the local layer-4. The destination is defined by the cr_id.

The local layer-4 accepts the service request, maps the cr_id to the destination group address using a communication reference list, maps the argument class to the corresponding argument of the N_Groupdata.req primitive, encodes the TSDU to the NSDU and passes it with a N_Groupdata.req to the local layer-3.

The remote layer-4 maps a N_Groupdata.ind primitive to a T_Groupdata.ind primitive. The remote layer-4 maps the destination_address to the cr_id using a communication reference list. The argument NSDU is mapped to the argument TSDU, the argument class is mapped to the corresponding argument class of the T_Groupdata.ind primitive.

Prior to passing a T_Groupdata.con primitive to the local user, the local layer-4 needs a N_Groupdata.con from the local layer-3. If the confirmation is positive (n_status = OK), the local layer-4 passes a positive T_Groupdata.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_Groupdata.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Groupdata.req didn’t succeed.

```
T_Groupdata.req(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4
```

```
T_Groupdata.con(cr_id, t_status)
 cr_id:                          identifier of the communication relationship
 t_status:  OK;                  T_Groupdata sent successfully with
                                 N_Groupdata service
            not_ok;              transmission of the associated N_Groupdata
                                 request frame didn't succeed

T_Groupdata.ind(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data that has been
                                 transferred by layer-4
```

## 3.2 T_Broadcast Service

The T_Broadcast service is applied by the user of layer-4, to transmit a TSDU over a connection-less communication relationship to all remote partners. The T_Broadcast service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 prepares a TSDU for all the remote users of layer-4. The local user of layer-4 applies the T_Broadcast.req primitive to pass the TSDU to the local layer-4. The local layer-4 accepts the service request and passes it with a N_Broadcast.req to the local layer-3.

The local layer-4 encodes the TSDU to the NSDU and maps the argument class to the corresponding argument class of the N_Broadcast.req primitive.

The remote layer-4 maps a N_Broadcast.ind primitive to a T_Broadcast.ind primitive. The argument NSDU is mapped to the argument TSDU, the argument class is mapped to the corresponding argument class of the T_Broadcast.ind primitive.

Prior to passing a T_Broadcast.con primitive to the local user, the local layer-4 needs a N_Broadcast.con from the local layer-3. If the confirmation is positive (n_status = OK), the local layer-4 passes a positive T_Broadcast.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_ Broadcast.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Broadcast.req didn't succeed.

```
T_Broadcast.req(class, tsdu)
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Broadcast.con(t_status)
 t_status:  OK;                  T_Broadcast sent successfully with
                                 N_Broadcast service
            not_ok;              transmission of the associated N_Broadcast
                                 request frame didn't succeed

T_Broadcast.ind(source_address, class, tsdu)
 source_address:                 physical address of the EIB end device that
                                 requested the T_Broadcast service
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data that has been
                                 transferred by layer-4
```

Note: A following update of these EIBA Handbook Series will describe a T_SystemBroadcast service to be used on a One-to-Domain communication relationship.

## 3.3 T_Data_Unack

The T_Data_Unack service is applied by the user of layer-4, to transmit a TSDU over a connection-less one-to-one communication relationship to exactly one remote partner. The T_Data_Unack service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 prepares a TSDU for the remote user. The local user of layer-4 applies the T_Data_Unack.req primitive to pass the TSDU to the local layer-4. The destination is defined by the cr_id wich is the physical address of the remote partner.

The local layer-4 encodes the TSDU to the NSDU and maps the arguments cr_id (physical_address) and class to the corresponding arguments destination_address and class of the N_Data.req primitive.

The remote layer-4 maps a N_Data.ind primitive containing a T_DATA_UNACK-PDU to a T_Data_Unack.ind primitive. The remote layer-4 maps the destination_address to the cr_id (physical address). The argument NSDU is mapped to the argument TSDU, the argument class is mapped to the corresponding argument class of the T_Data_Unack.ind primitive.

Prior to passing a T_Data_Unack.con primitive to the local user, the local layer-4 needs a N_Data.con from the local layer-3. If the confirmation is positive (n_status = OK), the local layer-4 passes a positive T_Data_Unack.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_Data_Unack.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Data.req didn't succeed.

```
T_Data_Unack.req(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Data_Unack.con(cr_id, n_status)
 cr_id:                          identifier of the communication relationship
 n_status:  OK;                  T_Data_Unack sent successfully with N_Data
                                 service
            not_ok;              transmission of the associated N_Data
                                 request frame didn't succeed

T_Data_Unack.ind(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data that has been
                                 transferred by layer-4
```

## 3.4 T_Connect Service

The T_Connect service is applied by the user of layer-4, to establish a transport connection on a connection-oriented communication relationship. The T_Connect primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in paragraph 4. The local layer-4 accepts the service request only if the connection is not established (state CLOSED) and tries to send the T_CONNECT_REQ_PDU to the remote layer-4 with a N_Data.req. The destination address shall be a physical address. The T_Connect service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation, caused by the L_Data.con of layer-2.

The cr_id is always ´0´ for connection-oriented communication relationships. Only one connection-oriented communication relationship may be established in a device at a time.

| | |
|---|---|
| `T_Connect.req(address)` | |
| `address:` | `physical address of the device where the transport connection shall be established. The class parameter must be set to system priority by the layer-4.` |
| `T_Connect.con(cr_id, t_status)` | `connection is established, local initiator` |
| `cr_id:` | `identifier (0) of the communication relationship` |
| `t_status: ok:` | `T_Connect sent successfully with N_Data service` |
| `not_ok:` | `transmission of the associated N_Data-request frame did not succeed` |
| `T_Connect.ind(cr_id)` | `connection established, remote initiator` |
| `cr_id:` | `identifier (0) of the communication relationship` |

## 3.5 T_Disconnect Service

The T_Disconnect service is applied by the user of layer-4, to release a transport connection on a connection-oriented communication relationship. The T_Disconnect primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in paragraph 4. The T_Disconnect service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The T_Disconnect.ind primitive may also be caused by the layer-4 entity in order to indicate a protocol error.

The cr_id is always ´0´ for connection-oriented communication relationships. Only one connection-oriented communication relationship may be established in a device at a time.

| | |
|---|---|
| `T_Disconnect.req(cr_id)` | |
| `cr_id:` | `identifier of the communication relationship to be released` |
| | `Note: The class parameter must be set to system priority by the layer-4.` |
| `T_Disconnect.con(cr_id)` | `Connection released, state = CLOSED` |

```
T_Disconnect.ind(cr_id)          Connection released, state = CLOSED, caused
                                 by an error detected by the local layer-4 or
                                 a T_DISCONNECT_REQ_PDU was received
```

## 3.6 T_Data Service

The T_Data service is applied by the user of layer-4, to transmit a TSDU over a transport connection on a connection-oriented communication relationship to a remote partner. The T_Data service is acknowledged with a T_DATA_ACK_PDU by the remote layer-4 entity. The T_ Data primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in 4.

The local user of layer-4 prepares a TSDU for the remote user. The local user of layer-4 applies the T_Data.req primitive to pass the TSDU to the local layer-4. The local layer-4 accepts the service request only if the connection is established (state OPEN_IDLE) and tries to send the TSDU to the remote layer-4 with a N_Data.req. The destination address shall be a physical address. The local layer-4 passes a T_Data.con primitive to the local user that indicates either a correct data transfer or it passes a T_Disconnect.ind primitive to the local user that indicates an erroneous data transfer.

Prior to passing the confirmation to the local user, the local layer-4 needs an acknowledgment from the remote layer-4. If the acknowledgment is a positive acknowledgment (T_DATA_ACK_PDU), the local layer-4 passes a T_Data.con to the local user. If the acknowledgment is a negative acknowledgment (T_DATA_NAK_PDU), the local layer-4 passes a T_Disconnect.ind primitive to the local user indicating that the connection is released (state = CLOSED) caused by an error.

The remote layer-4 will only accept the N_Data.ind with the T_DATA_REQ_PDU received, if the connection is established, i.e. in the states OPEN_IDLE, OPEN_WAIT_FOR_DATA_ACK. Therefore mutual T_Data services are allowed on a connection-oriented communication relationship.

The local layer-4 repeats the transmission of the T_DATA_REQ_PDU up to 3 times with an acknowledgment time-out time of 3 s. If it fails, the local layer-4 passes a T_Disconnect.ind primitive to the local user indicating that the connection is released (state = CLOSED).

The cr_id is always ´0´ for connection-oriented communication relationships. Only one connection-oriented communication relationship may be established at a time in a device.

```
T_Data.req(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Data.con(cr_id)                transmission successful
 cr_id:                          identifier of the communication relationship

T_Data.ind(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data that has been
                                 transferred by layer-4
```

# 4. State Machine of Connection-Oriented Communication Relationship

The layer-4 state machine processes the services T_Connect, T_Disconnect and T_Data. Other layer-4 services are directly mapped as described for each individual service independent from the actual state of the layer-4 state machine. Invalid PDUs are ignored.

- Events caused by the user of layer-4:

```
T_Connect.req from user
     \-
T_Disconnect.req from user
     \-
T_Data.req from user
     \-
```

- Events caused inside layer-4:

```
Connection_Timeout
     \-
Acknowledgement_Timeout
     \-
```

- Events caused by layer-3:

```
N_Data.ind from layer-3
     \source_address<>connection_address
N_Data.ind from layer-3
     \nsdu=T_CONNECT_REQ_PDU
N_Data.con from layer-3
     \source_address=connection_address & nsdu=T_CONNECT_REQ_PDU
N_Data.ind from layer-3
     \source_address=connection_address & nsdu=T_DISCONNECT_REQ_PDU
N_Data.ind from layer-3
     \source_address=connection_address & nsdu=T_DATA_REQ_PDU
N_Data.ind from layer-3
     \source_address=connection_address & nsdu=T_DATA_ACK_PDU
N_Data.ind from layer-3
     \source_address=connection_address & nsdu=T_DATA_NAK_PDU
N_Data.con from layer-3
     \-
```

- Local variables of layer-4:

| | |
|---|---|
| connection_address | used to store the actual physical address of the partner |
| SeqNoSend | binary 4 bit value, used to handle the sequence number of the T_DATA_XXX-PDU |
| SeqNoRcv | binary 4 bit value, used to handle the sequence number of the T_DATA_XXX-PDU |

| | |
|---|---|
| connection_timeout_timer | time interval of 6 s;<br>starts with transition CLOSED→CONNECTING;<br>stops with transition →CLOSED;<br><br>restarts if N_Data.req is applied in the state machine or N_Data.ind received with source_address=connection_address |
| acknowledgment_timeout_timer | time interval of 3 s;<br>starts with transition OPEN_IDLE→OPEN_WAIT_FOR_T_DATA_ACK;<br><br>stops if N_DATA.ind & source_address=connection_address & nsdu=T_DATA_ACK-PDU & SeqNo_of_PDU=SeqNoRcv received or with transition into →CLOSED |
| rep_count | used to count the number of T_DATA_REQ repetitions |

The user of layer-4 always gets a confirmation for a request. When reading the state machine, keep in mind that the user cannot request a second service primitive before getting the confirmation to the preceding primitive, i.e. no parallel services are allowed.

The event N_Data.con after the transmission of a N_Data.req is not described in all of the states. In states where this event is not mentioned, a subordinate state is active where the state machine is waiting for either a time-out or the N_Data.con. As soon as the N_Data.con or the time-out appears, the state machine switches back to the associated superordinate state.

```
CURRENT STATE                    TRANSITION NAME                    NEXT STATE
  Event
     \Condition
     ⇨ Action

POWER_ON                              1.START                          CLOSED
  -
     \-
     ⇨  SeqNoRcv = 0
        SeqNoSend = 0
        rep_count=0
        stop acknowledgment_timeout_timer

CLOSED                           2.T_CONNECT_REQ                   CONNECTING
  T_Connect.req from User
     \-
     ⇨  connection_address = destination_address
        N_Data.req(address=destination_address,class=system,
                   nsdu=T_CONNECT_REQ_PDU)
        start connection_timeout_timer
```

**CLOSED** **3.T_CONNECT_IND** **OPEN_IDLE**

**N_Data.ind from layer-3**

\nsdu=T_CONNECT_REQ_PDU

⇨ connection_address = source_address
T_Connect.ind to the user
N_Data.req(address=destination_address,class=system,
nsdu=T_CONNECT_CON_PDU)
start connection_timeout_timer

**CONNECTING** **4.N_DATA_CON** **OPEN_IDLE**

**N_Data.con from layer-3**

\source_address=connection_address & nsdu=T_CONNECT_IND_PDU

⇨ T_Connect.con to the user
restart connection_timeout_timer

**CONNECTING** **5.CONNECTION-TIME-OUT** **CLOSED**

**Connection_Timeout**

\-

⇨ T_Disconnect.ind to the user

N_Data.req(address=connection_address,class=system,
nsdu=T_DISCONNECT_REQ_PDU)

**OPEN_???** **6.T_DATA_IND_OK** **OPEN_???**

**N_Data.ind from layer-3**

\source_address=connection_address & nsdu=T_DATA_REQ_PDU &
SeqNo_of_PDU=SeqNoRcv

⇨ Copy SeqNoRcv to T_DATA_ACK_PDU
N_Data.req(address=connection_address,class=system,
nsdu=T_DATA_ACK_PDU)
Increment SeqNoRcv
T_Data.ind to the user
restart connection_timeout_timer

**OPEN_???** **7.T_DATA_IND_REPEATED** **OPEN_???**

**N_Data.ind from layer-3**

\source_address=connection_address & nsdu=T_DATA_REQ_PDU &
SeqNo_of_PDU=SeqNoRcv-1

⇨ Copy SeqNo_of_PDU to T_DATA_ACK_PDU
N_Data.req(address=connection_address,class=system,
nsdu=T_DATA_ACK_PDU)
restart connection_timeout_timer

**OPEN_???** **8.T_DATA_IND_NOTOK** **OPEN_???**

**N_Data.ind from layer-3**

\source_address=connection_address & nsdu=T_DATA_REQ_PDU &
SeqNo_of_PDU<>SeqNoRcv or SeqNo_of_PDU<>SeqNoRcv-1

⇨ Copy SeqNo_of_PDU to T_DATA_NAK_PDU
N_Data.req(address=connection_address,class=system,
nsdu=T_DATA_NAK_PDU)
restart connection_timeout_timer

**OPEN_???** **9.T_DISCONNECT_IND** **CLOSED**

**N_Data.ind from layer-3**

\source_address=connection_address & nsdu=T_DISCONNECT_REQ_PDU

⇨ stop acknowledgement_timeout_timer
T_Disconnect.ind to the user
stop connection_timeout_timer

**OPEN_IDLE** **10.CONNECTION_TIMEOUT** **CLOSED**
**Connection_Timeout**
\-
⇨ N_Data.req(address=connection_address,class=system,
nsdu=T_DISCONNECT_REQ_PDU)
T_Disconnect.ind to the user

**OPEN_IDLE** **11.T_DATA_REQ** **OPEN_WAIT_FOR_T_DATA_ACK**
**T_Data.req**
\-
⇨ copy SeqNoSend to T_DATA_REQ_PDU
start acknowledgement_timeout_timer
N_Data.req(address=connection_address,class,
nsdu=T_DATA_REQ_PDU)
restart connection_timeout_timer

**OPEN_IDLE** **12.T_DISCONNECT_REQ** **CLOSED**
**T_Disconnect.req**
\-
⇨ N_Data.req(address=connection_address,class=system,
nsdu=T_DISCONNECT_REQ_PDU)

**OPEN_WAIT_FOR_T_DATA_ACK** **13.T_DATA_ACK** **OPEN_IDLE**
**N_Data.ind from layer-3**
\source_address=connection_address & nsdu=T_DATA_ACK_PDU &
SeqNo_of_PDU=SeqNoSend
⇨ stop acknowledgement_timeout_timer
Increment SeqNoSend
T_Data.con(t_status=ok) to the user
restart connection_timeout_timer

**OPEN_WAIT_FOR_T_DATA_ACK** **14.T_DATA_ACK_NOTOK** **OPEN_WAIT_FOR_T_DATA_ACK**
**N_Data.ind from layer-3**
\source_address=connection_address & nsdu=T_DATA_ACK_PDU &
SeqNo_of_PDU<>SeqNoSend
⇨ discard T_DATA_ACK_PDU and optional breakdown connection

**OPEN_WAIT_FOR_T_DATA_ACK** **15.ACK-TIMEOUT123** **OPEN_WAIT_FOR_T_DATA_ACK**
**Acknowledgement_Timeout123**
\rep_count < 3
⇨ start acknowledgement_timeout_timer
repeat last N_DATA.req(address=connection_address,class,
nsdu=T_DATA_REQ_PDU)
increment rep_count

**OPEN_???** **16.T_DATA_NAK_NOT_OK** **CLOSED**
**N_Data.ind from layer-3**
\source_address=connection_address & nsdu=T_DATA_NAK_PDU &
SeqNo_of_PDU<>SeqNoSend
⇨stop ackowledgement_timeout_timer
N_Data.req(address=connection_address, class=system,
n_sdu=T_DISCONNECT_REQ_PDU)
T_Disconnect.ind to the user
stop connection_timeout_timer

```
OPEN_WAIT_FOR_T_DATA_ACK       17.T_DATA_NAK_OK     OPEN_WAIT_FOR_T_DATA_ACK
  N_Data.ind from layer-3
    \source_address=connection_address & nsdu=T_DATA_NAK_PDU &
     rep_count < 3 &SeqNo_of_PDU=SeqNoSend
    ⇨  start acknowledgement_timeout_timer
       repeat last N_DATA.req(address=connection_address,class,
                              nsdu=T_DATA_REQ_PDU)
       increment rep_count

OPEN_WAIT_FOR_T_DATA_ACK       18 T_DATA_NAK_OK                      CLOSED
  N_Data.ind from layer-3
    \source_address=connection_address & nsdu=T_DATA_NAK_PDU &
     rep_count = 3 &SeqNo_of_PDU=SeqNoSend
    ⇨  stop acknowledgement_timeout_timer
       rep_count=0
       N_Data.req(address=connection_address,class=system,
                  nsdu=T_DISCONNECT_REQ_PDU)
       T_DISCONNECT_IND to User

OPEN_WAIT_FOR_T_DATA_ACK       19 ACK-TIMEOUT4                       CLOSED
  Acknowledgement_Timeout 4
    \rep_count=3
    ⇨  stop acknowledgement_timeout_timer
       rep_count=0
       N_Data.req(address=connection_address,class=system,
                  nsdu=T_DISCONNECT_REQ_PDU)
       T_DISCONNECT_IND to User
```

Note : Received N_Data.ind PDU's with address <> connection address and nsdu = T_DATA forced a N_Data.req(address=sourceaddrss,class=system, nsdu=T_DISCONNECT_REQ_PDU)

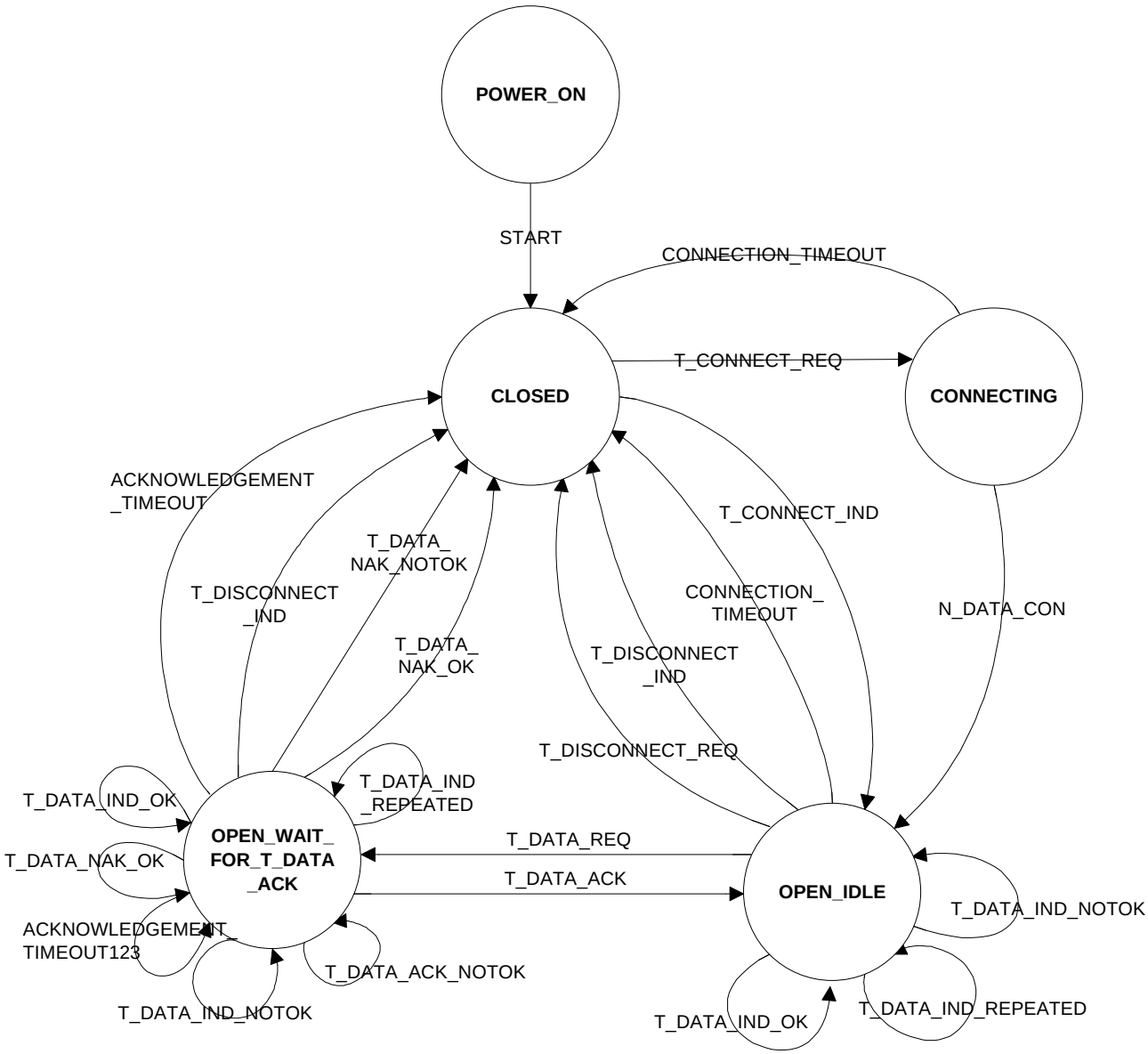


**Fig. 3/3/4-4: State Machine of Layer-4**

# 5. Parameters of Layer-4

The communication relationship list is the only parameter of layer-4. The implementation of this parameter, e.g. table or algorithmic is up to the manufacturer. The latter could be achieved by mapping the address of the communication partner (group address or physical address with DAF flag) to a 13 bit cr_id.

| | |
|---|---|
| Communication relationship list: | addresstable<br>maps cr_id's to destination addresses and vice versa. |
| Connection timeout: | time interval of 6 s;<br>timeout to breakdown a connection |
| Acknowledment timeout: | time interval of 3 s;<br>timer to start a repetion if no acknowledgement was received |
| Repeatcounter: | 3;<br>maximum of T_Data.req repetitions |

## 5.1 Algorithm for the Identifier of a Communication Relationship (cr_id)

The cr_id used in the T_Groupdata primitive shall be the cr_id of the group address in the address table, see Chapter 3/3/2 “Data Link Layer General”.

The cr_id used in the T_Data_Unack primitive shall be the physical_address of the communication partner.

The cr_id used in connection-oriented transport communication shall be always zero.

# 6. Externally Accessible Transport Layer Interface

The transport layer services can be made available to an external user application by switching off layers 7 and 8 and the internal user application. See Chapter 3/6/3 "External Message Interface", clause "Transport Layer EMI" for the transport layer message formats and clause "Layer Access Management" how to switch to the transport layer EMI.