MESSAGE LINK PROTOCOL (MLP)
Functional Specification

by

G.v.Bochmann
University of Montreal
Montreal Canada

F.H.Vogt
Hahn-Meitner-Institut
Berlin-West Germany

Abstract

This document is a specification of the Message Link Protocol (MLP). The MLP creates a transport service between session control entities via transport connections, and some additional session control service. It is based on virtual circuits a la X.25. It is concerned with transport connection establishment and termination, and handles resets and clears to provide a reliable transport service. Furthermore it provides synchronization support related to session establishment, termination and negotiation. The allocation of transmission resources to connections is also considered. It is assumed that the flow control capabilities of the X.25 virtual circuits can be used for transport connection flow control. If, for other transmission media, flow control may not be assumed, an additional function to provide flow control at the transport service level has to be introduced.

1. Introduction

The need for a transport service in computer networks has first been found with packet switched datagram transmission networks in order to provide a reliable message transmission between the communicating processes. It has been argued that no separate transport layer is needed when communication is based on virtual circuits of public data networks. However, certain applications require that the integrity of process to process communications be guaranteed independently of this underlying transmission service. In fact, the data transmission over a virtual circuit may be disrupted by network generated resets and clears, and possibly even by transmission errors and unforeseen malfunctions.

The so called "Message Link Protocol" (MLP) (HERTW77, HERTW78) specified in this paper deals with process to process message transport over dynamically established connections and a simple form of process synchronization. It was designed to operate over a packet switched network providing virtual circuits. It may operate in a hostile environment with network generated resets and clears, and other difficulties.

Two classes of service are distinguished:

a) normal service, where the loss of messages is not excluded

and

b) reliable service, where all network malfunctioms may be recovered by the protocol (as long as the data transmission service does not break completely, and both communicating partners work correctly).

In order to simplify the understanding of the MLP, we distinguish several "sub-layers", each of which has a particular function. In ascending order of hierarchy, they are the following:

- Transmission Resource Allocation (TRA)

Assuming the use of virtual circuit, as provided by public packet switched data networks, this sub-layer handles the allocation of permanent and switched virtual circuits (VC). It establishes agreement between the communicating transport entities about the allocation of the available channels to the active process-to-process connections.

- Connection Establishment and Clearing (CON)

Based on the service provided by the transmission resource allocation, the "connection" sub-layer (CON) is responsible for the establishment and clearing of process to process connections, over which messages may be exchanged by the next higher sub-layer. This involves the end-to-end identification of processes and connections, and the detection of contention. It is considered that this sub-layer consists of the following entities:

a) one connection entity for each connection end point,

b) one connection administration entity for each subsystem identified by a transmission network subscriber address.

For each incoming or outgoing connection establishment request, a connection entity is responsible for the connection establishment and later disconnection. It interacts with the connection administration entity which detects contention and identifies and possibly activates the appropriate partner process for an incoming connection request. Temporary deallocation of the transmission resource (network VC) without loosing the transport connection may be supported. The complementary function of reactivating a released connection may be supported as well, of course. Based on this feature, "break recovery" is proposed to provide recovery after network generated <u>clears</u>.

- Message Transport (MT)

When a connection is established by the sub-layer below, this sub-layer provides reliable transport of messages and interrupts (in case of the normal class of service, this sub-layer is very rudimentary). The retransmission sub-layer detects any irregularities, including <u>resets</u>, of the underlying transmission service. It transmits messages and interrupts, and exchanges delivery confirmations for the communicating processes and retransmission buffer managment.

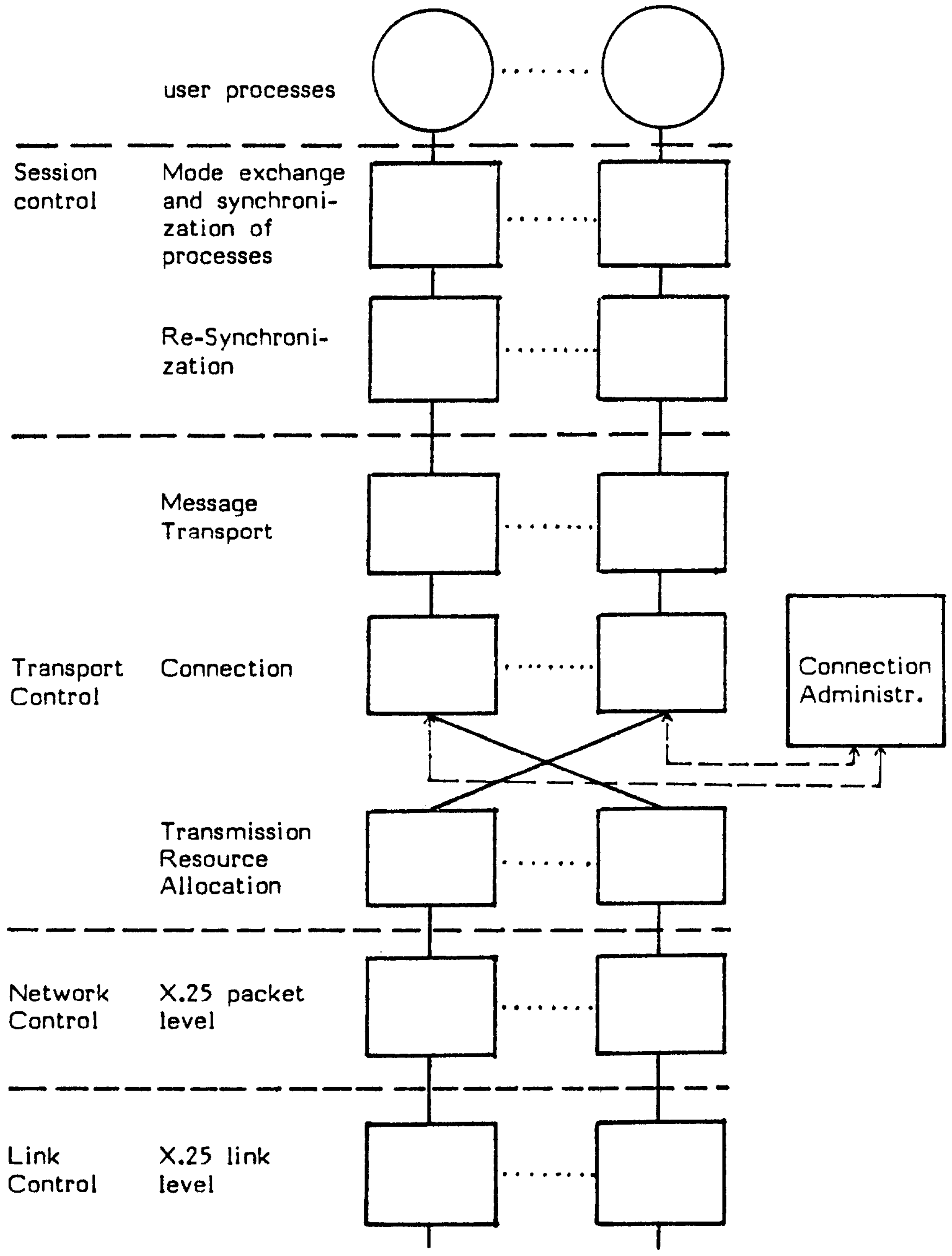
- Re-Synchronization (RS)

This sub-layer provides, in addition to the message transport service provided by the layers below, controlled flushing and non-flushing interrupt synchronization between the communicating processes, as well as data synchronization which operates in line with the transmitted messages.

- Mode Exchange and Synchronization of processes (MS)

A more advanced synchronization mechanism for the use of session control with the additional possibility of nesting sessions is supported by the MLP, based on the synchronization mechanism of the RS sub-layer.

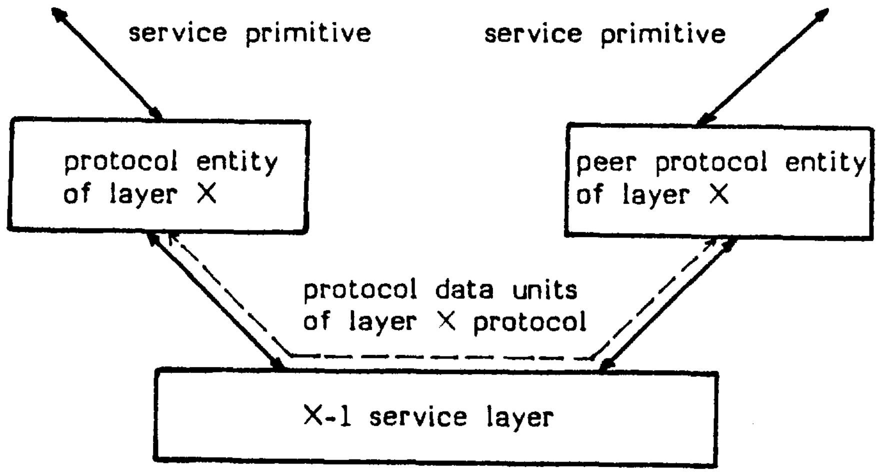
The architecture of a typical implementation of the MLP is indicated in figure 1, showing the different sub-layers and their relation to the standard layered architecture proposed by ISO (see ISO/TC97/SC16/N46).



MLP architecture
fig.1

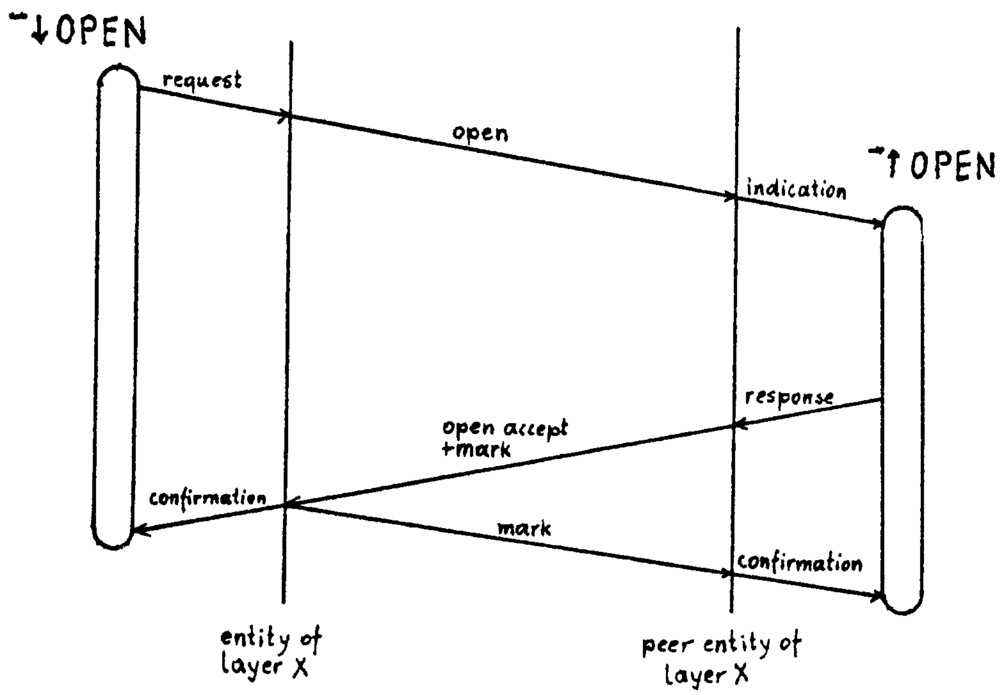
2. Notation and Terminologie

Protocol data units are defined according to (ISO/TC97/SC16/N46). They are generated in a certain layer and exchanged by the service of the underlying layer (see figure 2).


service primitive
service primitive
protocol entity of layer X
peer protocol entity of layer X
protocol data units of layer X protocol
X-1 service layer

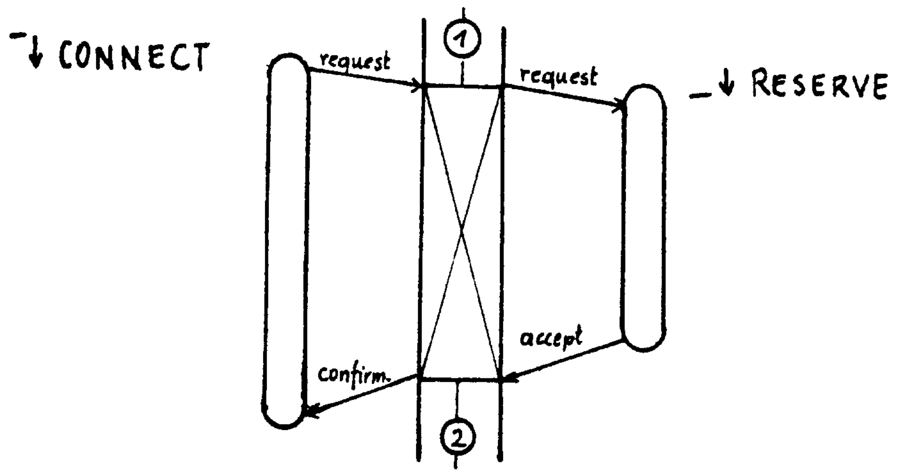
Terminologie
fig.2

The service provided by a layer X is accessible through service primitives (see figure 3). They are written in capital letters. An service primitive may be initiated by the layer below (X) or the layer above (X+1). This is indicated by the symbols " ↑ " and " ↓ ", respectively. The symbols " ¯ " and " _ " are used to indicate that a given service primitive belongs to the interface above and below a given layer, repectively. For example, " ¯ ↓ OPEN" stands for the OPEN primitive (a service of the MS sub-layer), as seen from the MS sub-layer, and initiated by the user layer above.
The execution of a service primitive may involve several interface events which perform the exchange of parameter values between the interacting layers. They are distinguished by names like "request", "confirmation", etc. In the case of the service primitive OPEN, we distinguish two or three events, as shown in figure 3. They are written as ¯ ↓ OPEN(request), ¯ ↓ OPEN(confirmation), ¯ ↑ OPEN(indication), and ¯ ↑ OPEN(response). Open, open accept and mark are the related protocol data units. The coding of protocol data units is not defined in this document. Also the question of concatenation of several protocol data units for efficient transmission over a VC is left for further study.



Service primitives, interface events and protocol data units
fig.3

Different internal states may be distinguished for a given protocol entity. They are indicated by numbers (n). An example is given in figure 4, showing a connection entity executing a CONNECT primitive.



(1) "non-existant" state
(2) "connected" state
⋈ represents an intermediate state in which no activity is performed until the service primitive initiated by the entity is terminated.

fig. 4

The parameters of the service primitives are essential for the logical understanding of their meaning. The values of the parameters are exchanged between the interacting layers by the interface events. The symbol "→" written before the name of a parameter means that its value is provided by the layer which initiates the service primitive. The symbol "←" means that the parameter value is provided by the responding layer. The parameters marked with "→" or "←" usually have an end-to-end significance between the communicating entities (see for example figure 5). This is not the case for a parameter marked with " ↑ ". For example, the value (either OK or broken) of the status parameter of figure 5 is provided by the layer below the interface, and has no end-to-end significance. The value broken at a local interface implies that the end-to-end parameters, received at the interface, do not necessarily have correct values, and that the peer entity does not necessarily execute the corresponding service primitive.

```
- ↓ CONNECT  (  →  local generic process communication name,
                →  local process identifier,
                →  distant subscriber address,
                →  distant generic proc. communication name,
                ←  distant process identifier,
                →  service class,
                ←  response: (accepted,non-accepted,collision)
                ↑  status,
                →  break recovery: boolean,
                →  protocol identifier)
```

Parameter example
fig.5

3. Transport Control Servive

3.1 Underlying Transmission Service

The transport service may be based on different underlying transmission services. In the case of switched circuits or leased lines an HDLC-like interprocessor protocol should be added to provide flow control and error control. Multiplexing of several transmission channels over a given circuit may also be considered. For a datagram based transmission service, flow control has to be added. This may be provided, for example, by a credit scheme.

For error control, the retransmission function of the MLP (reliable service class) may be used. This would imply that the retransmission is performed end-to-end between the transport service entities.

For this document we assume that the basic transmission service, is given by Virtual Circuits (VC) as provided through an X.25 interface (CCITT78).

The function of the transmission service is to transfer data via virtual circuits (or logical channels) which are locally identified by logical channel identifiers (LCI). For switched virtual circuits (or virtual calls) logical channel identifiers are assigned during the call set-up phase. For permanent virtual circuits logical channel identifieres are assigned in agreement with the Administration at the time of subscription to the service.

Since the transmission service is not exactly defined by the interface standard, we make the following relatively weak assumptions which (hopefully) are satisfied for all networks and also for inter-network virtual circuits:

a) During a circuit reset, data and interrupt packets in transit may be discarded.

b) Exept for (a), messages in form of "user sequences" are sequentially transmitted without errors.

c) Interrupts are never duplicated.

d) The interrupt confirmation does not necessarily have end-to-end significance.

e) An interrupt packet usually travels faster, but may travel slower than data packets.

f) To each reset operation at one end-point of the circuit, there is a corresponding reset operation at the other end, but a subscriber initiated reset operation at one end may correspond to a network generated reset at the other end.

The data transmission service over virtual circuit is characterized by the following service primitives*) (initiation and clearing only for switched circuits):

```
↑ or ↓  INITIATE VIRTUAL CIRCUIT (   →  LCI,
                                     →  distant subscriber,
                                     ←  accepted: boolean)

↑ or ↓  CLEAR            (   →  LCI)
```

*) For our purposes, we ignore such information as facility parameters, call user data, clearing cause, etc.

```
↑ or ↓  USER SEQUENCE   ( →  LCI,
                          →  user sequence)


↑ or ↓  INTERRUPT       ( →  LCI,
                          →  code: 8 bits)


↑ or ↓  RESET           ( →  LCI)
```

Note, for example, that the service primitive ↑ INTERRUPT represents the receipt of an interrupt (including the sending of the interrupt confirmation packet), and ↓ INTERRUPT the sending of an interrupt (including the receipt of the interrupt confirmation packet), as explained in section 2.

3.2 Transmission Resource Allocation (TRA)

Introduction

This sub-layer provides allocation and deallocation of transmission channels (VC's) between two transport entities for transport connections to be established or resumed. The protocol is based on the exchange of allocate, deallocate, confirm and release protocol data units which are sent over the channel to be allocated or deallocated. The sub-layer also provides a mechanism to recover from packet loss during the allocation and deallocation phase. In the case of virtual circuits, packet loss may occur due to network generated resets. This is accomplished by retransmission of protocol data units after resets or time-outs. The mechanism is used as a basis for the CON sub-layer.

Interface

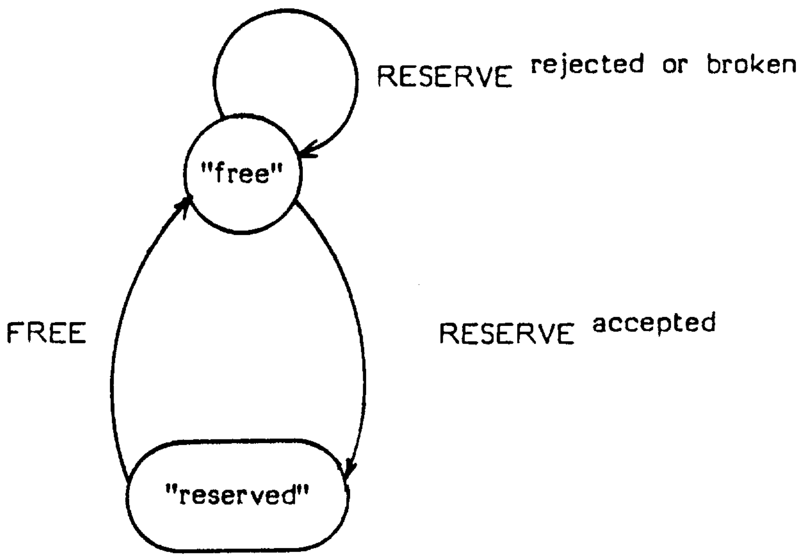
The service interface consists of the following primitives, which are locally associated with a given transmission channel (LCI):

```
↑ or ↓  RESERVE ( →  for-whom,
                  ←  accepted: boolean,
                  ←  reason,
                  ↑  status)


↑ or ↓  FREE    ( →  why,
                  ↑  status)
```

where the for-whom parameter specifies whether the channel reservation is made for a RESUME or CONNECT primitive and which connection is involved. The responding side may accept or reject the reservation, giving the reason for a possible rejection. The TRA layer may also reject a reservation. This may only happen in the case of a collision (see below). If the transmission channel breaks (i.e. the network clears the circuit, due to congestion) when (locally) an interface primitive is in progress, the primitive is terminated and the status parameter has the value broken. The why parameter for the FREE primitive indicates whether the primitive is invoked by a RELEASE or a DISCONNECT.

The following state diagram (see figure 6) shows in which order the service primitives may be executed.



Order of TRA interface primitives
fig.6

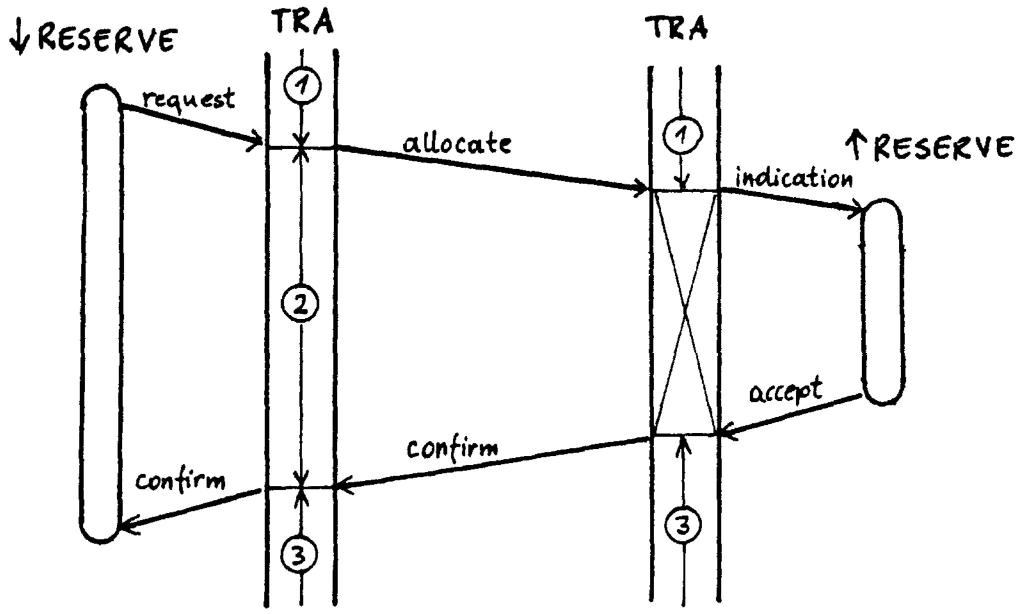
Protocol

The operation of the TRA protocol is summarized in the table below (figure 7). It contains, for each local state of the TRA entity, the operations that are initiated by service requests, received protocol data units, time-outs and other signals from the underlying transmission channel. We note that the p.d.u. retransmitted after time-out or network generated reset must be coded such that they can be distinguished, by the receiving entity, from initial transmissions.

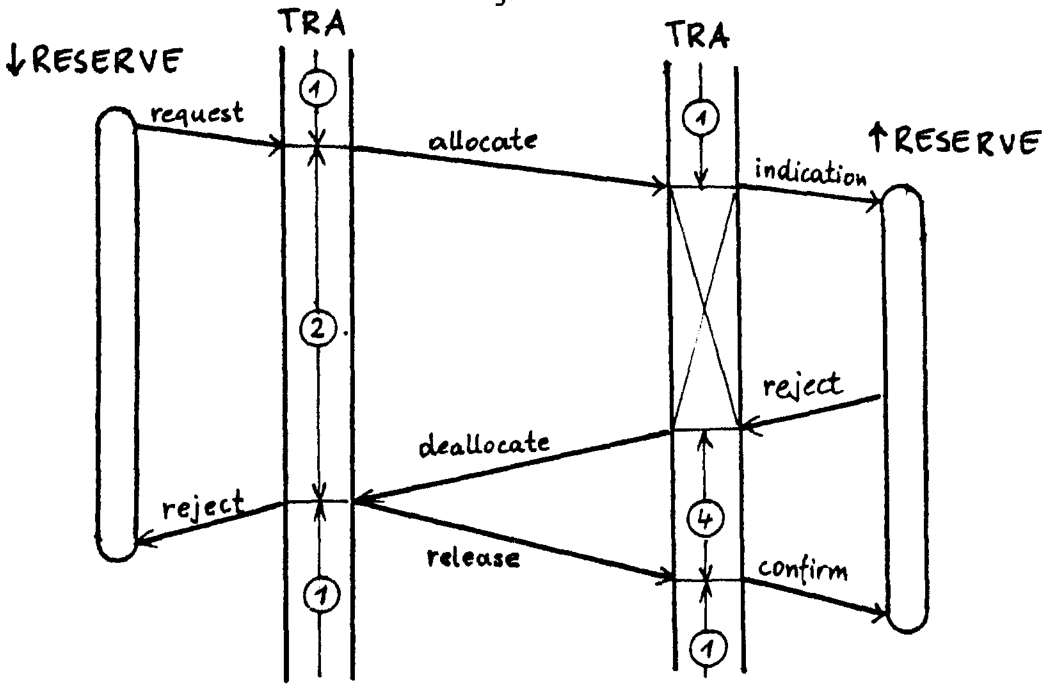
state of TRA / actions	"free" ①	"reserving" ②	"reserved" ③	"freeing" ④
¯↓RESERVE (request)	_↓allocate; ⇒ ②	Y (not possible)	Y	Y
¯↓ FREE (request)	Y	Y	_↓ deallocate ⇒ ④	Y
time out or _↑ RESET	/ (no action)	retransmit last p.d.u.	/	retransmit last p.d.u.
_↑ CLEAR	/	¯↓RESERVE (reject) [status= broken] ⇒ ①	¯↑ FREE [status= broken] ⇒ ①	confirm last service primitive with [status= broken] ⇒ ①
_↑ allocate	¯↑RESERVE (indication); **either** [¯↑RESERVE (accept); _↓ confirm; ⇒ ③] or [¯↑RESERVE (reject); _↓ deallocate; ⇒ ④]	reserve collision on the same channel if the <u>for-whom</u> of the received allocate has prio. over the <u>for-whom</u> for which TRA is reserving **then** [¯↓RESERVE (reject) [reason=collision], action as in state ①]	the allocate should be a retransmission; retransmit response: _↓ confirm	if it is a retransmission **then** retransmit response: _↓ deallocate **else** this is a new request: action as in state ①
_↑ confirm	X (should never happen)	¯↓ RESERVE (accept) ⇒ ③	should be retransmission: / obs: an initial transm. of confirm would indicate an unresolved collision conflict.	as in state ③
_↑ deallocate	should be a retransmission: _↓ release	if not a re- transmission **then** _↓ release; ¯↓RESERVE (reject) ⇒ ①	_↓ release; ¯↑ FREE ⇒ ①	freeing collision: confirmation of last service primitive; _↓ release ⇒ ①
_↑ release	should be a retransmission or collision: /	X	X	confirmation of last service primitive; ⇒ ①

Operation of the TRA protocol
fig.7

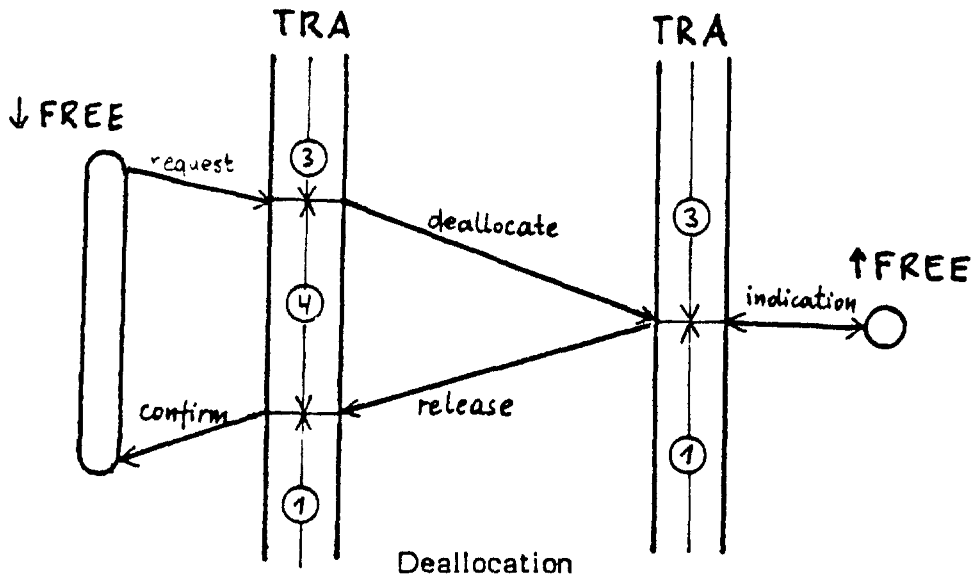
Some examples for the operation of the protocol are given in figure 8,9,10 and 11. We note that there may be three interface events for the RESERVE primitive, as shown in figure 9.



Allocation,accepted
fig.8

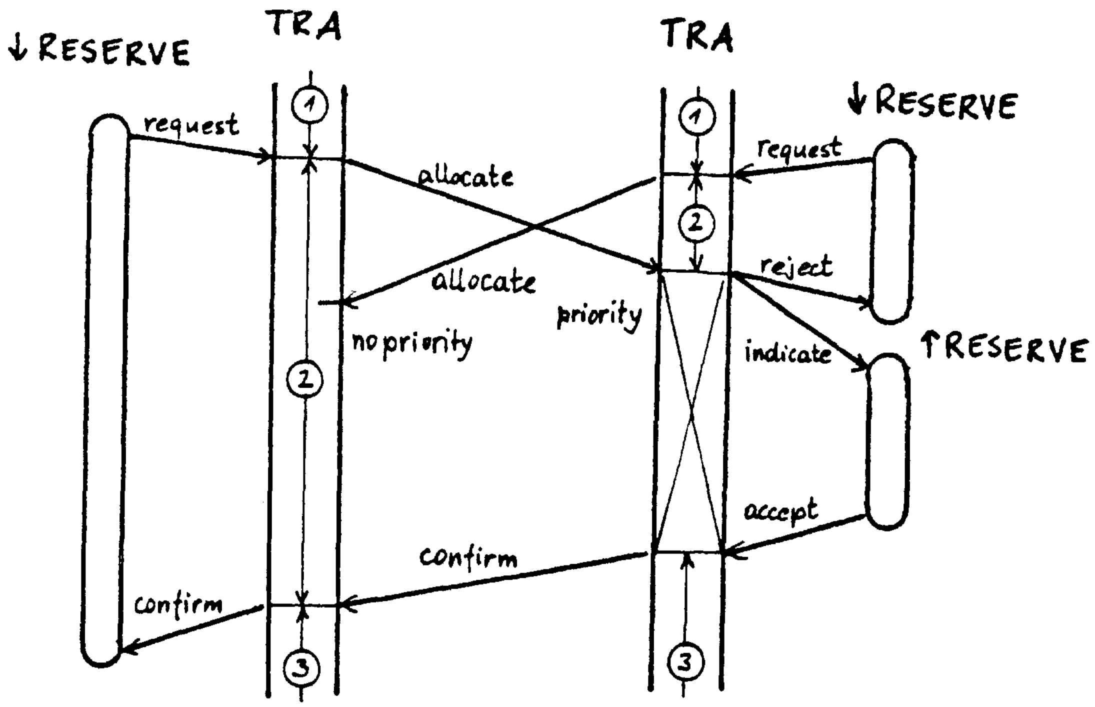


Allocation,not accepted
fig.9



Deallocation
fig.10

The TRA protocol handles collisions, i.e. the case that RESERVE primitives are initiated simultaneously at both ends of the transmission channel (figure 11). This conflict is resolved by giving priority to one side, the reservation for the other side being rejected. The priority decision must be made independently in each TRA entity, based on the exchanged for-whom parameters. Both sides should come to the same. decision. For example, this decision could be based on the DTE addresses of the two sides.



Allocation with collision
fig.11

3.3 Connection Service (CON)

Introduction

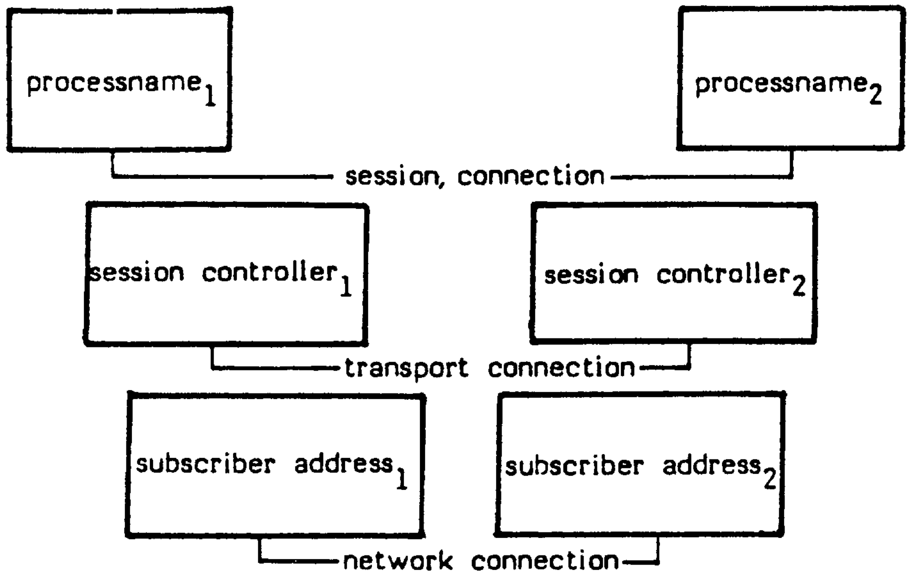
The CON service is responsible for establishing and terminating process to process transport connections. In order to use the flow control of the VC's, each VC is allocated to at most one transport connection.

A session control entity may be understood as an entity requiring a transport connection for communication and performing, together with higher level entities, the user required task. Such a task may be implemented as a set of processes performing different activities within the task, or simply as one process. However the addressing of a process has to be distinguished from the allocation of a transport connection. This does not exclude that an establishment of a transport connection may be combined with process addressing. It is assumed that normally this combination of functions is required, because a transport connection normally is established together with the access to some initial session. This does not exclude that different sessions may use the transport connection during its life time (see section 4.2). This is the justification that the CONNECT primitive establishes initially a transport connection between processes (initial session !).

The service of temporary deallocating the transmission channel, using the RELEASE primitive, is part of the CON service. A RELEASE primitive is also invoked when the transmission channel breaks (the network clears the VC). Note that after a RELEASE is performed, the connection is still existing, although in a "dormant" state. The RESUME primitive may be used to "reactivate" a released connection, or to recover from a broken transmission channel. It allocates a new channel to the resumed connection.

Addressing

The distant subscriber is addressed in an X.25 environment by the distant DTE address. The address which includes the identification of the session controller and the name of the initial process together with a process port name (if more than one connection has to be distinguished by that process) is called "process communication name". A pair of "specific" process communication names identifies a certain process to process transport connection. To make a "generic" process communication name "specific" (indicating the specific instance!) a local process identifier has to be added (see figure 12).



Connections and Addresses
fig.12

No explicit session controller address is forseen in the MLP. The CONNECT primitive implicit connects two session controllers via a transport connection and enables the initial session connection between the identified processes.

Service interface

The service primitives provided by the CON sub-layer are executed in interaction with the TRA sub-layer. The primitives of the message transmission sub-layer (MT see section 3.4) and the session control sub-layers (RS see section 4.1, MS see section 4.2) may also be executed when the CON entity is in the "connected" state.

The service primitives of the CON sub-layer are the following:

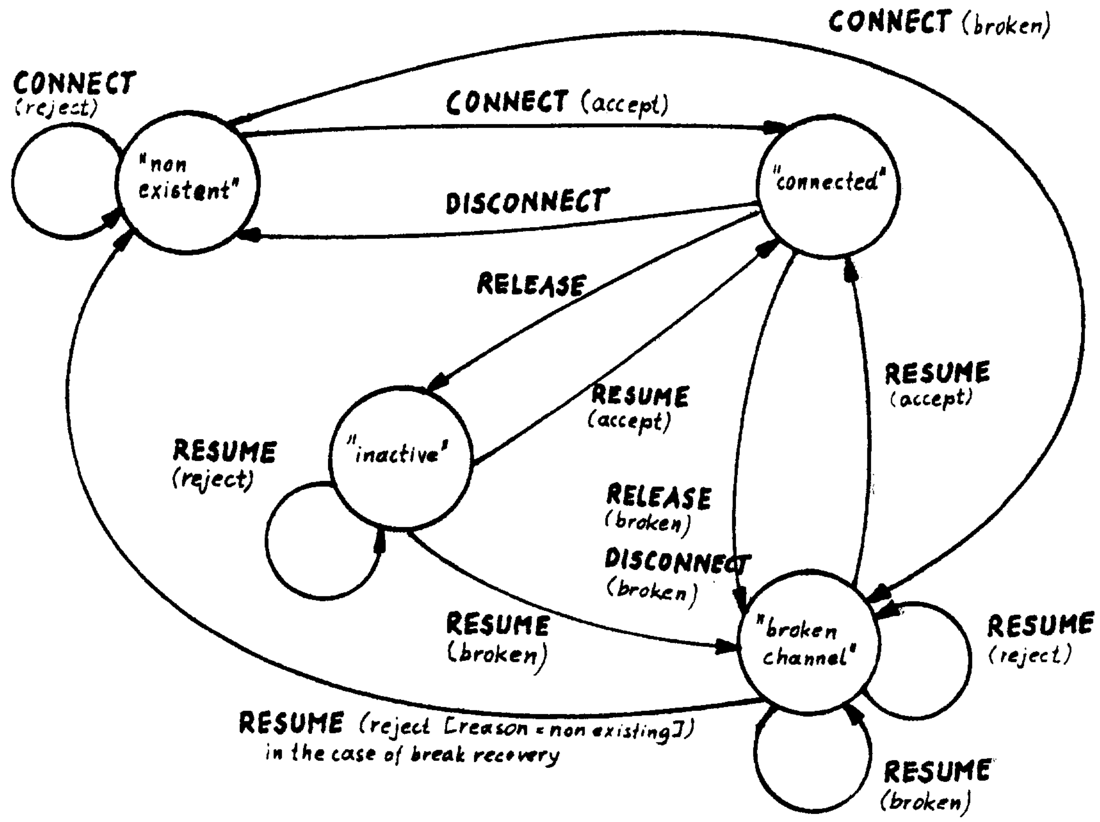
↓ or ↑ RECONNECT
and
↓ or ↑ CONNECT (→ local generic process communication name,
→ local process identifier,
→ distant subscriber address,
→ distant generic process communication name,
← distant process identifier,
→ service class,
← response: (accepted,non-accepted,collision),
↑ status,
→ break recovery: boolean,
→ protocol identifier)

↓ or ↑ DISCONNECT (↑ status)

↓ or ↑ RELEASE (↑ status)

↓ or ↑ RESUME (→ local generic process communication name,
→ local process identifier,
→ distant subscriber address,
→ distant generic process communication name,
→ distant process identifier,
→ service class,
← response: (accepted,non-accepted,collision, non-existing)
↑ status)

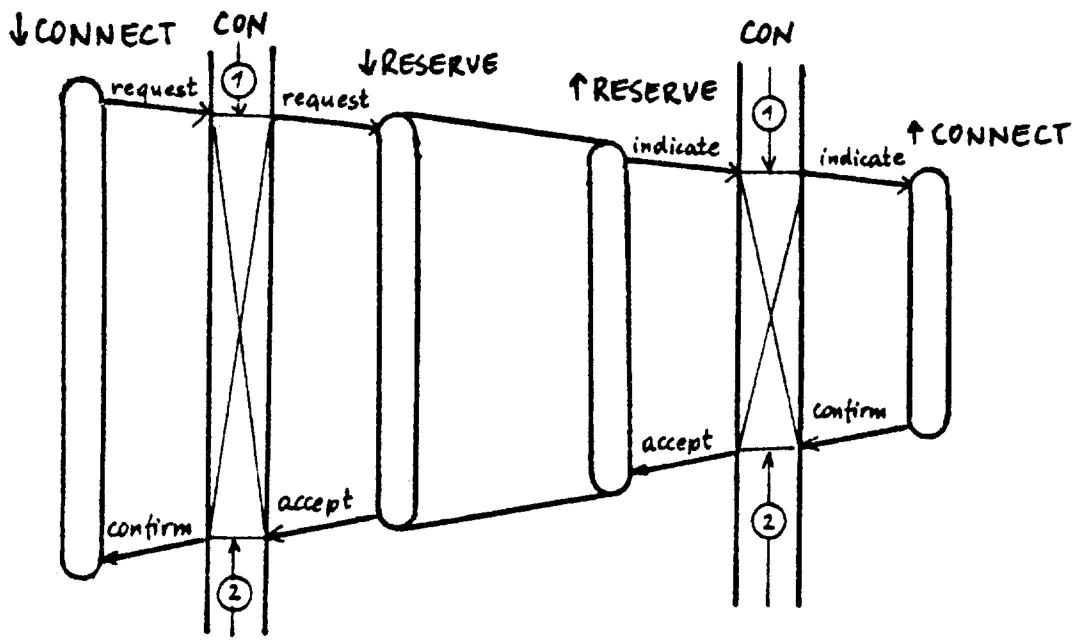
The service primitives may be executed only in a certain order, as shown in figure 13. The figure also shows the possible states of a connection entity.



Order of CON service primitives
fig.13

"Reject" indicates that for any reason the <u>distant side</u> does not accept the request of the service primitive (CONNECT,RESUME). The status "broken" indicates that a <u>clear</u> occured during a requested service primitive.

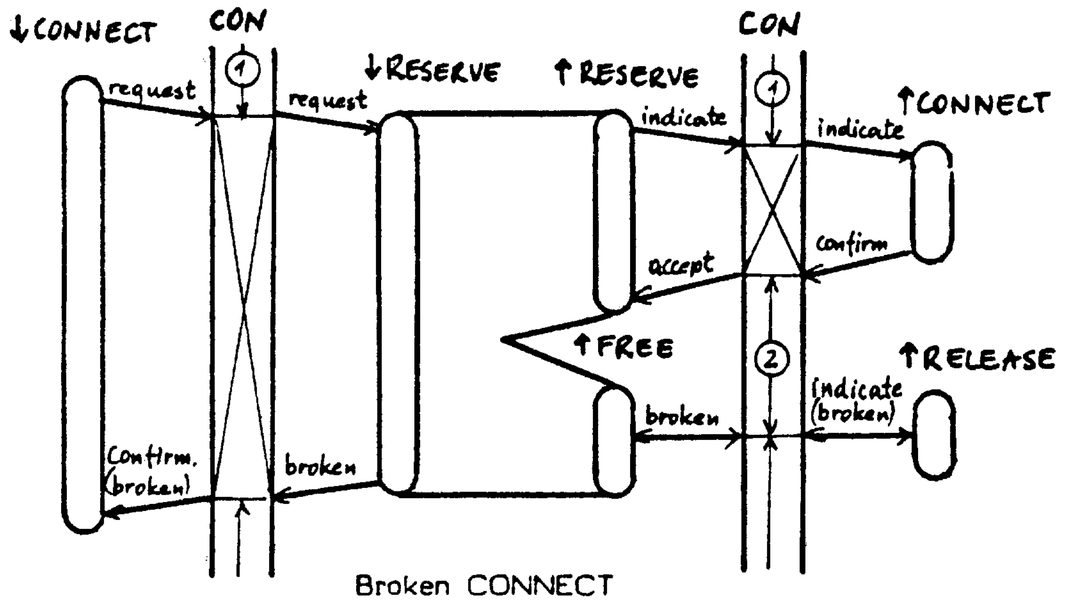
An example of the operation of the protocol in the asymmetric CONNECT is shown in figure 14.



① "non existent"
② "connected"

Asymmetric CONNECT
fig.14

The following example illustrates the situation where a break is indicated after the other side has confirmed the CONNECT (see figure 15).

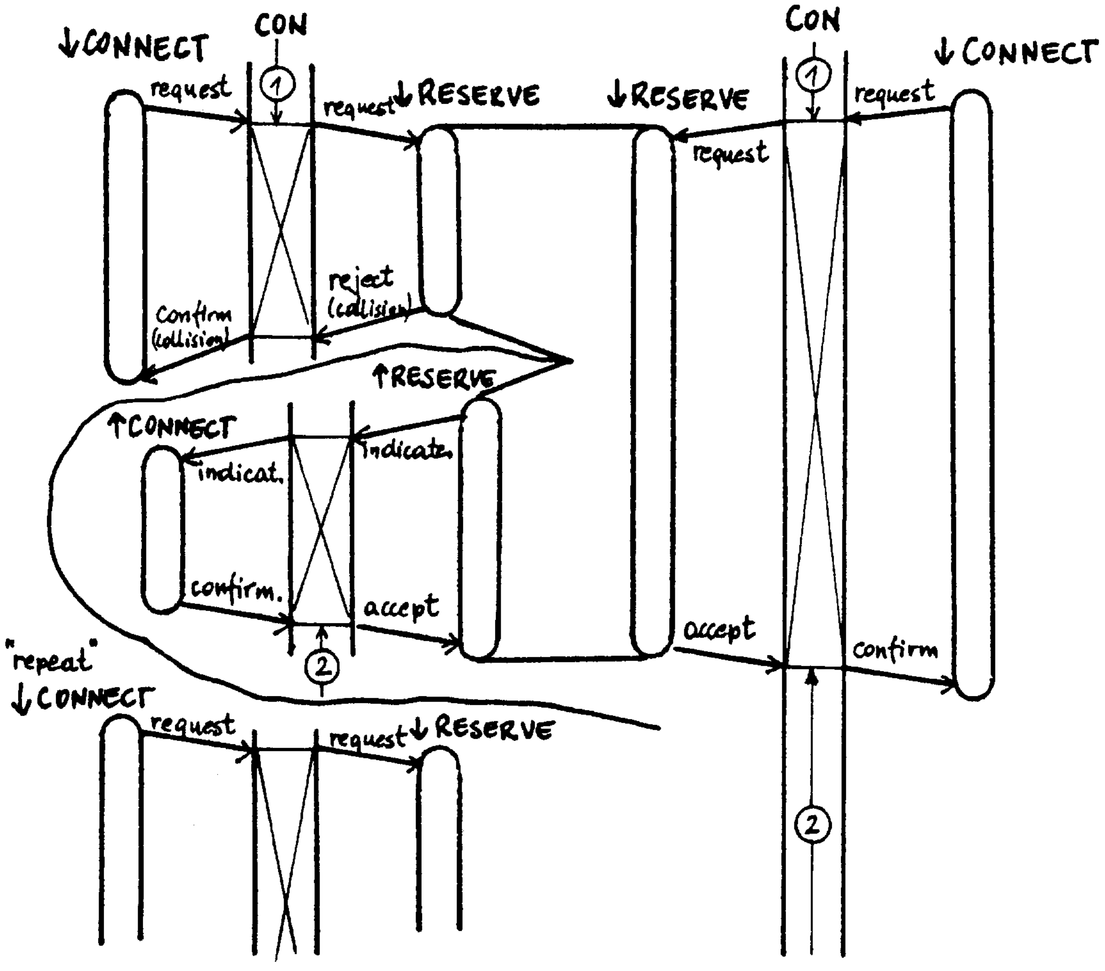


Broken CONNECT
fig.15

Two problems have to be considered in the symmetric case (i.e. if both sides initiate CONNECT and RESUME primitives):

a) collision

It occurs if two different process-to-process transport connections try to reserve the same VC. This conflict is handled by the TRA sub-layer, which rejects the reservation for one of the connection. The CON sub-layer then has to RESERVE another circuit, as shown in figure 16.

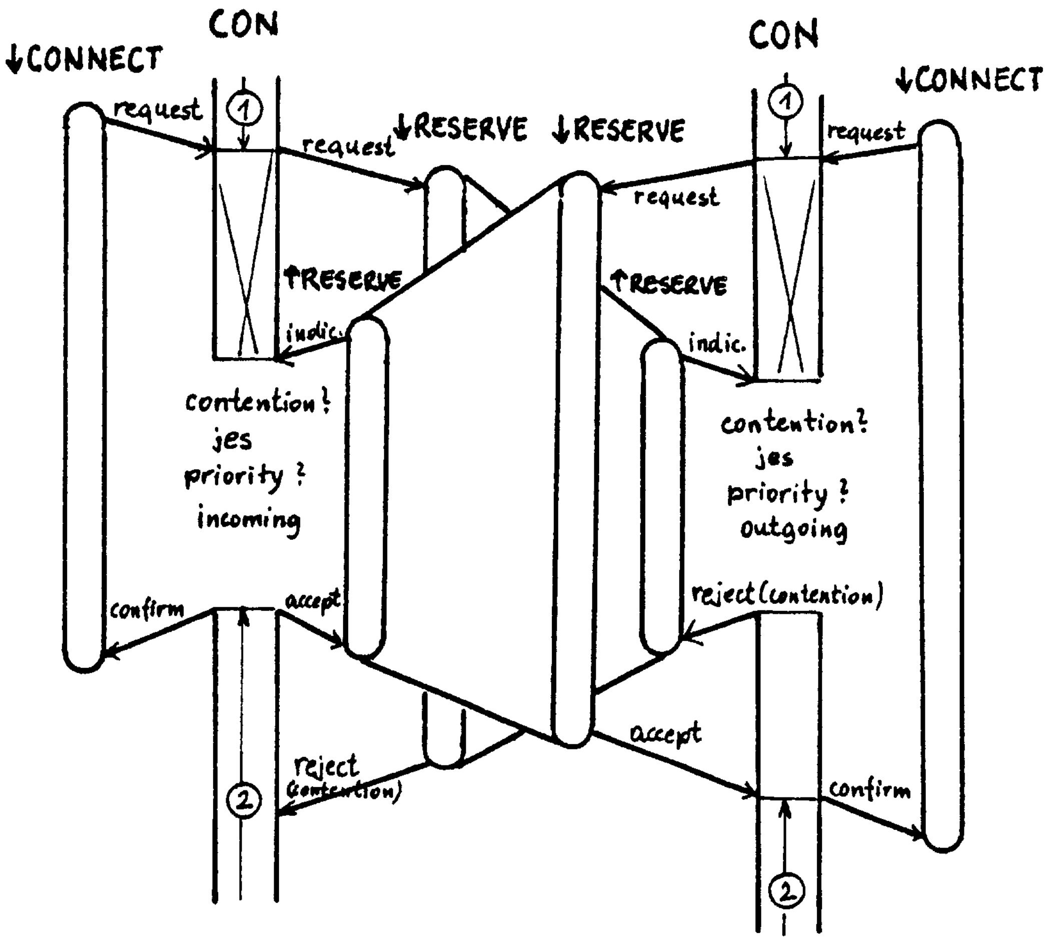


① "non existent"
② "connected"

collision
fig. 16

b) contention

In this case two different circuits are established for the same process-to-process transport connection. The connection administration entity of the CON sub-layer has to detect contention. It is assumed that the process communication names are sufficient for this purpose. When contention is detected each side has to evaluate whether the outgoing or incoming CONNECT has priority. This evaluation is based on the asymmetric subscriber addresses. The side with the lower subsriber address has to reject the incoming CONNECT. The other side has to accept the incoming CONNECT. Figure 17 shows an example of the protocol operation in the contention case.

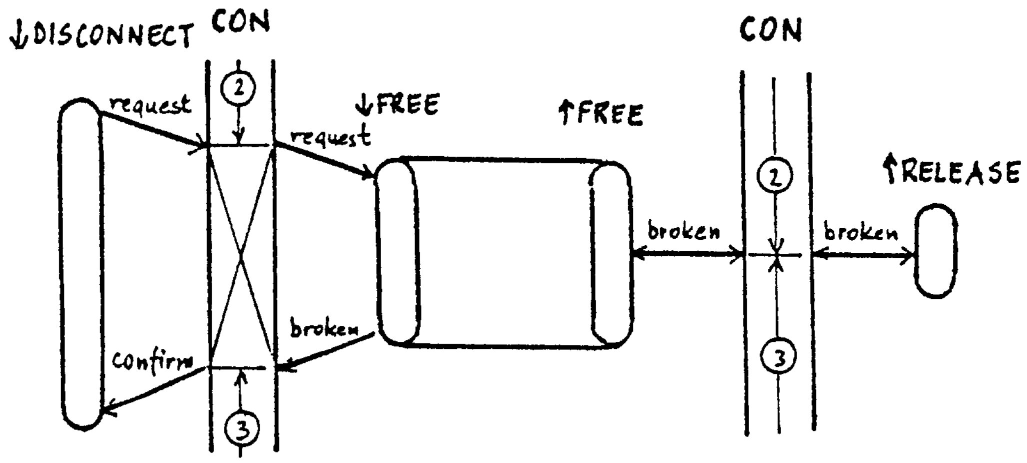


① "non existent"
② "connected"

contention
fig.17

Collision and contention may also occur between CONNECT - RESUME, RESUME - CONNECT and RESUME - RESUME primitives. These cases are handled in an analogue manner, giving RESUME priority over CONNECT.

DISCONNECT is the interface primitive responsible for terminating the process-to-process connection. It uses the FREE primitive of the TRA sub-layer with "disconnect" as the why parameter. Figure 18 shows a relatively unusual case where the transmission channel breaks during the execution of the DISCONNECT.



② "connected"
③ "broken channel"

Broken DISCONNECT
fig.18

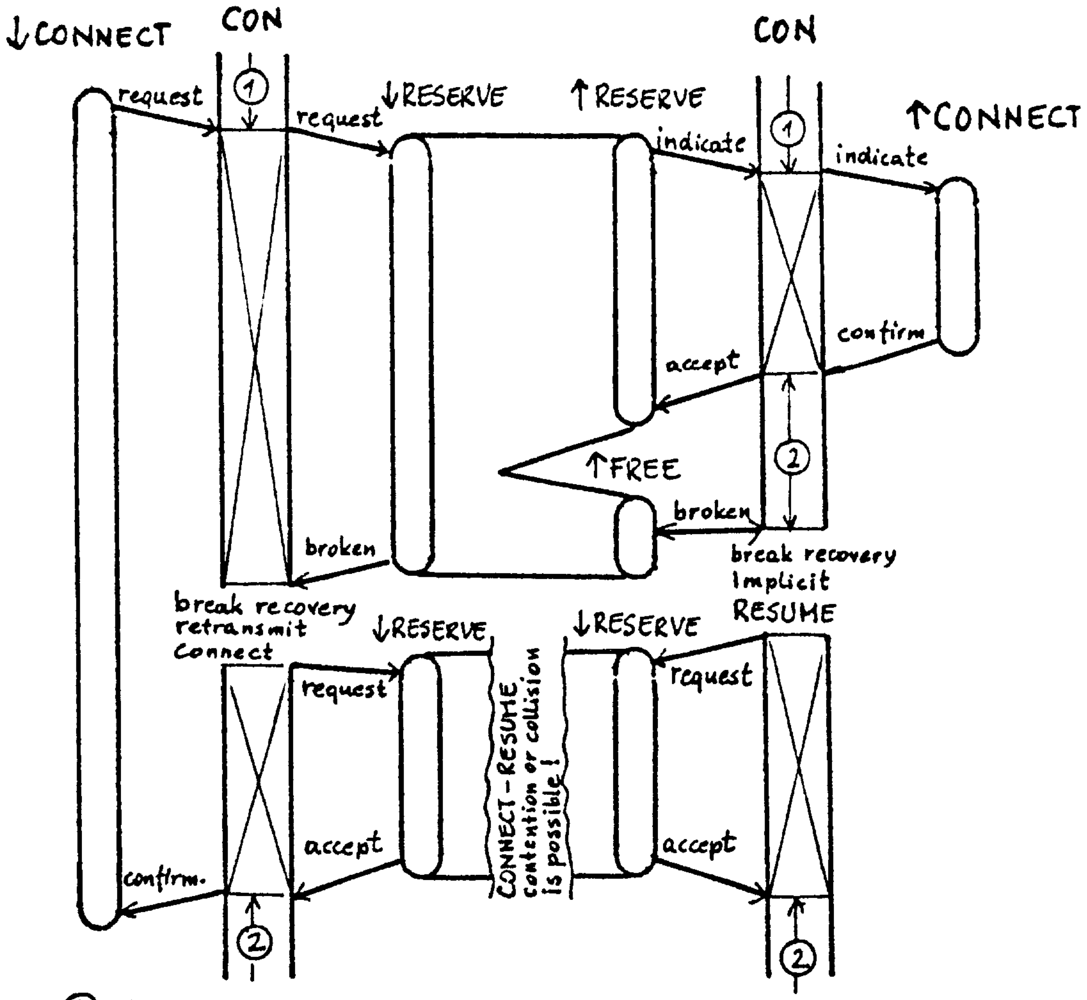
RECONNECT is logically equivalent to a DISCONNECT followed by a CONNECT, but may be implemented more efficiently (for example by not "freeing" the transmission channel if the distant subscriber address for the new connection is the same as before).

Break recovery

Recovery after network initiated clears is one of the most important features for a reliable transport service based on VC's (CHUNG78,CHUNG79). In MLP, this is provided explicitly or implicitly ("automatic break recovery"). To recover from a network initiated clear (indicated by status "broken") a RESUME function has to be performed. Whether the RESUME is accepted (leading to the "connected" state) or rejected (leading to the "non existent" state) the states on both sides are again synchronized.

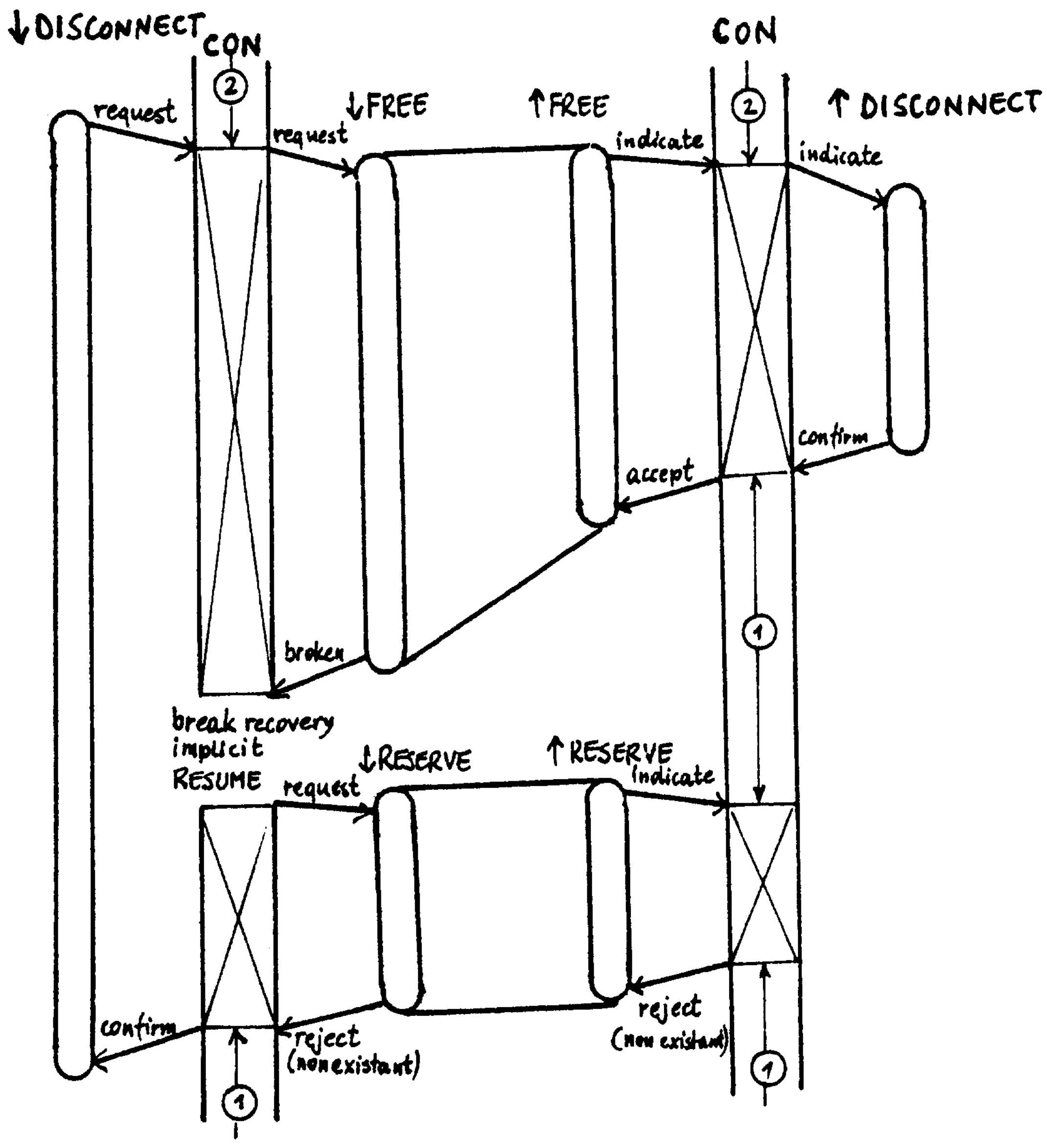
This RESUME function can be performed either by the user of the CON service (explicit recovery of broken circuit) or implicitly by a service called "break recovery". The second case gives the justification to introduce a sub-layer above the CON sub-layer providing recovery after a VC is "broken". The interface above this sub-layer does not indicate the status "broken" to any primitive invoked (and no "broken" RELEASE is indicated either, of course).

Examples for this case are shown in the figures 19 and 20. If a DISCONNECT is confirmed by a "broken", the "broken channel" state is reached (figure 20). In this case it does not know in which state the other side has been the time a "broken" was indicated. It could have been in the "connected" state, as shown in figure 19. Or it could have been in the DISCONNECTing, or in "non existent" state. These cases are distinguished by invoking a RESUME in the "broken channel" state.



① "non existent"
② "connected"

CONNECT with "break recovery"
fig.19



① "non existent"
② "connected"
③ "broken"

DISCONNECT with "break recovery"
fig.20

If no "break recovery" is agreed by the parameter of the CONNECT primitive, and no RESUME is expected (neither from the user nor from an automatic break recovery), the "non existant" state is reached directly from the "broken channel" state (see figure 13).

3.4 Message Transport (MT)

The MLP provides a sequential message transport service simultaneously and independently in both directions. In addition, a simultaneous interrupt transfer in both directions is supported. Different classes of service are distinguished.

Basically the message transport service of the MLP is equivalent to the service provided by a virtual circuit (see section 3.1), exept that some octets of each user sequence may be reserved for control purposes. However, the transmission network may reset the VC, which may lead to loss of data, or clear the call (in the case of switched VC's). A clear may be recovered by the "break recovery" function of the CON sub-layer (a recovered clear looks like a reset), if this is desired. Resets may be acceptable to certain applications, but certainly not for all, since data loss may lead to undefined system states and deadlocks. We consider two approaches to avoiding data loss, which lead to two classes of service (as described below):

a) A reset leads to disconnection. This is a quite drastic way to avoid the loss of data. It is only feasable if network generated resets are sufficiently seldom.

b) A retransmission mechanism provides recovery from data loss after resets. In this case, the "break recovery" function of the CON sub-layer provides additional reliability and availability for the transport service.

A "high reliable class" of service (see section 3.4.2) is obtained by adopting approach (b) and introducing additional checks, at the receiving transport entity, for transmission errors and sequential delivery.

If for some reason the approaches above are not appropriate (e.g. applications which recover resets or clears themselves) resets may be signalled to the user through the service interface by ↑RESET primitives.

3.4.1 Normal class of service

The occurence of network resets and clears is assumed seldom enough for the user requirements. Network resets and clears give rise to a DISCONNECT - no attempt is made to recover the possible loss of data.

3.4.3 Reliable and high reliable class of service

The service is provided through the same interface primitives as the normal class of service. However network generated resets are recovered by the MT sub-layer, and transmission malfunctions may be detected and recovered. The absense of message loss is guaranteed, and at any given time, the last interrupt sent (in each direction) is guaranteed not to be lost. If in addition, the "break recovery" function of the CON sub-layer is used, one obtains a transport service of high reliability and availability.

Delivery confirmation within the MT sub-layer for the reliable class of service has two functions. As for the normal class of service, it provides a "one sided" synchronization, and in addition, it is used for the buffer management by the MT entity for the messages for which retransmission is not excluded. The rt-mark protocol data unit is used to determine the necessary retransmission. It contains the next expected message number, as well as the number and content of the last interrupt sent (This information is recorded at each side of the transport connection; the message and interrupt numbers are initialized by the CONNECT primitive). The rt-mark is exchanged after each reset.

Two sub classes of the reliable service may be distinguished:

- with sequence and transmission error detection (high reliable class):

A sequence number and transmission error check is added to each transmitted message. Therefore any malfunction of the transmission medium can be detected by the receiving transport entity. If a sequence error or a transmission error is dedected a reset is sent over the transmission channel by the MT sub-layer. This leads to the same recovery mechanism as for network generated resets (see above). For this class, the "data" protocol data units have to carry a sequence number and a transmission error checksum.

- without sequence and transmission error detection (reliable class):

The integrity of the received messages relies on the correct operation of the underlying data transmission service. The retransmission recovery mechanism is initiated after network generated resets only. No sequence number nor error check is added to the transmitted messages, thus minimizing the overhead.

The MT service consists of the primitives:

$\downarrow$ MESSAGE ($\rightarrow$ message,
$\rightarrow$ delivery confirmation request: boolean)

$\uparrow$ MESSAGE ($\rightarrow$ message)

$\downarrow$ or $\uparrow$ INTERRUPT ($\rightarrow$ code: 8 bits)

Two examples of the operation of the MT protocol are given below.

a) Message exchange without delivery confirmation (figure 21)

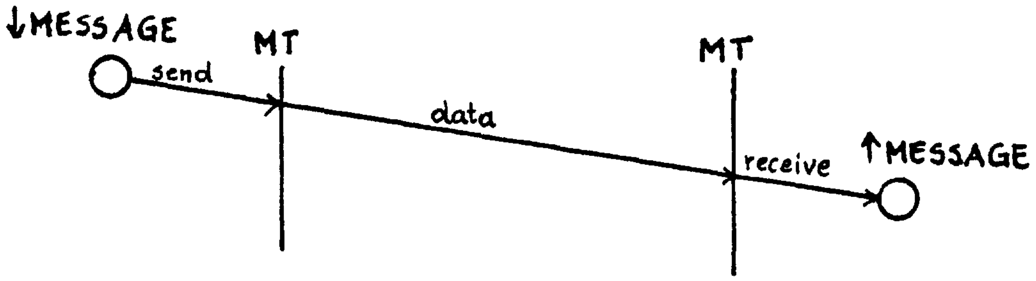


fig.21

b) Message exchange with delivery confirmation (figure 22)

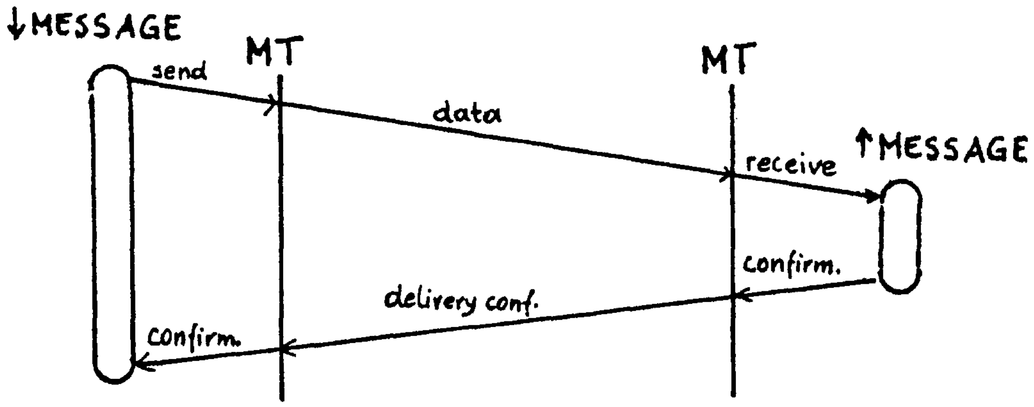
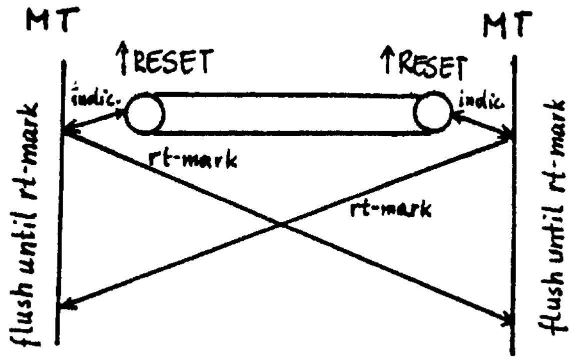


fig.22

The protocol data units for the TS protocol are <u>data</u> and <u>delivery confirmation</u>. <u>Data</u> is transmitted as a user sequence containing the indicator "data" together with the parameter "delivery confirmation required" in the first octet, and the parameter "message" in the following octets.

An example of the operation of the retransmission recovery mechanism is shown in figure 23. It shows what happens when a reset occurs during data transfer.



rt-mark	(next expected message number, last interrupt content sent, last interrupt number)

Reset recovery
fig.23

If the next expected "data" protocol data unit number contained in the received rt-mark is lower than the number of the last "data" protocol data unit sent, the lost "data" will be retransmitted subsequently. If the last interrupt number contained in the received rt-mark is higher than the number of the last interrupt received, the last interrupt content of the rt-mark gives rise to an ↑ INTERRUPT primitive (and the local interrupt receive counter is updated).

4. Session Control Service

The Re-Synchronization (RS) and the Mode Exchange and Synchronization (MS) sub-layers of the MLP belong to the session control layer. The basic session control service provided by the MS sub-layer is obtained through an extension of the SYNCHRONIZATION primitive of the RS sub-layer. Other functions of session control, like commitment control and logically related sessions, are left for further studies. The here described session control functions may be used as a basis for a presentation control layer, as described in (RAUBOLD77).

4.1 Re-Synchronisation service (RS)

This service may be used to provide a negotiation mechanism for applications. It is similar to the synchronization proposed in (BAUW77).

The following synchronization primitives are provided at the service interface:

- DATA SYNChronization
  This primitive may be used by a communicating process to enter a new phase of operation. The meaning depends on previosly agreed understanding of phases between the communication entities.

- INTERRUPT SYNChronization
  This primitive may be used independently of the flow control mechanism for messages, whereas the primitive above may be blocked within the message flow between the two communication entities. Two types of INTERRUPT synchronization, discarding and non-discarding, are distinguished. INTERRUPT synchronization primitives exist at different priorities. The initiation of a higher priority interrupt automatically leads to abandoning any lower priority primitive in progress.

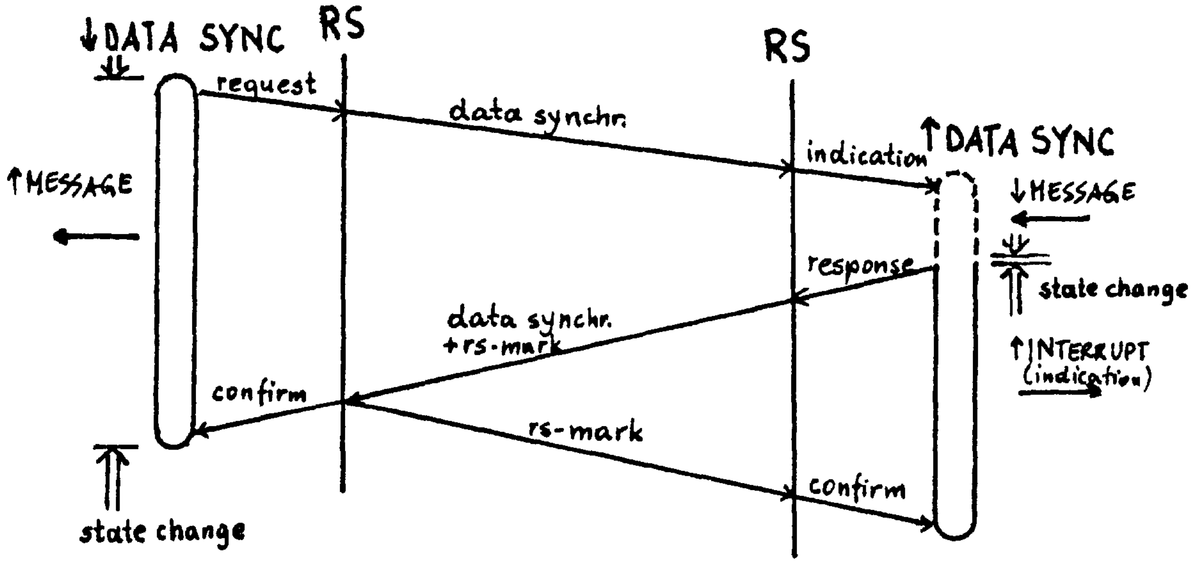
The RS service consists of the following primitives (where a high integer parameter value indicates high priority):

↓ or ↑ DATA SYNC	(→	synchronization parameter,
	←	accepted: boolean)
↓ or ↑ INTERRUPT SYNC	(→	priority: 0...127,
	↑	abandoned: boolean)
↓ or ↑ FLUSH INTERRUPT SYNC	(→	priority: 128...255,
	↑	abandoned: boolean)

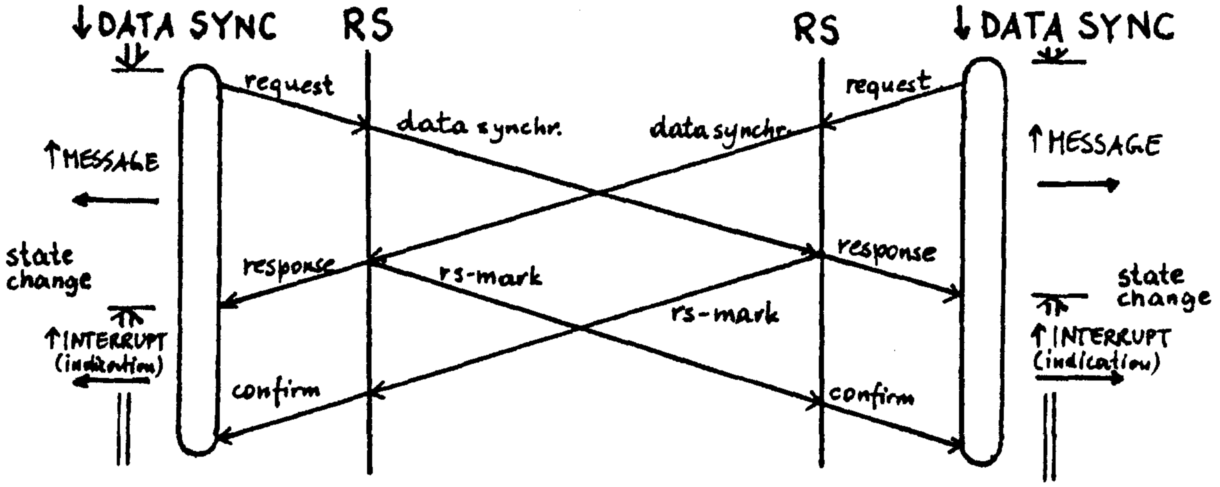
The protocol data units of the RS sub-layer (exchanged between two RS entities to provide the primitives above) are the "data synchr." (identical with "data synchr. accept"), the "data synchr. reject", the "rs-mark" and the "interrupt". The last one is transmitted as an X.25 interrupt and not as a user sequence, of course.

In general the RS service primitives can operate in parallel with MESSAGE and INTERRUPT primitives. Several restrictions depending on the primitive and state have to be considered. In the following examples the "dotted" phase of an service primitive allows for parallelism with the ↓ MESSAGE primitive. The remaining phase may only operate in parallel with ↑ MESSAGE, and possibly with an ↑ INTERRUPT SYNC. The data received by ↑ MESSAGE and sent by ↓ MESSAGE belong to the phase before the invokation of the synchronization. The possibly arriving indications of ↑ INTERRUPT SYNC or ↑ FLUSH INTERRUPT SYNC belong to the phase after the previously invoked synchronization. No MESSAGE primitive may operate in parallel with the FLUSH INTERRUPT SYNC primitive.

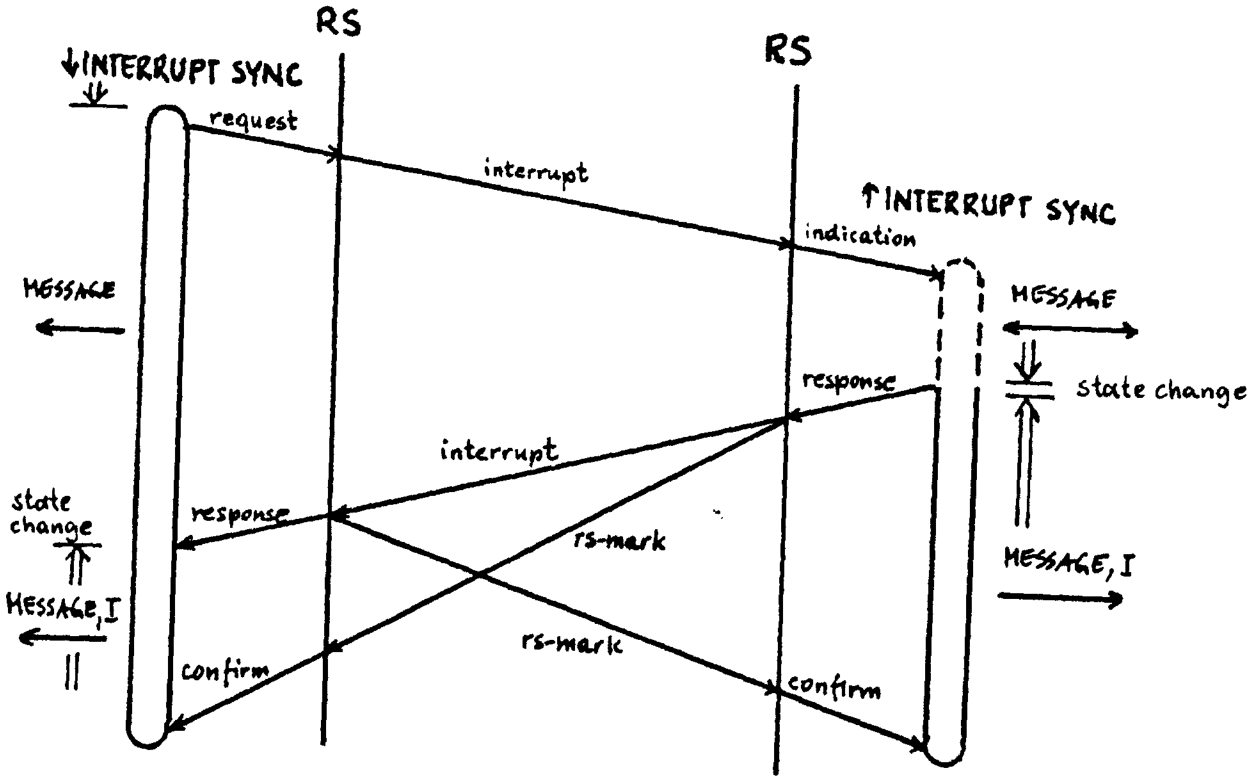
Several examples of RS service primitives and the related protocol operations are described in the following figures (24,25,26,27 and 28):



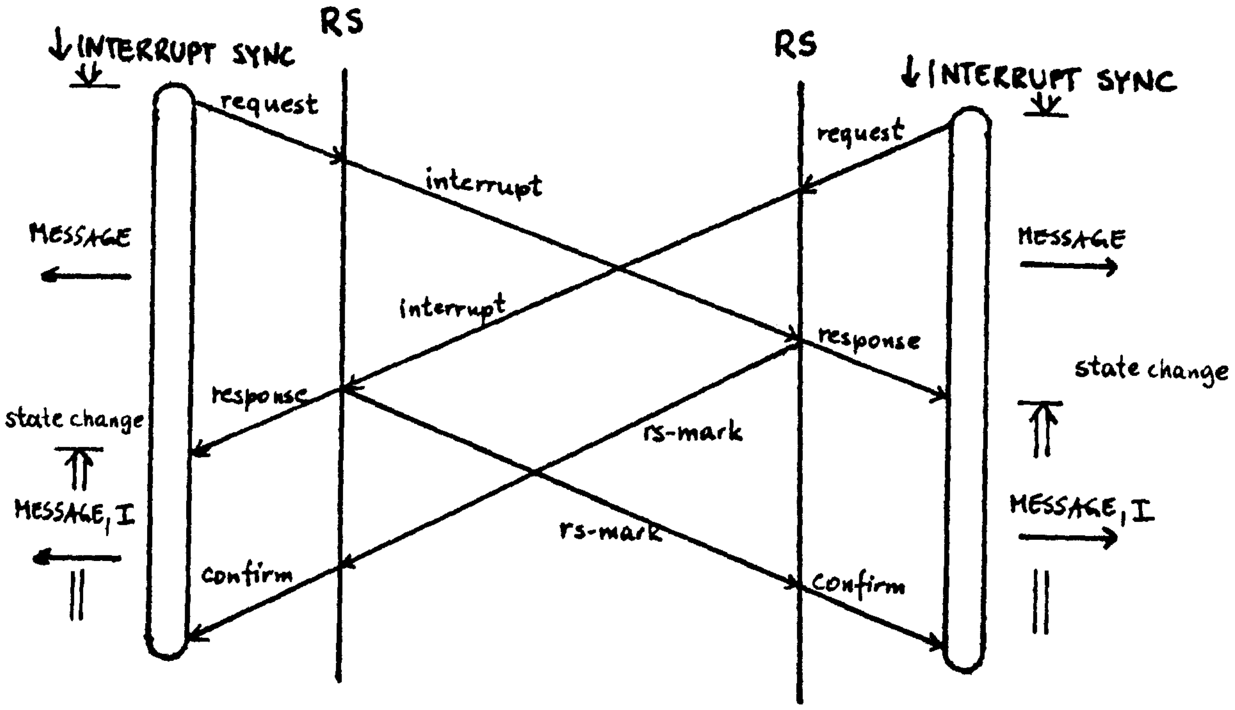
Asymmetric DATA SYNChronisation
fig.24



Symmetric DATA SYNChronisation
fig.25



Asymmetric INTERRUPT SYNChronisation
fig.26

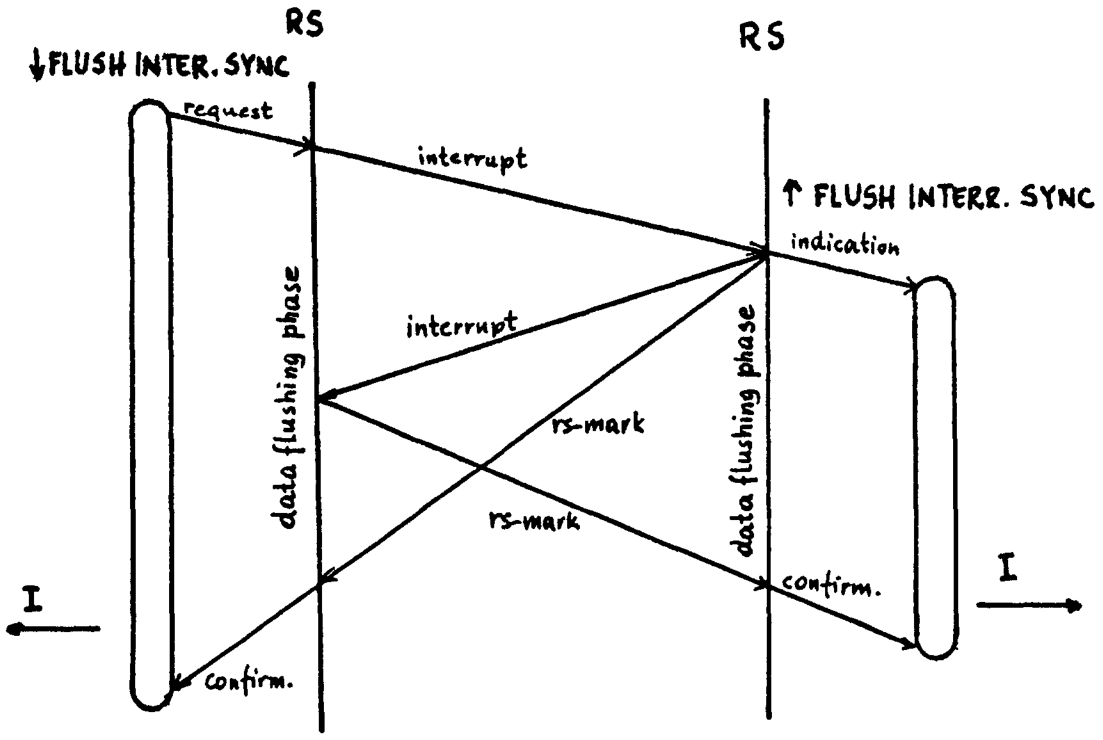


Symmetric INTERRUPT SYNC
fig.27

The symbol " I " in figure 26 and figure 27 indicates that during this phase of the INTERRUPT SYNC primitive a new interrupt protocol data unit may arrive. This conflict with the INTERRUPT SYNC in progress is resolved as follows:

- if a lower priority interrupt arrives its indication to the layer above has to wait until the INTERRUPT SYNC in progress is confirmed,
- if a higher priority interrupt arrives the current INTERRUPT SYNC is abandoned and the new INTERRUPT SYNC is indicated immediatly.

An example for the FLUSH INTERRUPT primitive is given in fig. 28



FLUSH INTERRUPT SYNC
fig.28

4.2 Mode Exchange and Synchronisation of processes (MS)

The synchronization mechanism propsed in the RS sub-layer can be used for selecting a new communication mode as well as to select a new process (providing this communication mode!). If, therefore, the synchronization parameter of the RS interface is used to select from a set of available synchronization primitives with different additional parameters, a basic set of session control primitives is created. The MLP distinguishes three primitives to perform mode exchange and synchronization of processes:

```
↓ or ↑ OPEN   ( →  open parameter,
                ←  accepted: boolean)


↓ or ↑ CLOSE


↓ or ↑ REOPEN (  →  open parameter,
                 ←  accepted: boolean)
```

The MS protocol data units of the MS sub-layer are "sub protocol data units" to the already defined "data synchronization" protocol data unit of the RS sub-layer (see section 4.1), i.e. the synchronization parameter of the "data synchronization" protocol data unit indicates whether the protocol data unit has to be interpreted as an "open", "close" or "reopen" protocol data unit, respectively. The "open accept", "close accept" and "reopen accept" should be coded identically to the "open", "close" and "reopen" protocol data units. The "open reject", and "reopen reject" protocol data units must also be defined.

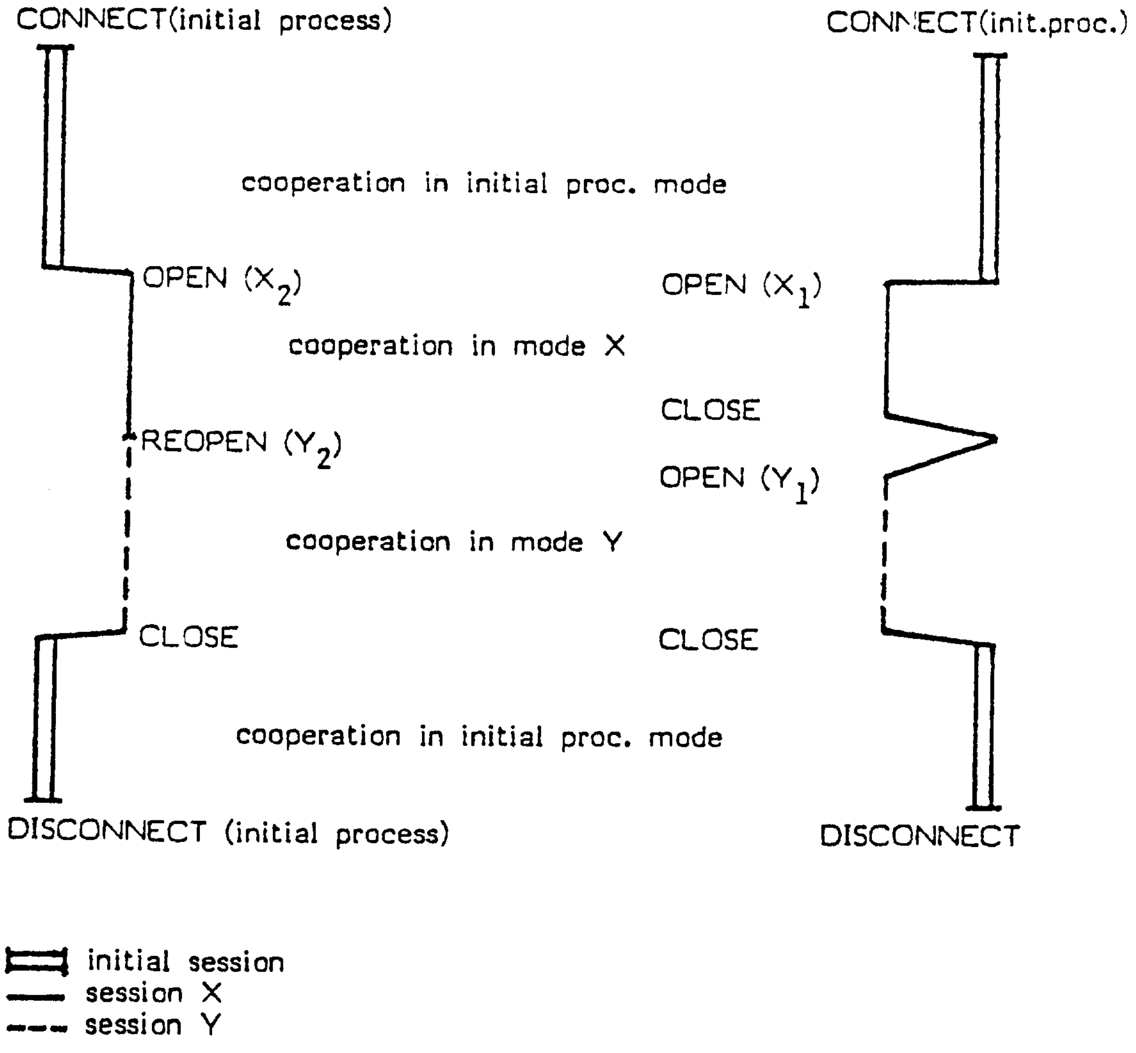
Note that certain values of the synchronization parameter of the DATA SYNC primitive of the RS sub-layer may also be reserved for user defined synchronization data units.

The OPEN (and REOPEN) parameter of the MS service may be used for the following purpose:

- to indicate that a certain process should be accessed,
- to indicate that a certain communication mode should be entered,
- to indicate both, a new process and a new communication mode.

The order in which the OPEN, REOPEN and CLOSE primitives can be used is defined by the "bracket structure" (nesting sessions!). This structure requires that each opening bracket "(" is followed by a corresponding closing bracket ")" with embedded pairs in between. The possible order of the primitives is defined by considering the OPEN and CLOSE primities as opening and closing brackets, respectively, and noting that REOPEN is equivalent to the sequence CLOSE OPEN.

Two functions are considered to be part of the CONNECT primitive (see section 3.2); first the function of establishing a transport connection, and second the function of establishing a session connection between processes (in the following called "initial processes"). This second function is equivalent to the OPEN function of MS sub-layer. Therefore the CONNECT - DISCONNECT bracket can be interpreted as an OPEN - CLOSE bracket for the initial processes. Further OPEN - CLOSE or OPEN - REOPEN - CLOSE brackets may be necessary to describe the communication structure for the cooperation of distributed applications. An example is shown in figure 29.



Communication structure for sessions
fig.29

Acknowledgement

Many thanks to E. Raubold for the numerous fruitful discussions about the initial ideas in this protocol.

References

(HERTW77) F.Hertweck, E.Raubold, F.Vogt
X.25 Based Process-Process Communication
Computer Networks
Volume 2, Number 4/5, September/October 1978
and
Proceedings of Computer Network Protocols
A. Danthine, Editor
Universite de Liege, 1978

(HERTW78) F.Hertweck, E.Raubold, F.Vogt
The ML Protocol description
PIX/HLP/TAG/78/01

(RAUBCLD77) E.Raubold
The PIX communication model
PIX/HLP/GMD/77/06
and
E.Raubold, et.al.
Application protocol design based on an unified communication model
Proceedings of ICCC, Kyoto, 1978

(BAUW77) E.Bauwens, F.Magnee
Remark on negotiation mechanism and attention handling
Universite de Liege, Belgium
Faculte des Siences Appliquees
Systems et Automatique, S.A.R.T. 77/12/13

(CCITT78) CCITT
Provisional Recommandations X.3,X.25,X.28,X.29 on packet switched data transmission services
Geneva 1978 ISBN 92-61-00591-8

(ISO/TC97/SC16/N117)Reference Model of Open Systems Architecture
(Version 3, November 1978)

(CHUNG78) Study of efficient methods for achiving reliable access to packet switched networks
project carried out for the Comp. Comm. Group, Canada, and Master's project, Departement d'IRO, Universite de Montreal

(CHUNG79) R.J.Chung and A.M.Rybczynski
Alternatives for providing highly reliable access to X.25 networks, submitted to NCC 79, New York.