INTERNATIONAL STANDARD

ISO 14229-2

First edition
2013-02-15

Road vehicles— Unified diagnostic services (UDS) —

Part 2: Session layer services

Véhicules routiers — Services de diagnostic unifiés (SDU) —

Partie 2: Séquence des couches de services

Reference number
ISO 14229-2:2013(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 14229-2 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

ISO 14229 consists of the following parts, under the general title *Road vehicles — Unified diagnostic services (UDS)*:

— *Part 1: Specification and requirements*

— *Part 2: Session layer services*

— *Part 3: Unified diagnostic services on CAN implementation (UDSonCAN)*

— *Part 4: Unified diagnostic services on FlexRay implementation (UDSonFR)*

— *Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP)*

— *Part 6: Unified diagnostic services on K-Line implementation (UDSonK-Line)*

The following part is under preparation:

— *Part 7: Unified diagnostic services on Local Interconnect Network implementation (UDSonLIN)*

The titles of future parts will be drafted as follows:

— *Part n: Unified diagnostic services on … implementation (UDSon…)*

Introduction

ISO 14229 has been established in order to define common requirements for diagnostic systems that are independent of the underlying serial data link.

To achieve this, ISO 14229 is based on the Open Systems Interconnection (OSI) Basic Reference Model in accordance with ISO 7498-1 and ISO/IEC 10731, which structures communication systems into seven layers. When mapped on this model, the services used by a diagnostic tester (client) and an Electronic Control Unit (ECU, server) are broken into the following layers in accordance with Table 1:

— Application layer (layer 7), unified diagnostic services specified in ISO 14229-1, ISO 14229-3 UDSonCAN, ISO 14229-4 UDSonFR, ISO 14229-5 UDSonIP, ISO 14229-6 UDSonK-Line, ISO 14229-7 UDSonLIN, further standards and ISO 27145-3 WWH-OBD.

— Presentation layer (layer 6), vehicle manufacturer specific, ISO 27145-2 WWH-OBD.

— Session layer services (layer 5) specified in this part of ISO 14229.

— Transport layer services (layer 4), specified in ISO 15765-2 DoCAN, ISO 10681-2 Communication on FlexRay, ISO 13400-2 DoIP, ISO 27145-4 WWH-OBD.

— Network layer services (layer 3), specified in ISO 15765-2 DoCAN, ISO 10681-2 Communication on FlexRay, ISO 13400-2 DoIP, ISO 27145-4 WWH-OBD.

— Data link layer (layer 2), specified in ISO 11898-1, ISO 11898-2, ISO 17458-2, ISO 13400-3, IEEE 802.3, ISO 14230-2 and further standards, ISO 27145-4 WWH-OBD.

— Physical layer (layer 1), specified in ISO 11898-1, ISO 11898-2, ISO 17458-4, ISO 13400-3, IEEE 802.3, ISO 14230-1, further standards, ISO 27145-4 WWH-OBD.

Table 1 — Example of diagnostic/programming specifications applicable to the OSI layers

<table>
<tr><th>Applicability</th><th>OSI seven layer</th><th colspan="5">Enhanced diagnostics services</th><th>WWH-OBD</th></tr>
<tr><td rowspan="9">Seven layer according to ISO/IEC 7498-1 and ISO/IEC 10731</td><td>Application (layer 7)</td><td colspan="5">ISO 14229-1, ISO 14229-3 UDSonCAN, ISO 14229-4 UDSonFR, ISO 14229-5 UDSonIP, ISO 14229-6 UDSonK-Line, ISO 14229-7 UDSonLIN, further standards</td><td>ISO 27145-3</td></tr>
<tr><td>Presentation (layer 6)</td><td colspan="5">vehicle manufacturer specific</td><td>ISO 27145-2</td></tr>
<tr><td>Session (layer 5)</td><td colspan="6">ISO 14229-2</td></tr>
<tr><td>Transport (layer 4)</td><td rowspan="2">ISO 15765-2</td><td rowspan="2">ISO 10681-2</td><td rowspan="2">ISO 13400-2</td><td rowspan="2">Not applicable</td><td>further standards</td><td rowspan="4">ISO 27145-4</td></tr>
<tr><td>Network (layer 3)</td><td>further standards</td></tr>
<tr><td>Data link (layer 2)</td><td rowspan="2">ISO 11898-1, ISO 11898-2</td><td>ISO 17458-2</td><td rowspan="2">ISO 13400-3, IEEE 802.3</td><td>ISO 14230-2</td><td>further standards</td></tr>
<tr><td>Physical (layer 1)</td><td>ISO 17458-4</td><td>ISO 14230-1</td><td>further standards</td></tr>
</table>

Road vehicles — Unified diagnostic services (UDS) — Part 2: Session layer services

1 Scope

This part of ISO 14229 specifies data link independent requirements of session layer services.

This part of ISO 14229 specifies common session layer services to provide independence between unified diagnostic services (ISO 14229-1) and all transport protocols and network layer services (e.g. ISO 15765-2 DoCAN, ISO 10681-2 Communication on FlexRay, ISO 13400 DoIP, ISO 14230-2 DoK-Line, etc.)

This part of ISO 14229 specifies a common service primitive interface between OSI layer 4 (Transport) and layer 5 (Session) via so-called service request/confirmation/indication primitives. This interface allows seamless implementation of ISO 14229-1 Unified diagnostic services (UDS) with any communication protocol titled "DoXYZ / CoXYZ" like ISO 15765 DoCAN – Diagnostic communication over Controller Area Network, ISO 13400 DoIP, ISO 10681 Communication over FlexRay, ISO 14230 DoK-Line.

ISO 15031 (emissions-related OBD) and ISO 27145 (WWH-OBD) support the standardized service primitive interface.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 14229-1, *Road vehicles — Unified diagnostic services (UDS) — Part 1: Specification and requirements*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
gateway
networking device that transfers the PDU on different OSI layers

EXAMPLE A network device that enables communication between control module networks that use different communication protocols, different communication rates, etc. That includes, but is not limited to, gateway functionalities like bridge, switch, router or application layer routing.

3.1.2
router
networking device that transfers the PDU on OSI layers 3 and 4

3.1.3
switch
networking device that transfers the PDU on OSI layer 2

3.2 Abbreviated terms

CDD	common data dictionary
CMD	common message dictionary
DSC	diagnostic session control
ECU	electronic control unit
OSI	open systems interconnection
S_AE	session layer address extension
S_SA	session layer source address
S_Data	session layer data transfer service name
SI	service identifier
SOM	start of message
S_Mtype	session layer message type
S_PDU	session layer protocol data unit
S_TA	session layer target address
S_TAtype	session layer target address type

4 Conventions

This part of ISO 14229 is guided by the conventions discussed in the OSI Service Conventions (ISO 10731:1994) as they apply to the diagnostic services. These conventions specify the interactions between the service user and the service provider. Information is passed between the service user and the service provider by service primitives, which may convey parameters.

5 Document overview

Figure 1 illustrates implementations of ISO 14229-2 onto various protocols.

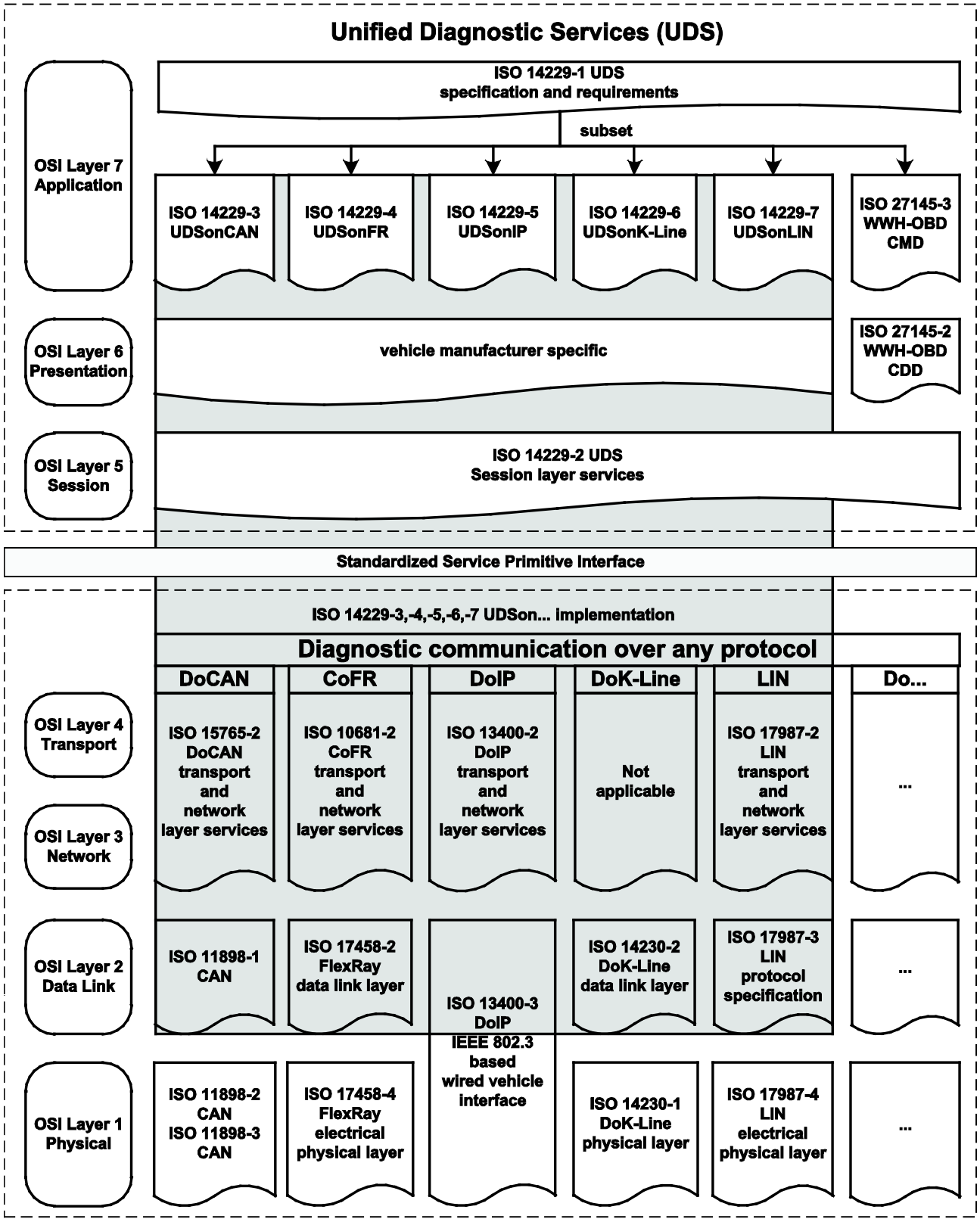


Figure 1 — Implementation of UDS document reference according to OSI model

6 Session layer services

6.1 General

The service interface defines a set of services that are needed to access the functions offered by the session layer, i.e. transmission/reception of data and setting of protocol parameters.

All session layer services have the same general structure. The service primitives define how a service user (e.g. diagnostic application) cooperates with a service provider (e.g. session layer). To define the services, three types of service primitives are specified:

— a service request primitive S_Data.request, used by the higher application layer to pass control information or data required to be transmitted to the session layer (i.e. the service provider is being requested by the service user to process control information or to transmit data);

— a service indication primitive S_Data.indication, used by the session layer to pass status information and received data to the higher application layer (i.e. the service user is being informed by the service provider about an internal event of the session layer or the service request of a peer protocol layer entity service user);

— a service confirmation primitive S_Data.confirm used by the session layer to pass status information to the application layer (i.e. the service user is being informed by service provider about the result of a preceding service request of the service user);

All session layer services have the same general format. Service primitives are written in the form:

```
service_name.type   (
                     parameter A,
                     parameter B,
                     parameter C
                    [, parameter X, ...]
                    )
```

Where:

— "service_name" is the name of the service (e.g. S_Data),

— "type" indicates the type of the service primitive (e.g. request, indication, confirm),

— "parameter A, ..." is the S_PDU (Session layer Protocol Data Unit) as a list of values passed by the service primitive (e.g. addressing information, Data, Length, Result),

— "parameter A, parameter B, parameter C" are mandatory parameters that shall be included in all service calls,

— "[parameter X]" is an optional parameter that is included if specific conditions are fulfilled.

Figure 2 shows the session layer service primitives for a single frame message.

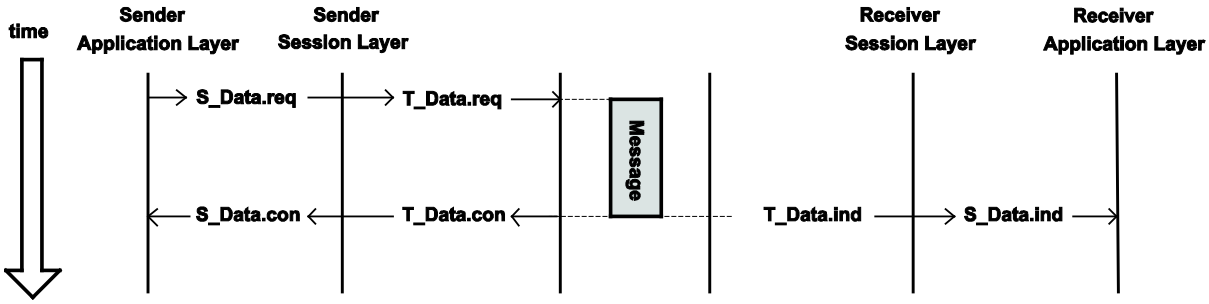
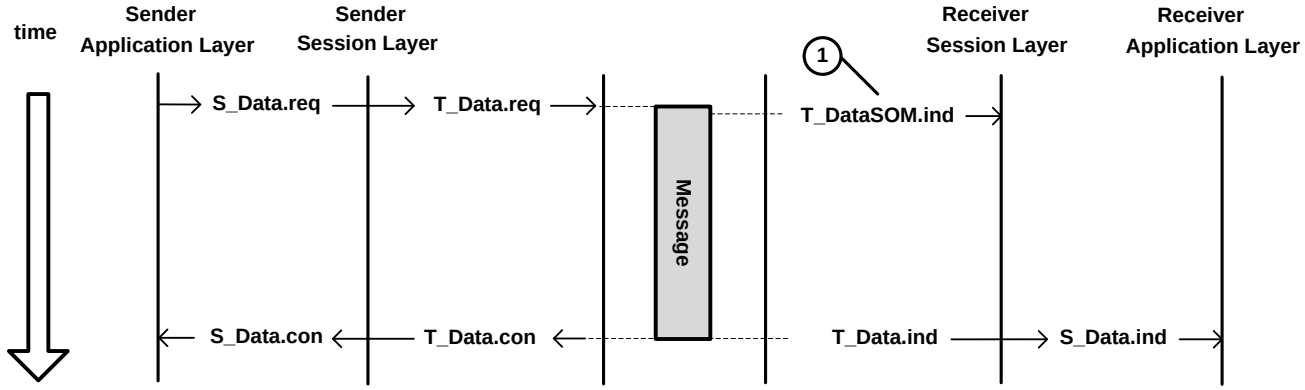


Figure 2 — Session layer service primitives – Single frame message

Figure 3 shows the session layer service primitives for a multiple frame message, if the transport/network layer supports the T_DataSOM.ind interfaces.



Key

1 optional, i.e. the transport/network layer supports the T_DataSOM.ind interfaces

Figure 3 — Session layer service primitives – Multiple frame message

The following communication scenarios shall be distinguished:

a) physical communication during

   1) default session, and

   2) non-default session — session handling required;

b) functional communication during

   1) default session, and

   2) non-default session — session handling required.

For all cases, the possibility of requesting an enhanced response-timing window by the server via a negative response message, including a negative response code 0x78, shall be considered. The transport/network layer services as defined in different ISO standards (e.g. ISO 15765-2 DoCAN or ISO 10681-2 CoFR) are used to perform the diagnostic session management timing in the client and the server.

6.2 Specification of session layer service primitives

6.2.1 General

In order to describe the function of the session layer, services provided to higher layers and the internal operations of the session layer have to be considered.

6.2.2 S_Data.request

The service primitive requests transmission of S_Data with S_Length number of bytes from the sender to the receiver peer entities identified by the address information in S_SA, S_TA, S_TAtype, and S_AE. Each time the S_Data.request service is called, the session layer shall signal the completion (or failure) of the message transmission to the service user by means of the issuing of an S_Data.confirm service call.

```
S_Data.request      (
                     S_Mtype,
                     S_SA,
                     S_TA,
                     S_TAtype,
                     [S_AE],
                     S_Data [Data#1, Data#2, …, Data#n ],
                     S_Length
                    )
```

6.2.3 S_Data.confirm

The S_Data.confirm service is issued by the session layer. The service primitive confirms the completion of an S_Data.request service identified by the address information in S_SA, S_TA, S_TAtype, and S_AE. The parameter S_Result provides the status of the service request.

```
S_Data.confirm      (
                     S_Mtype,
                     S_SA,
                     S_TA,
                     S_TAtype,
                     [S_AE],
                     S_Result
                    )
```

6.2.4 S_Data.indication

The S_Data.indication service is issued by the session layer. The service primitive indicates S_Result events and delivers S_Data with S_Length bytes received from a peer protocol entity identified by the address information in S_SA, S_TA, S_TAtype to the adjacent upper layer.

The parameters S_Data and S_Length are only valid if S_Result equals S_OK.

```
S_Data.indication   (
                     S_Mtype,
                     S_SA,
                     S_TA,
                     S_TAtype,
                     [S_AE],
                      S_Data [Data#1, Data#2, …, Data#n],
                     S_Length,
                     S_Result
                    )
```

6.3 Session data unit specification

6.3.1 S_Mtype, Session layer message type

Type: enumeration

Range: diagnostics, remote diagnostics

Description:

The parameter Mtype shall be used to identify the type and range of address information parameters included in a service call. This part of ISO 14229 specifies a range of two values for this parameter. The intention is that users of the document can extend the range of values by specifying other types and combinations of address information parameters to be used with the transport/network layer protocol specified in this document. For each such new range of address information, a new value for the Mtype parameter shall be specified to identify the new address information.

— If S_Mtype = diagnostics, then the address information shall consist of the parameters S_SA, S_TA, and S_TAtype.

— If S_Mtype = remote diagnostics, then the address information shall consist of the parameters S_SA, S_TA, S_TAtype, and S_AE.

6.3.2 S_SA, Session layer source address

Type: 2 byte unsigned integer value

Range: 0x0000 – 0xFFFF

Description:

S_SA parameter shall be used to encode the sending session layer protocol entity. The parameter S_SA shall be used to encode client and server identifiers.

6.3.3 S_TA, Session layer target address

Type: 2 byte unsigned integer value

Range: 0x0000 – 0xFFFF

Description:

S_TA parameter shall be used to encode the receiving session layer protocol entity. The parameter S_TA shall be used to encode client and server identifiers.

6.3.4 S_TAtype, Session layer target address type

Type: enumeration

Range: physical, functional

Description:

The parameter S_TAtype is a configuration attribute to the S_TA parameter. It shall be used to encode the communication model used by the communicating peer entities of the communication layer. Two communication models are specified: '1 to 1' communication, called physical addressing, and '1 to n' communication, called functional addressing.

— Physical addressing (1-to-1 communication) shall be supported for all types of session layer messages.

— Functional addressing (1-to-n communication) shall be supported. The transport/network layer requirements may restrict the usage of functional addressing (e.g. SingleFrame on CAN data link layer).

6.3.5 S_AE, Session layer Address Extension (optional parameter)

Type: 2 byte unsigned integer value

Range: 0x0000 - 0xFFFF

Description:

The S_AE parameter is used to extend the available address range for large networks, and to encode both sending and receiving transport/network layer entities of subnets other than the local network where the communication takes place. S_AE is only part of the addressing information if Mtype is set to remote diagnostics.

6.3.6 S_Length

Type: 4 Byte

Range: 0x0000 0000 – 0xFFFF FFFF

Description: This parameter includes the length of data to be transmitted/received.

6.3.7 S_Data

Type: string of bytes

Range: not applicable

Description: This parameter includes all data to be exchanged by the higher layer entities.

6.3.8 S_Result

Type: enumeration

Range: S_OK, S_NOK

Description: This parameter contains the status related to the outcome of a service execution.

6.3.9 Mapping of S_PDU onto T_PDU and vice versa for message transmission

The parameters of the session layer protocol data unit defined to request the transmission of a diagnostic service request/response are mapped as follows onto the parameters of the transport/network layer protocol data unit for the transmission of a message in the client/server.

The parameters of the transport/network layer protocol data unit defined for the reception of a message are mapped as follows onto the parameters of the session layer protocol data unit for the indication of the reception of a diagnostic response/request.

The transport/network layer confirmation of the successful transmission of the message (T_Data.con) is forwarded to the application, because it is needed in the application for starting those actions, which shall be executed immediately after the transmission of the request/response message (e.g. ECUReset, BaudrateChange, etc.).

The transport/network layer indication for the reception of a StartOfMessage T_PDU (T_DataSOM.ind) is not forwarded to the application layer, because it is only used within the session layer to perform the session layer timing (see clause 7). Therefore, no mapping of the T_DataSOM.ind T_PDU onto an S_PDU is defined.

Table 2 defines the mapping of Session layer S_PDU onto Transport/Network layer T_PDU and vice versa.

Table 2 — Mapping of Session layer S_PDU onto Transport/Network layer T_PDU and vice versa

S_PDU parameter (Session layer Protocol Data Unit)	Description	T_PDU parameter (Transport/Network layer Protocol Data Unit)	Description
S_Mtype	Session layer Message type	T_Mtype	Transport/Network layer Message type
S_SA	Session layer Source Address	T_SA	Transport/Network layer Source Address
S_TA	Session layer Target Address	T_TA	Transport/Network layer Target Address
S_TAtype	Session layer Target Address type	T_TAtype	Transport/Network layer Target Address type
S_AE [a]	Session layer Address Extension	T_AE[a]	Transport/Network layer Address Extension
S_Data[1] – S_Data[n]	Session layer Data	T_Data[1] – T_Data[n]	Transport/Network layer Application Data
S_Length	Session layer Data Length	T_Length	Transport/Network layer Data Length
S_Result	Session layer Result	T_Result	Transport/Network layer Result

[a] If Mtype = diagnostics, then the address information shall consist of the parameters SA, TA, and TAtype. If Mtype = remote diagnostics, then the address information shall consist of the parameters SA, TA, TAtype, and AE.

7 Timing parameter definition

7.1 General application timing considerations

7.1.1 Server

A server uses a single application timer ($P2_{Server}$) implementation which is triggered (started and stopped) by the T_Data service primitive interface (T_Data.ind, T_Data.con, T_Data.req).

The $P2_{Server}$ application timer is loaded with a $P2_{Server\_max}$/$P2^*_{Server\_max}$ parameter value. Both parameters and values are specified in this part of ISO 14229 (see definition in Table 3 and parameter value in Table 4).

The timing parameter $P4_{Server}$ is the time between the reception of a request (T_Data.indication) and the start of transmission of the final response (T_Data.request). A final response is a positive response or a negative response other than negative response code 0x78 "requestCorrectlyReceived-ResponsePending". In case of a request to schedule periodic responses, the initial USDT positive or negative response that indicates the acceptance or non-acceptance of the request to schedule periodic responses shall be considered the final response. $P4_{Server}$ is a performance requirement. $P4_{Server\_max}$ is the maximum value of $P4_{Server}$. If $P4_{Server\_max}$ is the same as $P2_{Server\_max}$, this means that a negative response with negative response code 0x78 is not allowed for that service or data.

These requirements are applicable only to services that are supported by the server/ECU. Unsupported services shall always utilize a $P4_{Server\_max}$ value equal to $P2_{Server\_max}$ (i.e., NRC 0x78 not allowed).

7.1.2 Client

A client uses a single application timer (P_{Client}) implementation which is triggered (started, reloaded, and stopped) by the T_Data service primitive interface (T_Data.con, T_DataSOM.ind, T_Data.ind).

The P_{Client} application timer is always loaded with a $P2_{Client\_max}$/$P2^*_{Client\_max}$ for all protocols which in principle support a T_DataSOM.ind service primitive (e.g. DoCAN). The P_{Client} application timer is always loaded with a $P6_{Client\_max}$/$P6^*_{Client\_max}$ parameter value for all protocols which in principle support a T_Data.ind service primitive only (e.g. DoIP).

The P_{Client} application timer is started, whenever the client application layer receives a T_Data.con service primitive. Depending on the protocol type (with T_DataSOM.ind or without T_DataSOM.ind) it is loaded with a $P2_{Client\_max}$ or a $P6_{Client\_max}$ parameter value.

Depending on the protocol type (with T_DataSOM.ind or without T_DataSOM.ind) the P_{Client} application timer is stopped, either when the T_DataSOM.ind or the T_Data.ind service primitive is received by the application.

For protocols which support T_DataSOM.ind the client application verifies the correct application timing by comparing its actual P_{Client} application timer with the $P2_{Client\_max}$ parameter value. If T_DataSOM.ind or T_Data.ind is received while P_{Client} is smaller or equal to $P2_{Client\_max}$ the timing fulfils the requirements established by this standard. If no .ind is received while P_{Client} is smaller or equal to $P2_{Client\_max}$ an error condition is detected. This shall be flagged to the application layer with the parameters included in either T_DataSOM.ind or the T_Data.ind service primitive.

For protocols which support T_Data.ind only the client application verifies the correct application timing by comparing its actual P_{Client} application timer with the $P6_{Client\_max}$ parameter value. If T_Data.ind is received while P_{Client} is smaller or equal to $P6_{Client\_max}$ the timing fulfils the requirements established by this part of ISO 14229. If no .ind is received while P_{Client} is smaller or equal to $P6_{Client\_max}$ an error condition is detected. This shall be flagged to the application layer with the parameters included in the T_Data.ind service primitive.

All parameters and values are specified in this part of ISO 14229 (see definition in Table 3 and parameter values in Table 4).

7.2 Application timing parameter definitions – defaultSession

A server shall always start the defaultSession when powered up. If no other diagnostic session is started, then the defaultSession shall be run as long as the server is powered. The timing parameter definitions for the defaultSession shall be in accordance with Table 3 and the definitions for the values shall be in accordance with Table 4.

Table 3 — Message timing parameter definitions for the defaultSession

Timing parameter	Description	Type
ΔP2	The ΔP2 parameter is defined to be the worst case vehicle network design-dependent message transmission delay such as delays introduced by gateways and bus-load dependent arbitration. The value of ΔP2 is divided into the time to transmit the request to the addressed server/ECU ($\Delta P2_{request}$) and in case the protocol supports T_DataSOM.ind till the start of the response transmission indicated by T_DataSOM.ind or T_Data.ind if the response is a single frame message (e.g. ISO 15765 DoCAN).	Performance requirement
ΔP6	The ΔP6 parameter is defined to be the worst case vehicle network design-dependent message transmission delay such as delays introduced by gateways and bus-load dependent arbitration. The value of ΔP6 is divided into the time to transmit the request to the addressed server/ECU ($\Delta P6_{request}$) and the time to transmit the complete response to the client/tester ($\Delta P6_{response}$). The ΔP6 is independent of whether a protocol supports a T_DataSOM.ind (e.g. ISO 15765 DoCAN) or does not support a T_DataSOM.ind (e.g. ISO 13400 DoIP).	Performance requirement
$P2_{Server}$	Performance requirement for the server to start with the response message after the reception of a request message (indicated via T_Data.ind).	Performance requirement
$P2_{Client}$	Timeout for the client to wait after the successful transmission of a request message (indicated via T_Data.con) for the start of incoming response messages (indicated via T_DataSOM.ind of a multi-frame message or T_Data.ind of a SingleFrame message).	Timer reload value
$P6_{Client}$	Timeout for the client to wait after the successful transmission of a request message (indicated via T_Data.con) for the complete reception of the corresponding response message (indicated via T_Data.ind) e.g. ISO 13400 DoIP.	Timer reload value
$P2^*_{Server}$	Performance requirement for the server to start with the response message after the transmission of a negative response message (indicated via T_Data.con) with negative response code 0x78 (enhanced response timing).	Performance requirement
$P2^*_{Client}$	Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 0x78 (indicated via T_Data.ind) for the start of incoming response messages (indicated via T_DataSOM.ind of a multi-frame message or T_Data.ind of a SingleFrame message).	Timer reload value
$P6^*_{Client}$	Enhanced timeout for the client to wait after the reception of a negative response message with negative response code 0x78 (indicated via T_Data.ind) for the complete reception of the corresponding response messages (indicated via T_Data.ind) e.g. ISO 13400 DoIP.	Timer reload value
$P3_{Client\_Phys}$	Minimum time for the client to wait after the successful transmission of a physically addressed request message (indicated via T_Data.con) with no response required before it can transmit the next physically addressed request message (see Figure 19).	Timer reload value
$P3_{Client\_Func}$	Minimum time for the client to wait after the successful transmission of a functionally addressed request message (indicated via T_Data.con) before it can transmit the next functionally addressed request message in case no response is required or the requested data is only supported by a subset of the functionally addressed servers (see 8.3).	Timer reload value
$P4_{Server}$	This is the time between the reception of a request (T_Data.indication) and the start of the transmission of the final response (T_Data.request) at the server side.	Performance requirement

Each server/ECU shall be able to process a new request message immediately after the successful transmission of a response message (T_Data.con) from the preceding request message. An exception to this requirement may be granted by the vehicle manufacturer for some use cases, where the server/ECU requires additional time after the execution of the previous service request e.g. EcuReset service.

Table 4 — Message timing parameter value definitions for the defaultSession

Timing parameter	Minimum	Maximum
$\Delta P2$	0 ms	vehicle manufacturer specific value $\Delta P2_{request} + \Delta P2_{response}$
$\Delta P6$	0 ms	vehicle manufacturer specific value $\Delta P6_{request} + \Delta P6_{response}$
$P2_{Server}$	0 ms	server specific value recommended value: 50 ms
$P2_{Client}$	$P2_{Server\_max} + \Delta P2_{max}$	--- [a)]
$P6_{Client}$	$P2_{Server\_max} + \Delta P6_{max}$	--- [a)]
$P2^*_{Server}$	0 ms [b)]	server specific value recommended value: 5 000 ms
$P2^*_{Client}$	$P2^*_{Server\_max} + \Delta P2_{response}$	--- [c)]
$P6^*_{Client}$	$P2^*_{Server\_max} + \Delta P6_{response}$	--- [c)]
$P3_{Client\_Phys}$	$P2_{Server\_max} + \Delta P2_{max}$	--- [d)]
$P3_{Client\_Func}$	$P2_{Server\_max} + \Delta P2_{max}$	--- [d)]
$P4_{Server}$	$P2_{Server}$	As defined by leglislation for OBD diagnostic use cases. Vehicle manufacturer specific value for enhanced diagnostic use cases.

a The maximum time a client waits for a response message is at the discretion of the client, provided that $P2_{Client}$ / $P6_{Client}$ is greater than the specified minimum value of $P2_{Client}$ / $P6_{Client}$.

b During the enhanced response timing, the minimum time between the transmission of consecutive negative messages (each with negative response code 0x78) shall be 0.3 * $P2^*_{Server\_max}$, in order to avoid flooding the data link with unnecessary negative response code 0x78 messages.

c The maximum value that a client uses for $P2^*_{Client}$ / $P6^*_{Client}$ is at the discretion of the client, provided it is greater than the specified minimum value of $P2^*_{Client}$ / $P6^*_{Client}$.

d The maximum time a client waits until it transmits the next request message is at the discretion of the client, provided that for non-default sessions the $S3_{Server}$ timing is kept active in the server(s).

The parameter $\Delta P2/\Delta P6$ considers any system network design-dependent delays such as delays introduced by gateways and bus bandwidth plus a safety margin (e.g. 50 % of worst case). The worst-case scenario (transmission time necessary for one “round trip” from client to server and back from server to client), based on system design, is impacted by

— the number of gateways involved,

— frame transmission time (baud rate),

— bus utilization, and

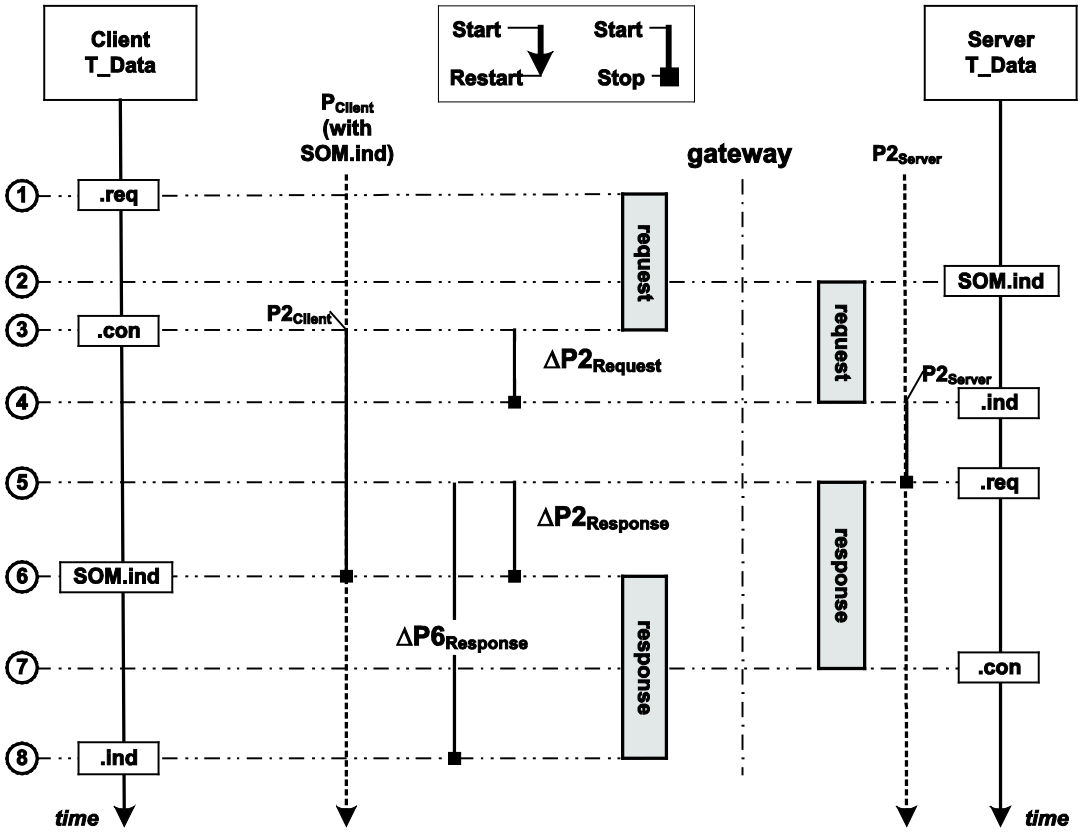
— the device driver implementation method (polling vs. interrupt) and processing time of the transport/network layer.

The value of $\Delta P2/\Delta P6$ is divided into the time to transmit the request to the addressed server and the time to transmit the response to the client:

$$\Delta P2 = \Delta P2_{Request} + \Delta P2_{Response}$$

$$\Delta P6 = \Delta P6_{Request} + \Delta P6_{Response}$$

Figure 4 and Figure 5 provides an example of how ΔP2 and ΔP6 can be composed.



Key

1 **Client T_Data.req**: diagnostic application issues a request message to the transport/network layer.

2 **Server T_DataSOM.ind**: transport/network layer issues to diagnostic application the StartOfMessage of the request message.

3 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

4 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

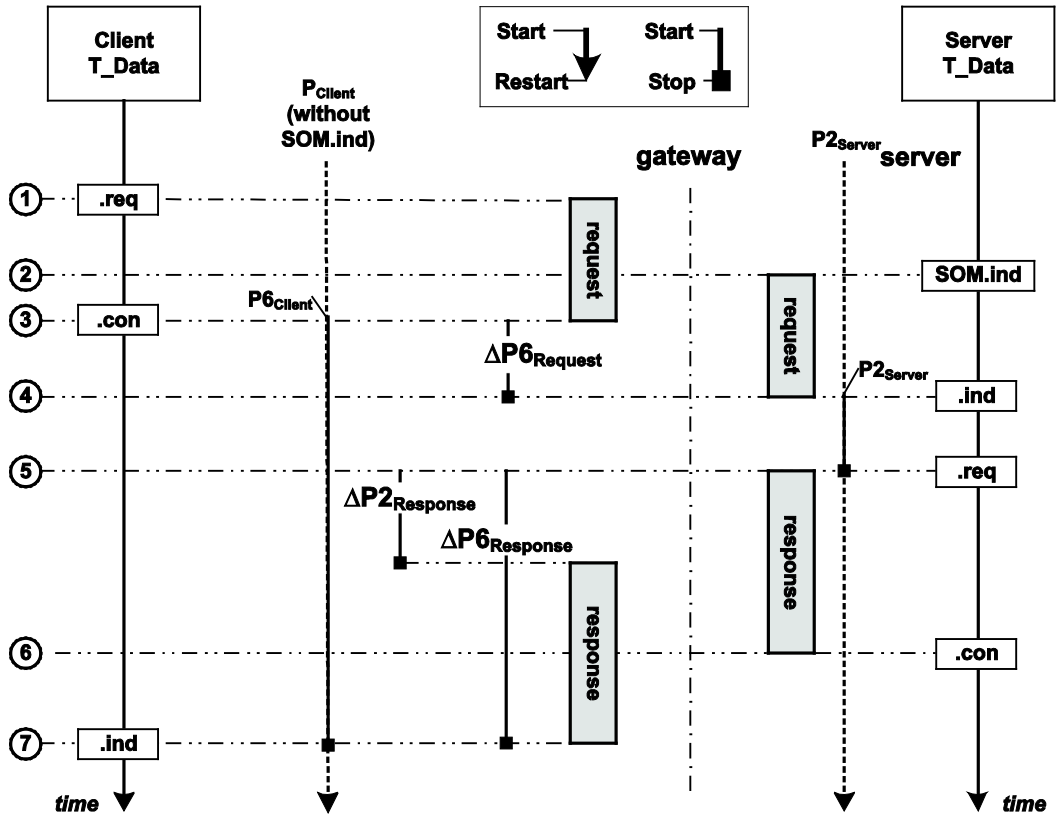
5 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

6 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the StartOfMessage of the response message. Client stops its P_{Client} timer.

7 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

8 **Client T_Data.ind**: transport/network layer issues a T_Data.ind to diagnostic application to confirm the completion of the response message.

Figure 4 — Example for ΔP2 and ΔP6 – Response message with SOM.ind



Key

1 **Client T_Data.req**: diagnostic application issues a request message to the transport/network layer.

2 **Server T_DataSOM.ind**: transport/network layer issues to diagnostic application the StartOfMessage of the request message.

3 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P6_{Client} = P6_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

4 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

5 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

6 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

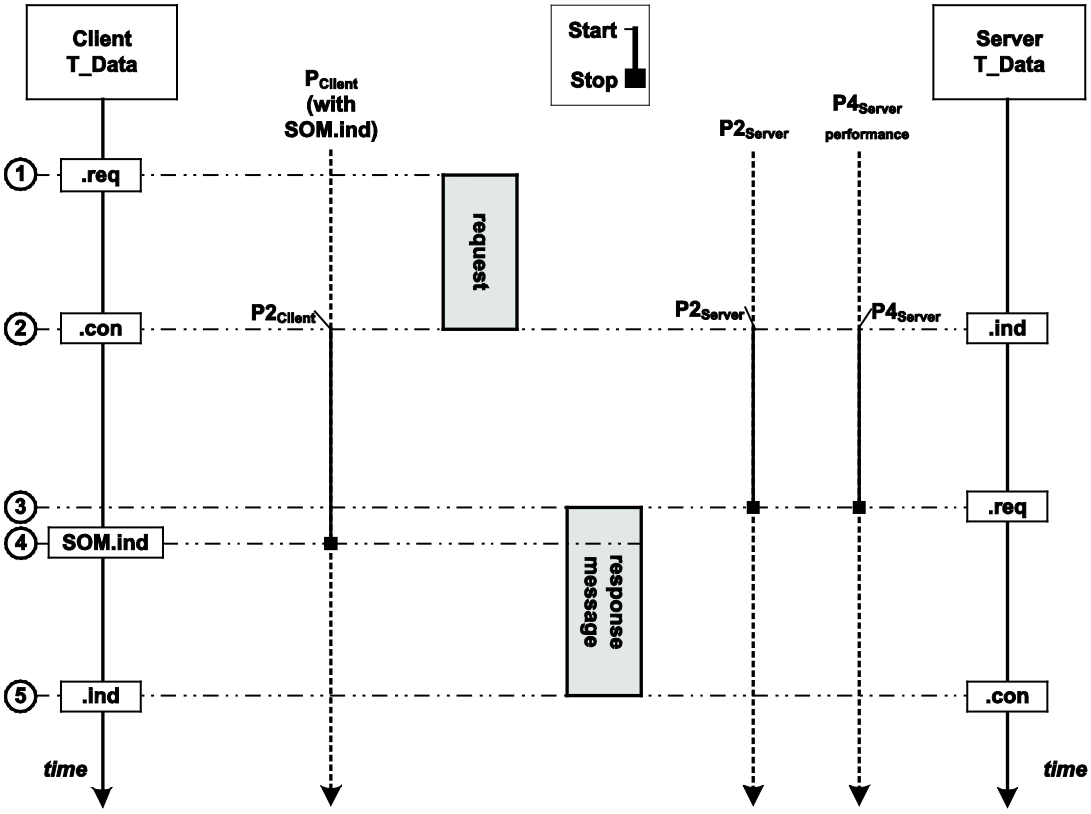
7 **Client T_Data.ind**: transport/network layer issues a T_Data.ind to diagnostic application to confirm the completion of the response message. Client stops its P_{Client} timer.

Figure 5 — Example for ΔP2 and ΔP6 – Response message without SOM.ind

NOTE For the sake of simplicity in describing the timing parameters, in all the figures that follow it is assumed that the client and the server are located on the same network. All descriptions and figures are presented in a time-related sequential order.

7.3 Example for P4Server without enhanced response timing

Figure 6 shows an example where $P4_{Server} = P2_{Server}$. In this scenario the server response performance timing parameter indicates that no negative responses including NRC 0x78 are allowed.



Key

1 **Client T_Data.req**: diagnostic application issues a request message to the transport/network layer.

2 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$ and the $P4_{Server}$ timer using the default value of $P4_{Server} = P4_{Server\_max}$.

If $P4_{Server} = P2_{Server}$ applies for a certain T_Data.ind, the server shall ensure that the final positive or negative response is started prior to $P2_{Server}$ timer expiration (i.e. no negative responses with NRC 0x78 are allowed).

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

3 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

The $P4_{Server}$ performance timer is stopped.

4 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. Client stops the P_{Client} timer.

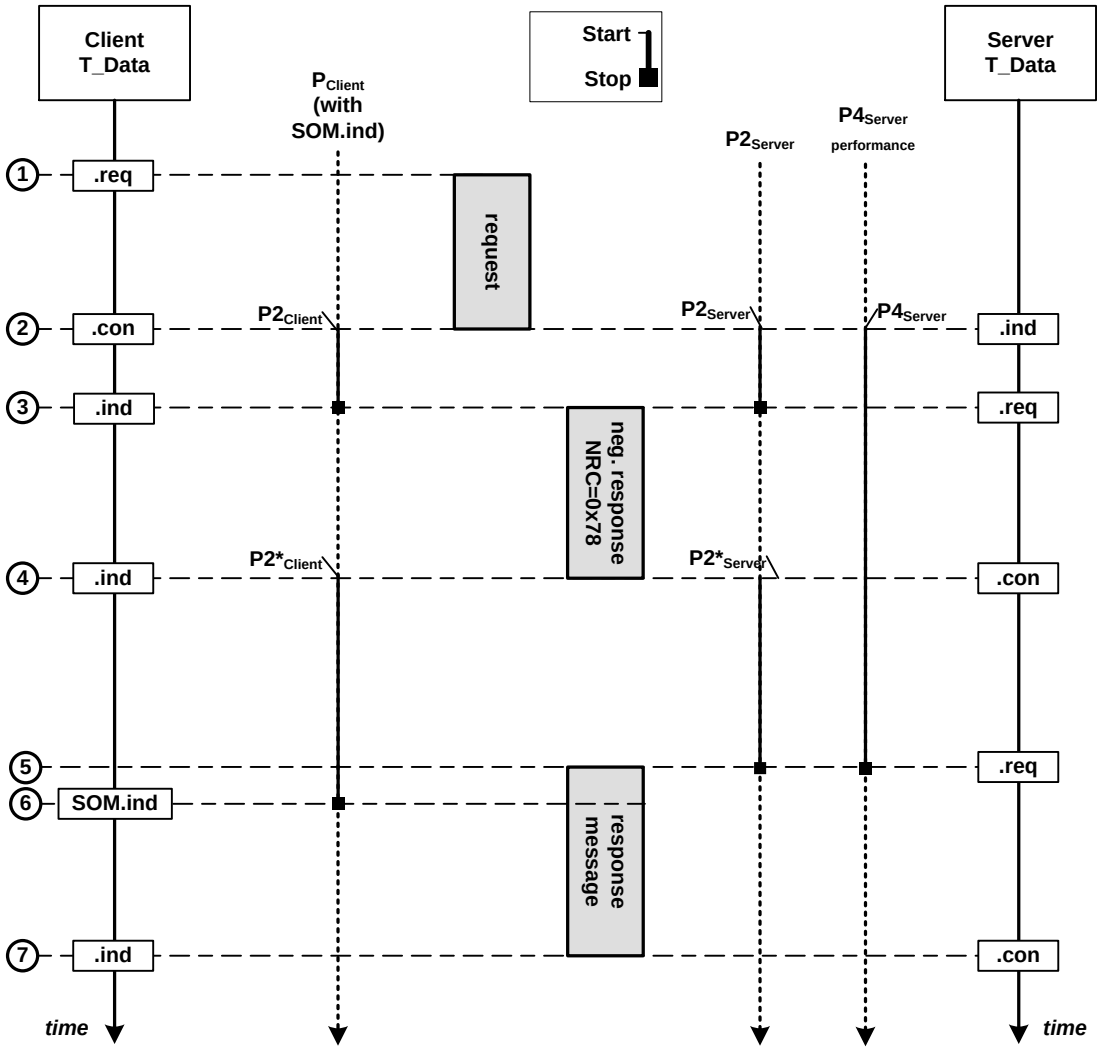
5 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

Figure 6 — Example for $P4_{Server}$ without enhanced response timing

7.4 Example for P4Server with enhanced response timing

Figure 7 shows an example where P4Server > P2Server. In this scenario the server response performance timing parameter indicates that negative responses including NRC 0x78 are allowed as long as $P4_{Server}$ is not exceeded.



Key

1 **Client T_Data.req**: diagnostic application issues a request message to the transport/network layer.

2 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$ and the $P4_{Server}$ timer using the default value of $P4_{Server} = P4_{Server\_max}$.

If $P4_{Server} > P2_{Server}$ applies for a certain T_Data.ind, the server shall ensure that the final positive or negative response is started prior to $P2_{Server}$ timer expiration (i.e. negative responses with NRC 0x78 are allowed as long as $P4_{Server}$ is not exceeded).

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

3 **Server T_Data.req**: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 0x78 by a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. Client stops the P_{Client} timer.

4 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message. Server starts the $P2_{Server}$ timer using the value of $P2^*_{Server} = P2^*_{Server\_max}$ (default enhanced timing).

In case the server can still not provide the requested information within the enhanced $P2^*_{Server}$, then a further negative response message including negative response code 0x78 can be sent by the server. This will cause the client to restart its P_{Client} timer, using the enhanced reload value $P2^*_{Client}$. For simplicity, the figure only shows a single negative response message with negative response code 0x78.

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. Client starts the P_{Client} timer with the value of $P2^*_{Client} = P2^*_{Client\_max}$ (default enhanced timing).

5 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

The $P4_{Server}$ performance timer is stopped.

6 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. Client stops the P_{Client} timer.

7 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

Figure 7 — Example for $P4_{Server}$ with enhanced response timing

7.5 Session timing parameter definitions for the non-default session

When a diagnostic session other than the defaultSession is started, then a session handling is required which is achieved via the session layer timing parameters given in Table 5.

Table 5 — Session timing parameter definitions for the non-default session

Timing parameter	Description	Type	Recommended reload ms	Timeout ms
$S3_{Client}$	Time between functionally addressed TesterPresent (0x3E) request messages transmitted by the client to keep a diagnostic session other than the defaultSession active in multiple servers (functional communication) or maximum time between physically transmitted request messages to a single server (physical communication). The $S3_{Client}$ timeout value includes the travel time of the message on the network (gateway delays, etc.).	Timer reload value	2 000	$<S3_{Server}$
$S3_{Server}$	Time for the server to keep a diagnostic session other than the defaultSession active while not receiving any diagnostic request message. The tolerance of $S3_{Server}$ is -0 ms, +200 ms.	Timer reload value	N/A	5 000

The timing parameter definitions and values as specified in Table 3 and Table 4 are also valid for the non-default session. Furthermore, the server might change its application layer timings $P2_{Server}$ and $P2^*_{Server}$ when transitioning into a non-default session in order to achieve a certain performance or to compensate restrictions which might apply during a non-default diagnostic session. The applicable timing parameters for a non-default diagnostic session are reported in the DiagnosticSessionControl positive response message in the case where a response is required to be transmitted or have to be known in advance by the client in case no response is required to be transmitted. When the client starts a non-default session functionally, then it shall adapt to the timing parameters of the responding servers.

Table 5 defines the conditions for the client and the server to start/restart its $S3_{Client}$/$S3_{Server}$ timer. For the client a periodically transmitted functionally addressed TesterPresent (0x3E) request message shall be distinguished from a sequentially transmitted physically addressed TesterPresent (0x3E) request message, which is only transmitted in case of the absence of any other diagnostic request message. For the server there is no need to distinguish between that kind of TesterPresent (0x3E) handling. Furthermore, Table 5 shows that the $S3_{Server}$ timer handling is based on the transport/network layer service primitives, which means

that the $S3_{Server}$ timer is also restarted upon the reception of a diagnostic request message that is not supported by the server.

During a non-default session, the additional timer resource requirements given in Table 6 shall apply for the client and the server.

Table 6 — Session layer timing start/stop conditions for the client and the server

<table>
<tr><th>Timing parameter</th><th>Action</th><th>Physical and functional communication, using functionally addressed, periodically transmitted TesterPresent request message</th><th>Physical communication only, using a physically addressed, sequentially transmitted TesterPresent request message</th></tr>
<tr><td rowspan="5">$S3_{Client}$</td><td rowspan="2">Initial start</td><td rowspan="2">T_Data.con that indicates the completion of the DiagnosticSessionControl (0x10) request message. This is only true if the session type is a non-default session.</td><td>T_Data.con that indicates the completion of the DiagnosticSessionControl (0x10) request message in case no response is required.</td></tr>
<tr><td>T_Data.ind that indicates the reception of the DiagnosticSessionControl (0x10) response message in case a response is required.</td></tr>
<tr><td rowspan="3">Subsequent start</td><td rowspan="3">T_Data.con that indicates the completion of the functionally addressed TesterPresent (0x3E) request message, which is transmitted each time the $S3_{Client}$ timer times out.</td><td>T_Data.con that indicates the completion of any request message in case no response is required.</td></tr>
<tr><td>T_Data.con that indicates an error during the transmission of either a single-frame or multi-frame request message.</td></tr>
<tr><td>T_Data.ind that indicates the reception of any response message in case a response is required.</td></tr>
<tr><td rowspan="6">$S3_{Server}$</td><td rowspan="2">Initial start</td><td colspan="2">T_Data.con that indicates the completion of the transmission of a DiagnosticSessionControl positive response message for a transition from the default session to a non-default session, in case a response message is required.</td></tr>
<tr><td colspan="2">Successful completion of the requested action of the service DiagnosticSessionControl (0x10) for a transition from the default session to a non-default session, in case no response message is required/allowed.</td></tr>
<tr><td>Subsequent stop</td><td colspan="2">T_DataSOM.ind that indicates the start of a multi-frame request message or T_Data.ind that indicates the reception of any SingleFrame request message. If the defaultSession is active, the $S3_{Server}$ timer is disabled.</td></tr>
<tr><td rowspan="3">Subsequent start</td><td colspan="2">T_Data.con that indicates the completion of any response message that concludes a service execution (final response message) in case a response message is required/allowed to be transmitted (this includes positive and negative response messages). A negative response with negative response code 0x78 does not restart the $S3_{Server}$ timer.</td></tr>
<tr><td colspan="2">Completion of the requested action (service conclusion) in case no response message (positive and negative) is required/allowed.</td></tr>
<tr><td colspan="2">T_Data.ind that indicates an error during the reception of a multi-frame request message.</td></tr>
<tr><td colspan="3">For further details regarding the $S3_{Server}$ handling in the server when the server is requested to transmit unsolicited response message such as periodic data or responses based on an event see the data link specific implementation documents of ISO 14229.</td></tr>
</table>

7.6 Client and server timer resource requirements

The timer resource required for the client and the server to fulfil the above given timing requirements during the default session and any non-default session shall be in accordance with the list in Table 7 and Table 8.

Table 7 defines the timer resources requirements during defaultSession and non-defaultSession.

Table 7 — Timer resources requirements during defaultSession

Timing parameter	Client	Server
P_{Client}	A single timer is required for each logical communication channel (physical and functional communication), e.g. each point-to-point communication requires a separate communication channel.	N/A
$P2_{Server}$	N/A	A single timer is required for the enhanced response timing in order to ensure that subsequent negative response messages with negative response code 0x78 are transmitted prior to the expired $P2^*_{Server}$.
$P3_{Client\_Phys}$	A single timer is required per logical physical communication channel.	N/A
$P3_{Client\_Func}$	A single timer is required per logical functional communication channel.	N/A

Table 8 defines the additional timer resources requirements during non-defaultSession.

Table 8 — Additional timer resources requirements during non-defaultSession

Timing parameter	Client	Server
$S3_{Client}$	A single timer is required when using a periodically transmitted, functionally addressed TesterPresent (0x3E) request message to keep the servers in a non-defaultSession. There is no need for additional timers per activated diagnostic sessions.	N/A
	A single timer is required for each point-to-point communication channel when using a sequentially transmitted, physically addressed TesterPresent (0x3E) request message to keep a single server in a non-defaultSession in case of the absence of another diagnostic request message then.	
$S3_{Server}$	N/A	A single timer is required in the server, because only a single diagnostic session can be active at a time in a single server.

7.7 Error handling

Error handling for the application layer and session management to be fulfilled by the client and the server during physical and functional communication shall be in accordance with Table 9 and Table 10, in respect of which it is assumed that the client and the server implement the application and session layer timing according to this part of ISO 14229.

Table 9 — Recommendation for a generic client error handling

Communication phase	Client error type	Client handling	
		Physical communication	Functional communication
Request transmission	T_Data.con from transport/network layer with a negative result value.	The client shall repeat the last request, after the time $P3_{Client\_Phys}$ following the error indication. Restart $S3_{Client}$ in the case of a physically addressed and sequentially transmitted TesterPresent (because $S3_{Client}$t has been stopped based on the request message transmission).	The client shall repeat the last request, after the time $P3_{Client\_Func}$ following the error indication.
P_{Client} P^*_{Client}	Timeout	The client shall repeat the last request. Restart $S3_{Client}$ in the case of a physically addressed and sequentially transmitted TesterPresent (because $S3_{Client}$ has been stopped based on the request message transmission).	Where the client does not know the number of servers responding, then this is the indication for the client that no further response messages are expected. No retry of the request message is required. The client shall completely receive all response messages that are in progress until it can continue with further requests.
			Where the client knows the number of responding servers, then this is the indication for the client that not all expected servers responded. The client shall repeat the request after it has completely received any response message that is in progress at the point in time the timeout occurs.
Response reception	T_Data.ind from transport/ network layer with a negative result value.	The client shall repeat the last request. Restart $S3_{Client}$ in the case of a physically addressed and sequentially transmitted TesterPresent (because $S3_{Client}$ has been stopped based on the request message transmission).	The client shall repeat the last request after it has completely received any response message that is in progress at the point in time the error has been indicated.
The client error handling defined shall be performed for a maximum of two times, which means that the worst-case of service request transmissions is three.			

Table 10 defines the server error handling.

Table 10 — Server error handling

Communication phase	Server error type	Server handling
Request reception	T_Data.ind from transport/ network layer with a negative	Restart $S3_{Server}$ timer (because it has been stopped based on the previously received StartOfMessage indication). The

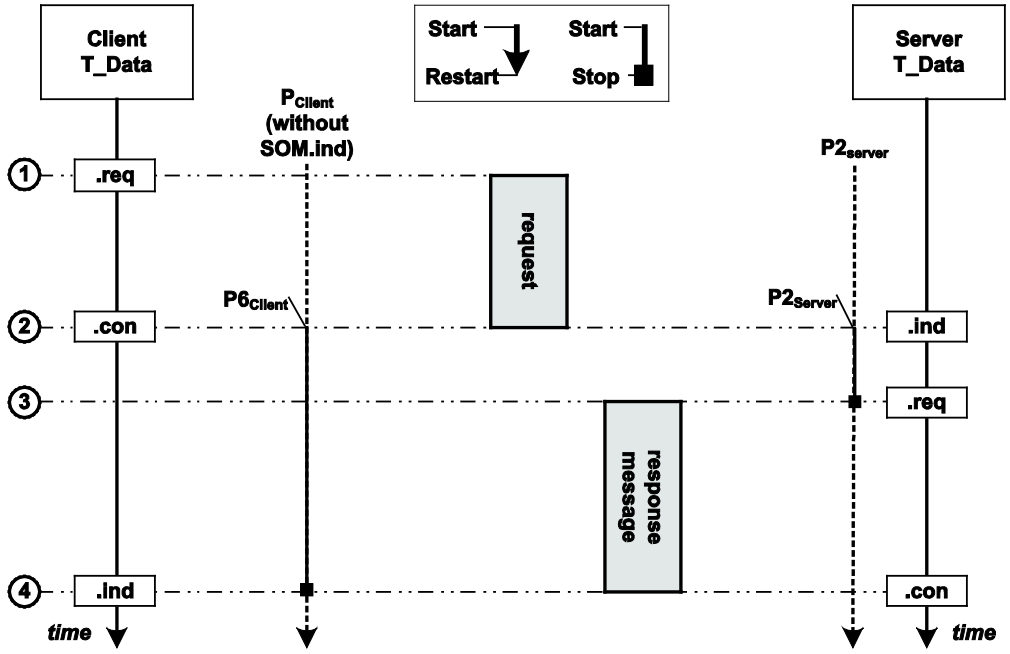
Communication phase	Server error type	Server handling
	result value.	server shall ignore the request.
Response transmission	T_Data.con from transport/ network layer with a negative result value.	Restart $S3_{Server}$ timer (because it has been stopped based on the previously received request message). The server shall *not* perform a retransmission of the response message.

8 Timing handling during communication

8.1 Physical communication

8.1.1 Physical communication during defaultSession – without SOM.ind

Figure 8 graphically depicts the timing handling in the client and the server for a physically addressed request message without SOM.ind during the default session.



Key

1 **Client T_Data.req**: diagnostic application issues a request message to the transport/network layer.

2 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P6_{Client} = P6_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.).

3 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

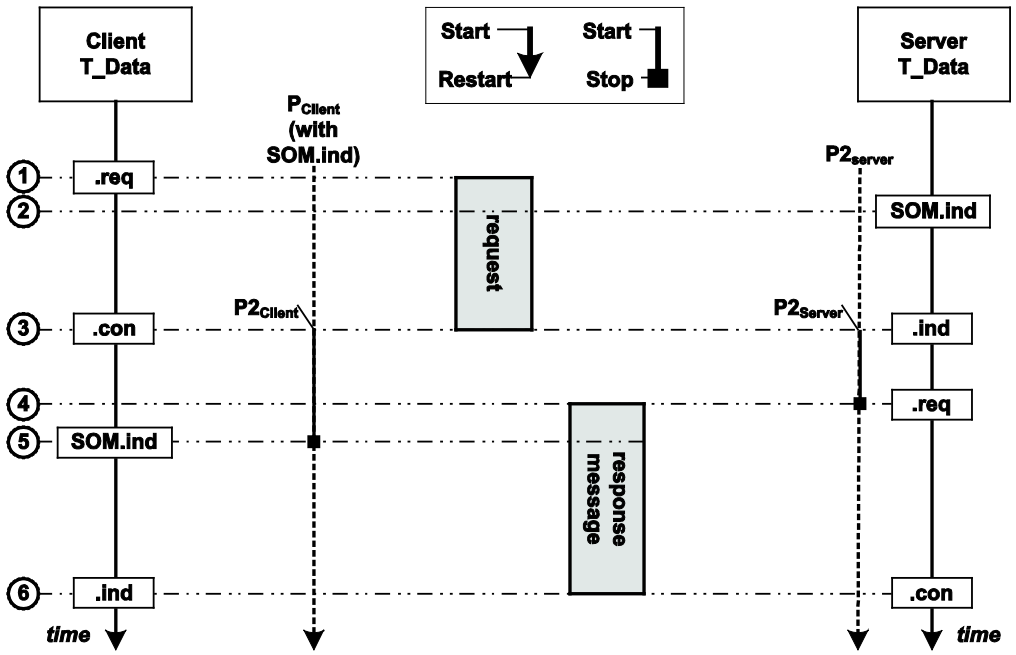
4 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. The client stops its P_{Client} timer.

Figure 8 — Physical communication during default session - without SOM.ind

8.1.2 Physical communication during defaultSession – with SOM.ind

Figure 9 graphically depicts the timing handling in the client and the server for a physically addressed request message with SOM.ind during the default session.



Key

1 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer.

2 **Server T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage of a request message.

3 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

4 **Server T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

5 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. Client stops the P_{Client} timer.

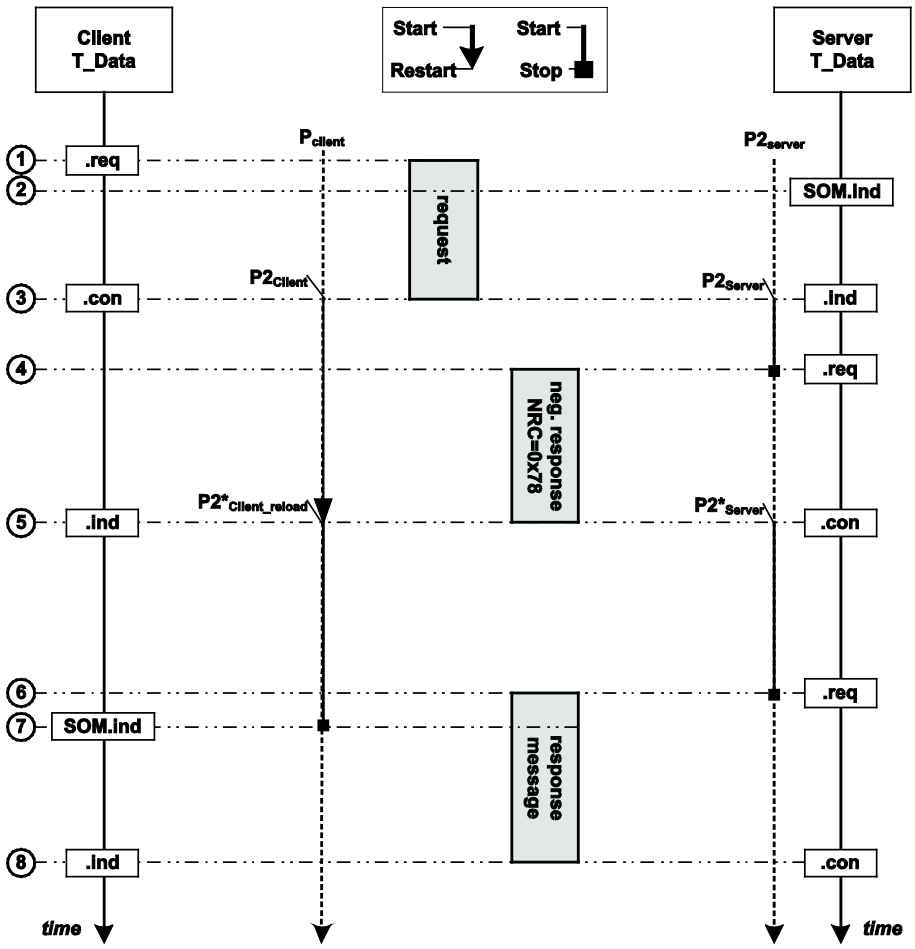
6 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

Figure 9 — Physical communication during default session – with SOM.ind

8.1.3 Physical communication during defaultSession with enhanced response timing

Figure 10 graphically depicts the timing handling in the client and the server for a physically addressed request message during the default session and the request of the server for an enhanced response timing (negative response code 0x78 handling).



Key

1 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer.

2 **Server T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage of a request message, if the transport/network layer supports the T_DataSOM.ind interfaces

3 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

4 **Server T_Data.req**: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 0x78 by a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

5 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message. Server starts the $P2_{Server}$ timer using the value of $P2_{Server} = P2^*_{Server\_max}$ (default enhanced timing).

In case the server can still not provide the requested information within the enhanced $P2^*_{Server}$, then a further negative response message including negative response code 0x78 can be sent by the server. This will cause the client to restart its P_{Client} timer, using the enhanced reload value $P2^*_{Client}$. For simplicity, the figure only shows a single negative response message with negative response code 0x78.

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. Client stops the P_{Client} timer and reloads the P_{Client} timer with the value of $P2^*_{Client} = P2^*_{Client\_max}$ (default enhanced timing).

6 **Server T_Data.req**: diagnostic application issues a response message (positive or negative response other than negative response code 0x78) to transport/network layer. Server stops the $P2_{Server}$ timer.

7 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application reception of a StartOfMessage of a response message, if the transport/network layer supports the T_DataSOM.ind interfaces. Client stops the P_{Client} timer.

8 **Server T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. When receiving this indication, the client stops its P_{Client} timer if the transport/network protocol does not support a T_DataSOM.ind interface.

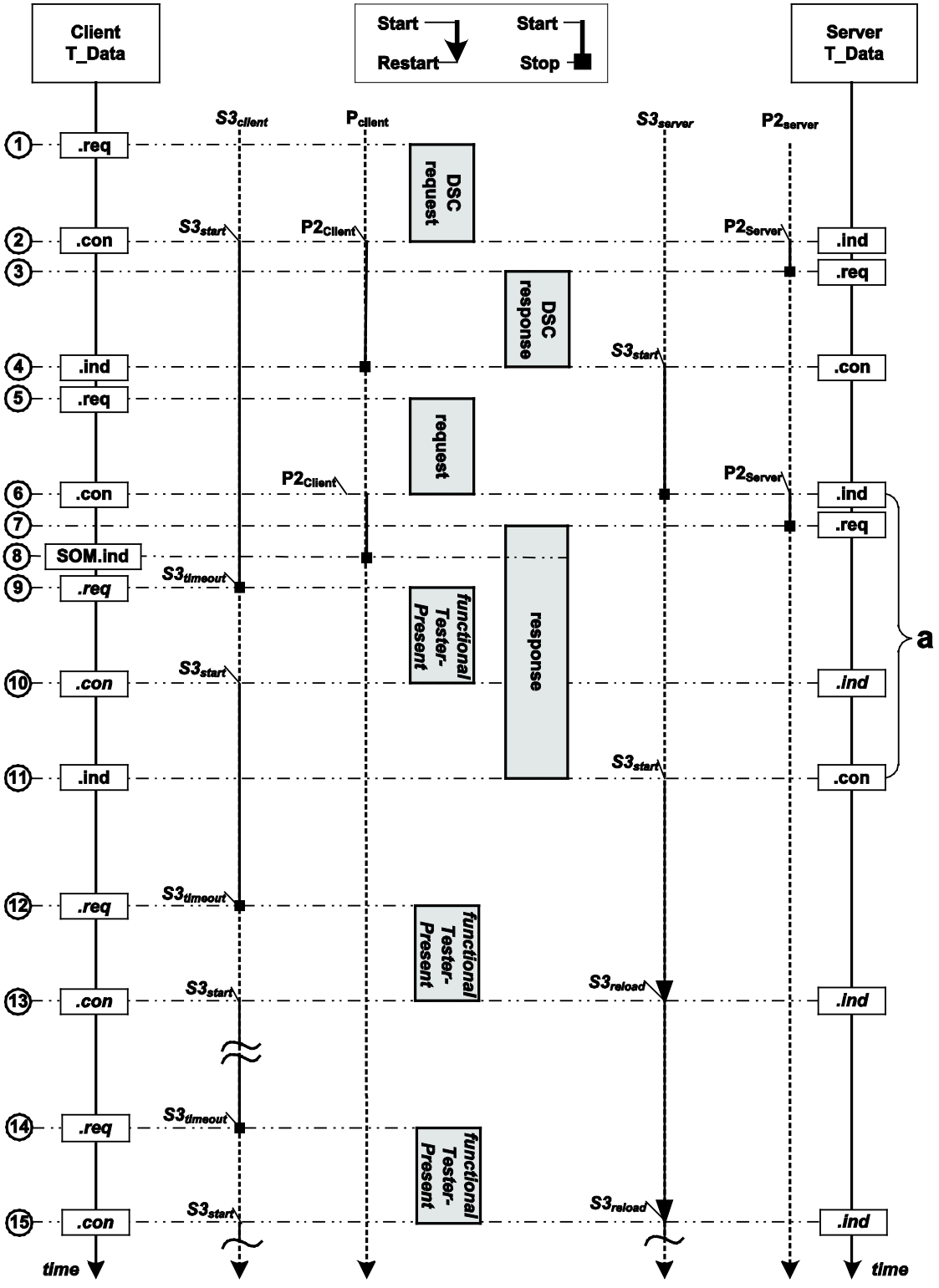
Figure 10 — Physical communication during default session – Enhanced response timing

8.1.4 Physical communication during a non-default session

8.1.4.1 Functionally addressed TesterPresent (0x3E) message

Figure 11 graphically depicts the timing handling in the client and the server when performing physical communication during a non-default session (e.g. programmingSession) and using a functionally addressed, periodically transmitted TesterPresent (0x3E) request message that does not require a response message from the server.

The handling of the P_{Client} and $P2_{Server}$ timing is identical to the handling as described in 8.1.2. The only exception is that the reload values on the client side and the resulting time where the server shall send its final response time might differ. This is based on the transition into a session other than the default session where different P_{Client} timing parameters might apply (see DiagnosticSessionControl (0x10) service in ISO 14229-1 for details on how the timing parameters are reported to the client).



Key

a Any testerPresent that is received during a disabled $S3_{Server}$ timer will be ignored by the server

1 **Client T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) request message to transport/network layer.

The transport/network layer transmits the request message to the server.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the reception of the DiagnosticSessionControl (0x10) request message. Now the response timing P_{Client} as described in 8.1.2 applies. The generated T_Data.con in the client causes the start of the $S3_{Client}$ timer (session timer). Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the DiagnosticSessionControl (0x10) request message. Now the response timing $P2_{Server}$ as described in 8.1.2 applies.

3 **Server T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) positive response message to transport/network layer. For the figure given, it is assumed that the client requires a response from the server.

4 **Server T_Data.con**: the completion of the transmission of the response message is indicated in the server via T_Data.con. Now the server starts its $S3_{Server}$ timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the $S3_{Server}$ timer is reset prior to its timeout to keep the server in the non-default session.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. Client stops the P_{Client} timer.

5 **Client T_Data.req**: diagnostic application issues a new request message to transport/network layer.

6 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message. Any time the server is in the process of handling any diagnostic service, it stops its $S3_{Server}$ timer. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the completion of the request message.

7 **Server T_Data.req**: diagnostic application issues the positive response message to transport/network.

8 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport/network layer. Client stops the P_{Client} timer.

9 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

10 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

Server T_Data.ind: any TesterPresent (0x3E) request message that is received during processing another request message can be ignored by the server, because it has already stopped its $S3_{Server}$ timer and will restart it once the service that is in progress is processed completely.

11 **Client T_Data.ind**: upon transport/network layer issues to diagnostic application the completion of the response message.

Server T_Data.con: when the diagnostic service is completely processed, then the server restarts its $S3_{Server}$ timer. This means that any diagnostic service, including TesterPresent (0x3E), resets the $S3_{Server}$ timer. A diagnostic service is considered to be in progress any time between the start of the reception of the request message (T_DataSOM.ind or T_Data.ind receive) and the completion of the transmission of the final response message, where a response message is required, or the completion of any action that is caused by the request, where no response message is required (point In time reached that would cause the start of the response message).

12 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

13 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

14 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

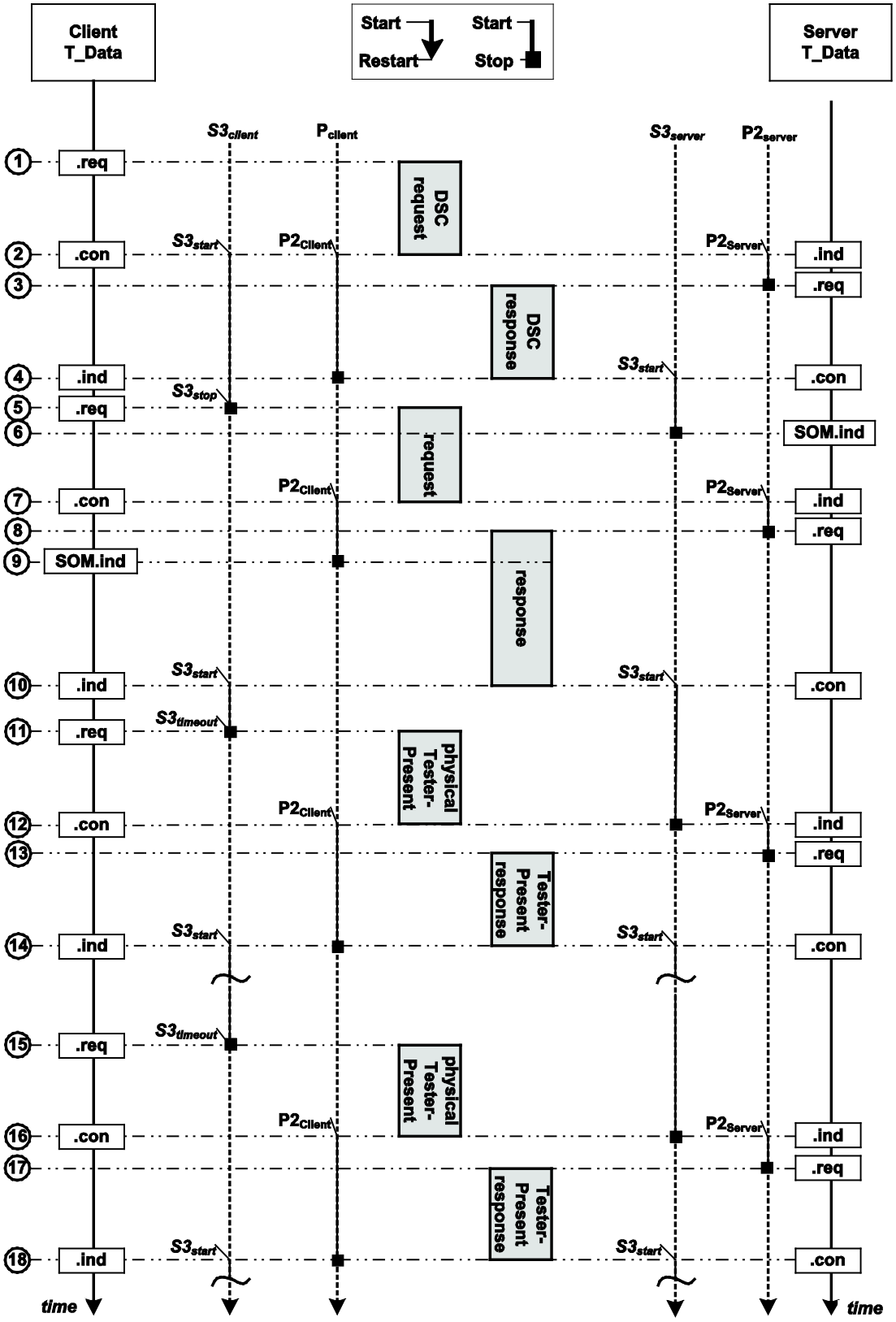
15 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

Figure 11 — Physical communication during non-default session – functionally addressed TesterPresent

8.1.4.2 Physically addressed TesterPresent (0x3E) message

Figure 12 graphically depicts the timing handling in the client and the server when performing physical communication during a non-default session (e.g. programmingSession) and using a physically addressed TesterPresent (0x3E) request message that requires a response message from the server to keep the diagnostic session active in case of the absence of any other diagnostic service.



Key

1 **Client T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) request message to transport/network layer.

The transport/network layer transmits the request message to the server.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the reception of the DiagnosticSessionControl (0x10) request message. Now the response timing P_{Client} as described in 8.1.2 applies. The generated T_Data.con in the client causes the start of the $S3_{Client}$ timer (session timer). Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the DiagnosticSessionControl (0x10) request message. Now the response timing $P2_{Server}$ as described in 8.1.2 applies.

3 **Server T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) positive response message to transport/network layer. For the figure given, it is assumed that the client requires a response from the server.

4 **Server T_Data.con**: the completion of the transmission of the response message is indicated in the server via T_Data.con. Now the server starts its $S3_{Server}$ timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the $S3_{Server}$ timer is reset prior to its timeout to keep the server in the non-default session.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. Client stops the P_{Client} timer.

5 **Client T_Data.req**: diagnostic application issues a new request message to transport/network layer. Whenever the client transmits a request message to the server (including the physically addressed TesterPresent (0x3E) message), it stops its $S3_{Client}$ timer.

6 **Server T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport/network layer. The reception of a StartOfMessage of the request message stops the $S3_{Server}$ timer in the server.

7 **Server T_Data.ind**: transport/network layer issues to diagnostic application the completion of the request message.

Client T_Data.con: transport/network layer issues to diagnostic application the completion of the request message. In the case where the client does not require a response message it shall start its $S3_{Client}$ timer when it receives confirmation of the completion of the request message, which is indicated via T_Data.con. In this case the server starts its $S3_{Server}$ timer when it has completed the requested action. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

8 **Server T_Data.req**: diagnostic application issues the positive response message to transport/network.

9 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage, if the T_DataSOM.ind interface is supported by the transport/network layer. Client stops the P_{Client} timer.

10 **Client T_Data.ind**: transport/network layer issues to diagnostic application the completion of the response message. The generated T_Data.ind in the client causes the start of the $S3_{Client}$ timer (session timer).

Server T_Data.con: transport/network layer issues to diagnostic application the completion of the response message. Now the server starts its $S3_{Server}$ timer.

11 **Client T_Data.req**: in case the client would not send any diagnostic request message prior to the timeout of $S3_{Client}$, then the timeout of the $S3_{Client}$ timer causes the client to transmit a physically addressed TesterPresent (0x3E) request message.

12 **Server T_Data.ind**: the reception of the TesterPresent (0x3E) request message is indicated in the server via T_Data.ind. This causes the server to stop its $S3_{Server}$ timer. Now the response timing as described in 8.1.2 applies.

Client T_Data.con: the completion of the TesterPresent (0x3E) request message is indicated in the client via T_Data.con. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

13 **Server T_Data.req**: diagnostic application issues the TesterPresent (0x3E) response message to transport/network.

14 **Client T_Data.ind**: the completion of the TesterPresent (0x3E) response message is indicated in the client via T_Data.ind, which causes the client to start its $S3_{Client}$ timer.

Server T_Data.con: the completion of the TesterPresent (0x3E) response message is indicated in the server via T_Data.con, which causes the server to start its $S3_{Server}$. In the case where the client would not require a response message, then it shall start its $S3_{Client}$ timer when it receives confirmation of the completion of the TesterPresent (0x3E) request message, which is indicated via T_Data.con. The server would start its $S3_{Server}$ timer when it has completed the requested action. For simplicity, the figure shows that a response is required. Client stops the P_{Client} timer.

15 **Client T_Data.req**: in case the client would not send any diagnostic request message prior to the timeout of $S3_{Client}$, then the timeout of the $S3_{Client}$ timer causes the client to transmit a physically addressed TesterPresent (0x3E) request message.

16 **Server T_Data.ind**: the reception of the TesterPresent (0x3E) request message is indicated in the server via T_Data.ind. This causes the server to stop its $S3_{Server}$ timer. Now the response timing as described in 8.1.2 applies.

Client T_Data.con: the completion of the TesterPresent (0x3E) request message is indicated in the client via T_Data.con. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

17 **Server T_Data.req**: diagnostic application issues the TesterPresent (0x3E) response message to transport/network.

18 **Client T_Data.ind**: the completion of the TesterPresent (0x3E) response message is indicated in the client via T_Data.ind, which causes the client to start its $S3_{Client}$ timer.

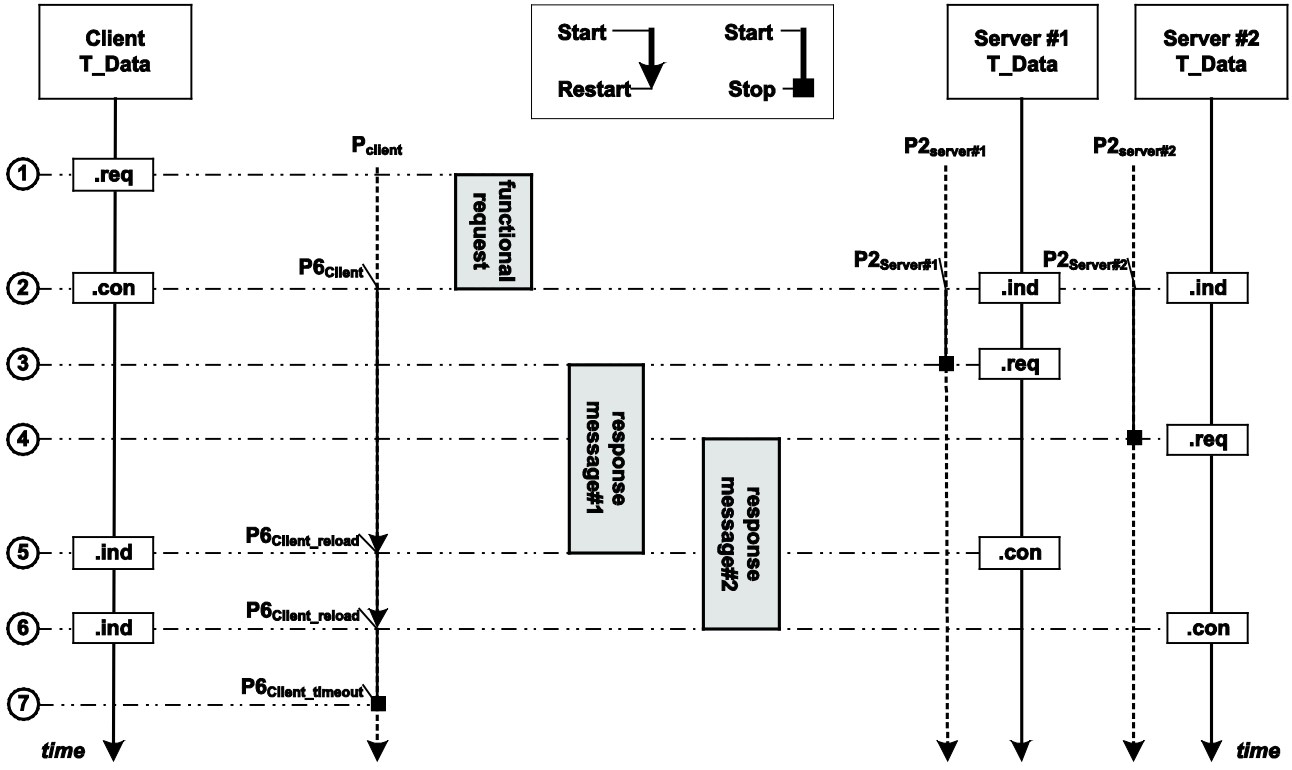
Server T_Data.con: the completion of the TesterPresent (0x3E) response message is indicated in the server via T_Data.con, which causes the server to start its $S3_{Server}$. In the case where the client would not require a response message, then it shall start its $S3_{Client}$ timer when it receives confirmation of the completion of the TesterPresent (0x3E) request message, which is indicated via T_Data.con. The server would start its $S3_{Server}$ timer when it has completed the requested action. For simplicity, the figure shows that a response is required.

Figure 12 — Physical communication during non-default session – Physically addressed TesterPresent

8.2 Functional communication

8.2.1 Functional communication during defaultSession – without SOM.ind

Figure 13 graphically depicts the timing handling in the client and two servers for a functionally addressed request message during the default session. From a server point of view, there is no difference in the timing handling compared to a physically addressed request message, but the client shall handle the timing different compared to physical communication.



Key

1 **Client T_Data.req**: diagnostic application issues functionally addressed request message to transport/network layer.

2 **All server T_Data.ind**: transport/network layer issues to diagnostic application the completion of a request message. All servers start the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P6_{Client} = P6_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

3 **Server #1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server stops the $P2_{Server}$ timer.

4 **Server #2 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server\_max}$. Server stops the $P2_{Server}$ timer.

5 **Server #1 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. This will cause the client to restart its P_{Client} timer, using the default reload value $P6_{Client\_max}$.

6 **Server #2 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

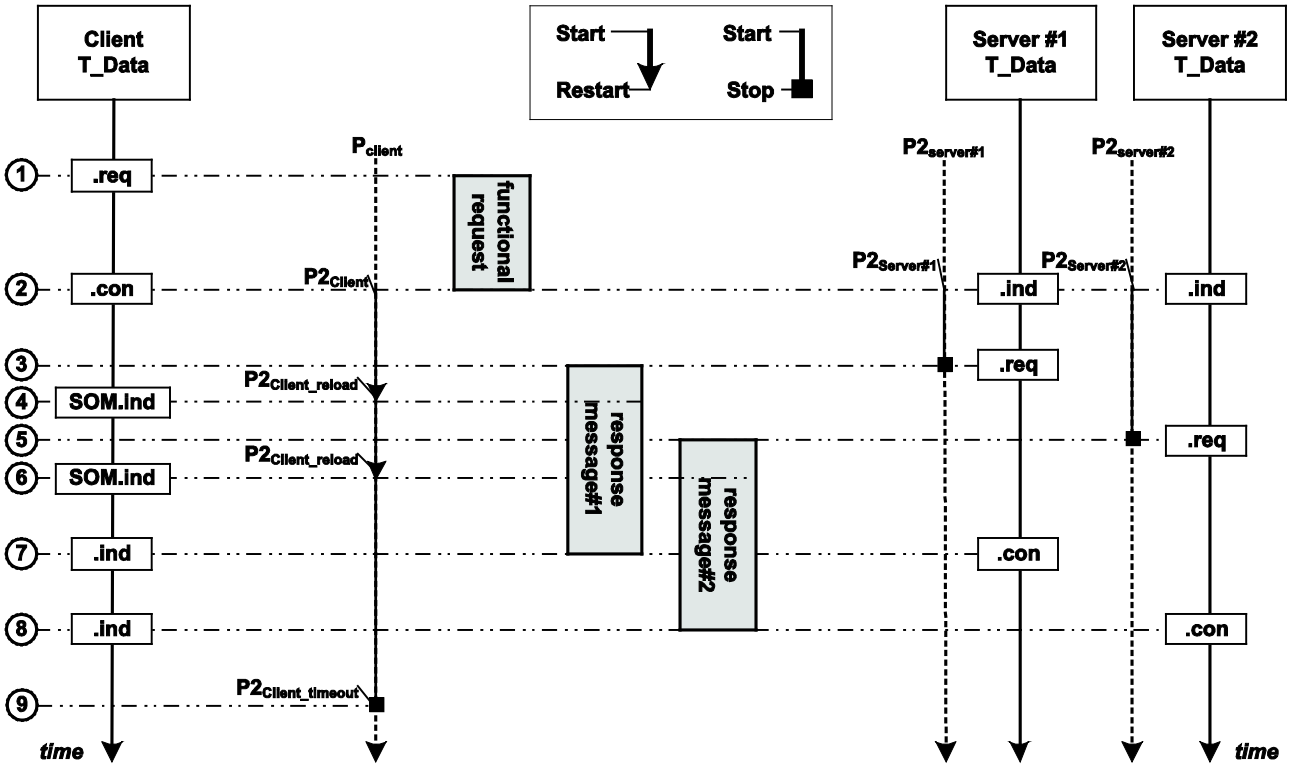
Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. This will cause the client to restart its P_{Client} timer, using the default reload value $P6_{Client\_max}$.

7 **Client**: this is the indication for the client that no further response messages are expected and can continue with further requests.

Figure 13 — Functional communication during default session – without SOM.ind

8.2.2 Functional communication during defaultSession – with SOM.ind

Figure 14 graphically depicts the timing handling in the client and two servers for a functionally addressed request message during the default session. From a server point of view, there is no difference in the timing handling compared to a physically addressed request message, but the client shall handle the timing different compared to physical communication.



Key

1 **Client T_Data.req**: diagnostic application issues functionally addressed request message to transport/network layer.

2 **All server T_Data.ind**: transport/network layer issues to diagnostic application the completion of a request message. All servers start the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

3 **Server #1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server#1 stops the $P2_{Server}$ timer.

4 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. This will cause

the client to restart its P_{Client} timer, using the default reload value $P2_{Client\_max}$.

5 **Server #2 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server\_max}$. Server#2 stops the $P2_{Server}$ timer.

6 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. This will cause the client to restart its P_{Client} timer, using the default reload value $P2_{Client\_max}$.

7 **Server #1 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

8 **Server #2 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

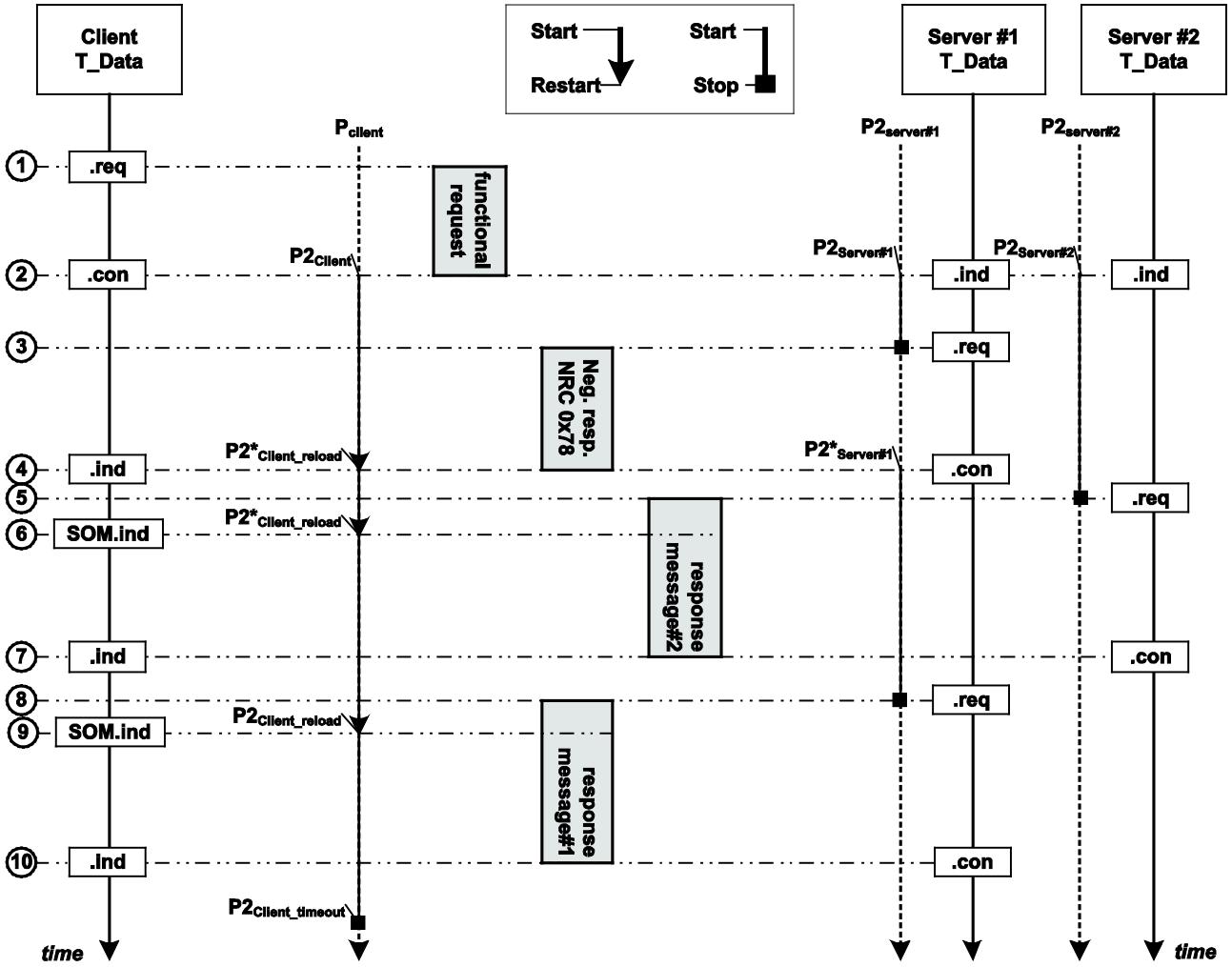
9 **Client**: this is the indication for the client that no further response messages are expected and can continue with further requests.

Figure 14 — Functional communication during default session – with SOM.ind

8.2.3 Functional communication during defaultSession with enhanced response timing – with SOM.ind

Figure 15 graphically depicts the timing handling in the client and two servers for a functionally addressed request message during the default session, where one server requests the enhanced response timing via a negative response message including negative response code 0x78.

From a server point of view there is no difference in the timing handling compared to a physically addressed request message that requires enhanced response timing, but the client shall handle the timing differently compared to physical communication.



Key

1 **Client T_Data.req**: diagnostic application issues functionally addressed request message to transport/network layer.

2 **All server T_Data.ind**: transport/network layer issues to diagnostic application the completion of a request message. All servers start the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{Server\_max}$.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The NRC 0x78 pending list is empty. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the server are located on the same network.

3 **Server #1 T_Data.req**: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 0x78 by a T_Data.req to transport/network layer within $P2_{Server}$. Server#1 stops the $P2_{Server}$ timer. In case any of the addressed servers cannot provide the requested information within the $P2_{Server}$ response timing, it can request an enhanced response timing window by sending a negative response message including negative response code 0x78.

4 **Server#1 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message. Server#1 starts the $P2_{Server}$ timer using the value of $P2^*_{Server\#1} = P2^*_{Server\_max}$ (default enhanced timing).

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. Client reloads the P_{Client} timer with the value of $P2^*_{Client} = P2^*_{Client\_max}$ (default enhanced timing). An entry containing the response message address is added to the NRC 0x78 pending list.

5 **Server#2 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server#2 stops the $P2_{Server}$ timer.

6 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. This will cause the client to restart its P_{Client} timer, using the default reload value $P2^*_{Client}$ (default enhanced timing).

7 **Server #2 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

8 **Server#1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2^*_{Server}$. Server#1 stops the $P2_{Server}$ timer.

9 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. The entry matching the response message address is removed from the pending list and the list is now empty. That means no further response messages are pending. This causes the client to restart its P_{Client} timer using the default reload value $P2_{Client}$.

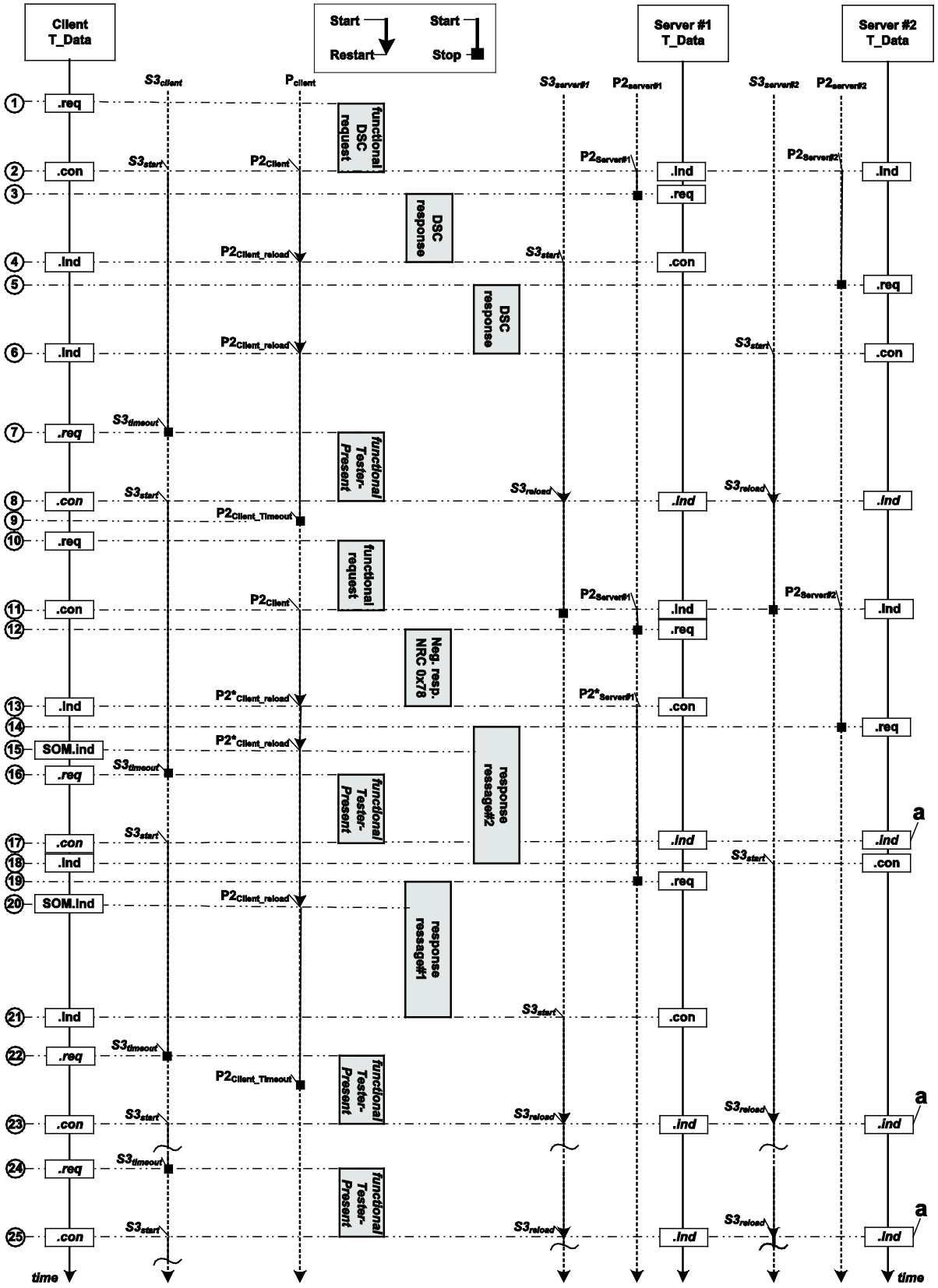
10 **Server #1 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message.

Figure 15 — Functional communication during default session – Enhanced response timing – with SOM.ind

8.2.4 Functional communication during non-default session – with SOM.ind

Figure 16 graphically depicts the timing handling in the client and two servers for a functionally addressed request message during the non-default session (e.g. programmingSession), where one server requests an enhanced response timing via a negative response message including negative response code 0x78.



Key

a Any testerPresent that is received during a disabled S3server timer will be ignored by the server

1 **Client T_Data.req**: diagnostic application issues the functional addressed DiagnosticSessionControl (0x10) request message to transport/network layer. The transport/network layer transmits the request message to the servers.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the reception of the DiagnosticSessionControl (0x10) request message. Now the response timing P_{Client} as described in 8.1.2 and 8.1.3 applies. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the servers are located on the same network. The generated T_Data.con in the client causes the start of the $S3_{Client}$ timer (session timer).

All Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the DiagnosticSessionControl (0x10) request message. All servers start the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{Server\_max}$. Now the response timing $P2_{Server}$ as described in 8.1.2 and 8.1.3 applies.

3 **Server#1 T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) positive response message to transport/network layer within $P2_{Server\_max}$. Server#1 stops the $P2_{Server}$ timer. For the figure given, it is assumed that the client requires a response from the server.

4 **Server#1 T_Data.con**: the completion of the transmission of the DiagnosticSessionControl (0x10) positive response message is indicated in the server#1 via T_Data.con. Now the server#1 starts its $S3_{Server}$ timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the $S3_{Server}$ timer is reset prior to its timeout to keep the server#1 in the non-default session.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. This will cause the client to restart its P_{Client} timer, using the default reload value $P2_{Client}$.

5 **Server#2 T_Data.req**: diagnostic application issues the DiagnosticSessionControl (0x10) positive response message to transport/network layer within $P2_{Server}$. Server#2 stops the $P2_{Server}$ timer. For the figure given, it is assumed that the client requires a response from the server.

6 **Server#2 T_Data.con**: the completion of the transmission of the DiagnosticSessionControl (0x10) positive response message is indicated in the server#2 via T_Data.con. Now the server#2 starts its $S3_{Server}$ timer, which keeps the activated non-default session active as long as it does not time out. It is the client's responsibility to ensure that the $S3_{Server}$ timer is reset prior to its timeout to keep the server#2 in the non-default session.

Client T_Data.ind: transport/network layer issues to diagnostic application the completion of the response message. This will cause the client to restart its P_{Client} timer, using the default reload value $P2_{Client}$.

7 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

8 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

All Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

9 **Client**: this is the indication for the client that no further response messages are expected and can continue with further requests.

10 **Client T_Data.req**: diagnostic application issues functionally addressed request message to transport/network layer.

11 **All server T_Data.ind**: transport/network layer issues to diagnostic application the completion of a request message. All servers start the $P2_{Server}$ timer using the value of $P2_{Server} = P2_{Server\_max}$. Any time the server is in the process of handling any diagnostic service, it stops its $S3_{Server}$ timer.

Client T_Data.con: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$. The value of the P_{Client} timer shall consider any latency that is involved based on the vehicle network design (e.g. communication over gateways, bus bandwidth, etc.). For simplicity, the figure assumes that the client and the servers are located on the same network.

12 **Server #1 T_Data.req**: diagnostic application does not have the positive response message ready and issues negative response message with NRC = 0x78 by a T_Data.req to transport/network layer within $P2_{Server}$. Server#1 stops the $P2_{Server}$ timer. In case any of the addressed servers cannot provide the requested information within the $P2_{Server}$ response timing, it can request an enhanced response timing window by sending a negative response message including negative response code 0x78.

13 **Server#1 T_Data.con**: transport/network layer issues to diagnostic application the completion of the response message. Server#1 starts the $P2_{Server}$ timer using the value of $P2^*_{Server} = P2^*_{Server\_max}$ (default enhanced timing).

Client T_Data.ind: transport/network layer issues to diagnostic application the reception of a response message. Client reloads the P_{Client} timer with the value of $P2^*_{Client} = P2^*_{Client\_max}$ (default enhanced timing). An entry containing the response message address is added to the NRC 0x78 pending list.

14 **Server#2 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server#2 stops the $P2_{Server}$ timer.

15 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. This will cause the client to restart its P_{Client} timer, using the default reload value $P2^*_{Client}$ (default enhanced timing).

16 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

17 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

All Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer. Any TesterPresent that is received during a disabled $S3_{Server}$ timer will be ignored by the server(s).

18 **Client T_Data.ind**: upon transport/network layer issues to diagnostic application the completion of the response message.

Server#2 T_Data.con: when the diagnostic service is completely processed, then the server#2 restarts its $S3_{Server}$ timer. This means that any diagnostic service, including TesterPresent (0x3E), resets the $S3_{Server}$ timer. A diagnostic service is considered to be in progress any time between the start of the reception of the request message (T_DataSOM.ind or T_Data.ind receive) and the completion of the transmission of the final response message, where a response message is required, or the completion of any action that is caused by the request, where no response message is required (point in time reached that would cause the start of the response message).

19 **Server#1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2^*_{Server}$. Server#1 stops the $P2_{Server}$ timer.

20 **Client T_DataSOM.ind**: transport/network layer issues to diagnostic application the reception of a StartOfMessage. The entry matching the response message address is removed from the pending list and the list is now empty. That means no further response messages are pending. This causes the client to restart its P_{Client} timer using the default reload value $P2_{Client}$.

21 **Client T_Data.ind**: transport/network layer issues to diagnostic application the completion of the response message.

Server#1 T_Data.con: transport/network layer issues to diagnostic application the completion of the response message. Now the server#1 starts its $S3_{Server}$ timer.

22 **Client T_Data.req**: Each time the $S3_{Client}$ timer times out causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message.

23 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

All Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

24 **Client T_Data.req**: once the $S3_{Client}$ timer is started in the client, this causes the transmission of a functionally addressed TesterPresent (0x3E) request message, which does not require a response message, each time the $S3_{Client}$ timer times out.

25 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

All Server T_Data.ind: any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

Figure 16 — Functional communication during non-default session – with SOM.ind

The handling of the P_{Client} and $P2_{Server}$ timing is identical to the handling as described in 8.1.2 and 8.1.3, the only exception being that the reload values on the client side and the resulting time the server shall send its final response time might differ. This is based on the transition into a session other than the default session where different P_{Client} timing parameters might apply (see DiagnosticSessionControl (0x10) service in ISO 14229-1 for details on how the timing parameters are reported to the client).

8.3 Minimum time between client request messages

The minimum time between request messages transmitted by the client is required in order to allow for a polling driven service data interpretation in the server. Based on normal functionality, a server might process diagnostic request messages with a design-specific scheduling rate (e.g. 10 ms). The time for the diagnostic service data interpretation scheduler shall be smaller than the performance requirement $P2_{Server}$ in order to meet the server requirements as specified in 8.1.1, 8.1.2 and 8.1.3.

The timing parameter for the minimum time between request messages is divided into the following two timing parameters.

— $P3_{Client\_Func}$: this timing parameter applies to any functionally addressed request message, because it can be the case that a server is not required to respond to a functionally addressed request message if it does not support the requested data.

— $P3_{Client\_Phys}$: this timing parameter applies to any physically addressed request message where there is no response required to be transmitted by the server (suppressPosRspMsgIndicationBit = TRUE).

In the case of physical communication where a response is required by the server, the client can transmit the next request immediately after the complete reception of the last response message, because the server has responded completely to the request — which means that the request is completely handled by the server.

Figure 17 graphically depicts an example of a problem that can occur during functional communication, when the client transmits the next request immediately after it has determined that all expected servers responded to a previous request message.

This scenario does not only apply to functionally addressed requests but also to physically addressed requests where the client does not want to receive any response message (suppressPosRspMsgIndicationBit = TRUE).

In order to handle the described scenarios, the minimum times $P3_{Client\_Phys}$ and $P3_{Client\_Func}$, between the end of a physically or functionally addressed request message and the start of a new physically or functionally addressed request message, are defined for the client.

a) The value of $P3_{Client\_Phys}$ will be identical to $P2_{Server\_max}$ for the physically addressed server. The timing applies to any physically addressed request message in any diagnostic session (default and non-default session) and in case no response is required by the server.

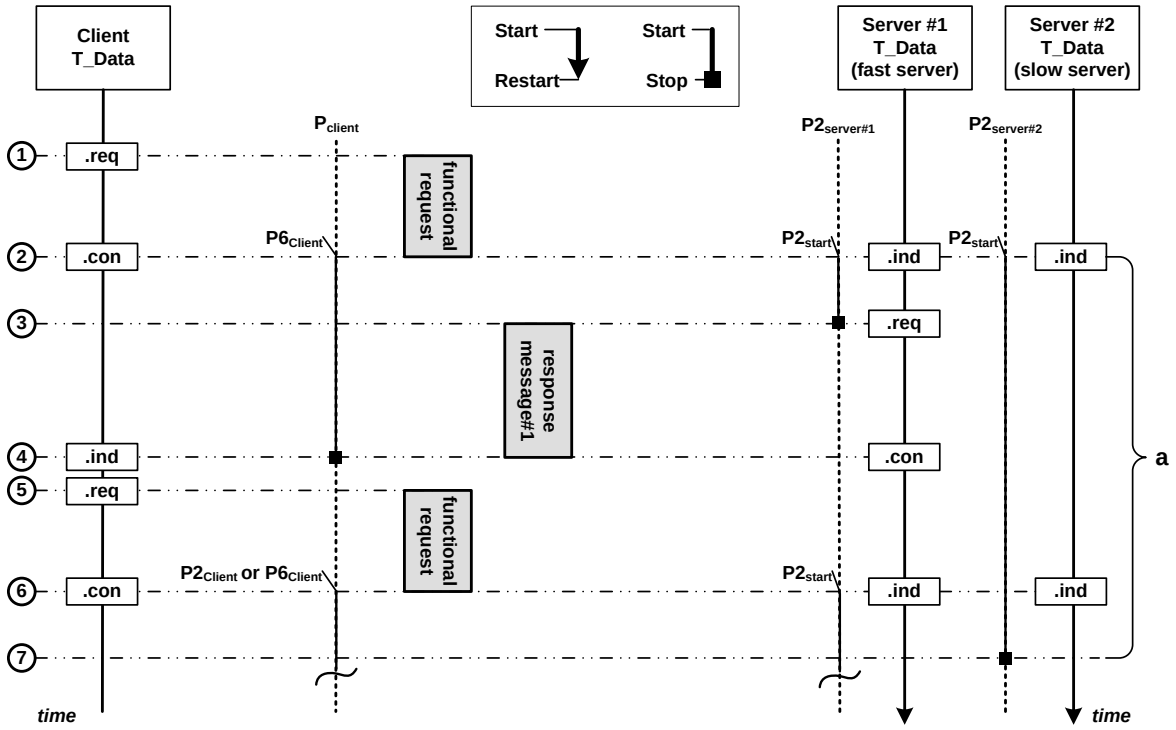
   The $P3_{Client\_Phys}$ timer is started in the client each time a physically addressed request message with no response required is successfully transmitted onto the bus, which is indicated via T_Data.con in the client. When the client wants to transmit a new physically addressed request message following a previous request that was completely handled, then this is only allowed in case the $P3_{Client\_Phys}$ timer is no longer active at the time the client wants to transmit the physically addressed request message. In case $P3_{Client\_Phys}$ would still be active at the point in time the client would like to transmit a new physically addressed request message, then the transmission shall be postponed until $P3_{Client\_Phys}$ is timed out.

b) The value of $P3_{Client\_Func}$ will be the maximum (worst-case) value of all functionally addressed server's $P2_{Server\_max}$ for any functionally addressed request message in any diagnostic session (default and non-default session).

   The $P3_{Client\_Func}$ timer is started in the client each time a functionally addressed request message with response required or with no response required is successfully transmitted onto the bus, which is indicated via T_Data.con in the client. When the client wants to transmit a new functionally addressed request message following a previous request that was completely handled, then this is only allowed in case the $P3_{Client\_Func}$ timer is no longer active at the time the client wants to transmit the functionally addressed request message. In case $P3_{Client\_Func}$ would still be active at the point in time the client would like to transmit a new functionally addressed request message, then the transmission shall be postponed until $P3_{Client\_Func}$ is timed out.

NOTE “Completely handled” means that either no response is received in case no response is required, all expected responses to a functionally addressed request are received in case the responding servers are known and responses are required or a P_{Client} timeout occurred in case the responding servers are not known and responses are required.

The requirement for the server is that it shall start with its response message within $P2_{Server}$. This means that the diagnostic data interpretation rate of the server shall be less than $P2_{Server\_max}$.



Key

a diagnostic service data interpretation rate of server#2

1 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$.

All Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. All servers start the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

3 **Server#1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server#1 stops the $P2_{Server}$ timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there will be no response from server#2. Server#1 is a fast server and can immediately process the received request message and transmits its response within $P2_{Server}$.

4 **Client T_Data.ind**: transport/network layer issues to diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its timer P_{Client}.

Server#1 T_Data.con: transport/network layer issues to diagnostic application the completion of the response message.

5 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer. The client would send the next request right after the completion of all expected response messages.

6 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$ (if T_DataSOM.ind is supported) or $P6_{Client} = P6_{Client\_max}$).

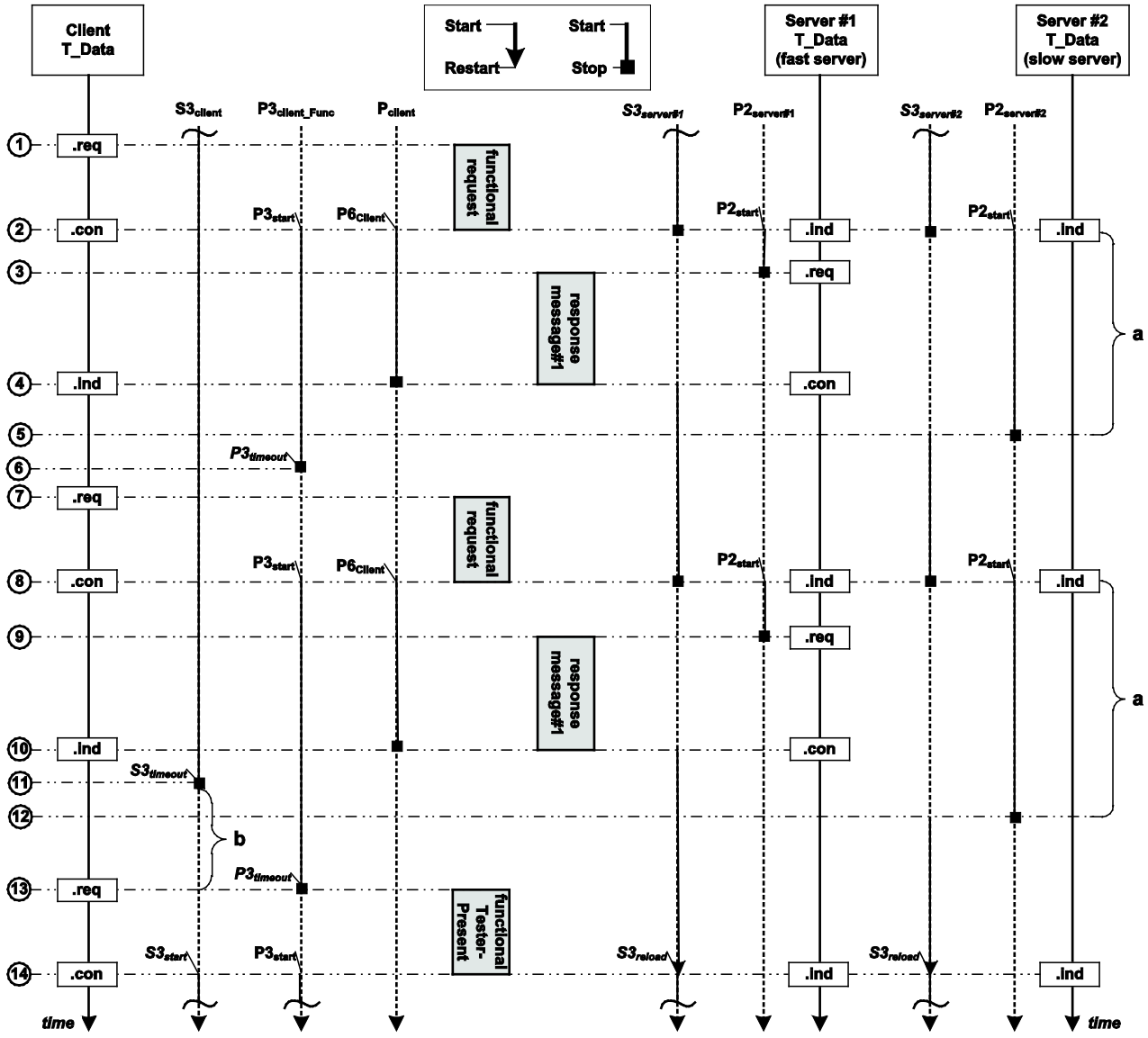
Server#1 T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. Server#1 starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$.

Server#2 T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. The request message will only be processed by the fast server#1, because server#2 did not yet handle the previous request. Any request message that is received during the diagnostic service data interpretation rate of server#2 will be ignored by the server#2.

7 **Server#2**: Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport/network layer reception of the functionally addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is always suppressed in case of a functionally addressed request message). As shown in the figure, this would be after the completion of the response message of server#1, and even after the completion of the next request message transmitted by the client.

Figure 17 — Example of critical issue when transmitting next request too early

Figure 18 graphically depicts the $P3_{Client\_Func}$ timing handling for the client (based on the communication scenario illustrated in Figure 16). In addition, Figure 18 shows the handling of a functionally addressed TesterPresent (0x3E) request message in the client in the case in which the $P3_{Client\_Func}$ timer is still active when $S3_{Client}$ times out (request will be postponed until $P3_{Client\_Func}$ times out).



Key

a diagnostic service data interpretation rate of server#2

b functional TesterPresent delay

1 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$ (if T_DataSOM.ind is supported) or $P6_{Client} = P6_{Client\_max}$) and, furthermore, its $P3_{Client\_Func}$ timer.

All Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. All servers start the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$. All servers stop the $S3_{Server}$ timer.

3 **Server#1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server}$. Server#1 stops the $P2_{Server}$ timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there will be no response from server#2. Server#1 is a fast server and can immediately

process the received request message and transmits its response within $P2_{Server}$.

4 **Client T_Data.ind**: transport/network layer issues to diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its P_{Client} timer.

Server#1 T_Data.con: transport/network layer issues to diagnostic application the completion of the response message. Now the server#1 starts its $S3_{Server}$ timer.

5 **Server#2**: Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport/network layer reception of the functionally addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is always suppressed in case of a functionally addressed request message). Now the server#2 stops the $P2_{Server}$ timer and starts its $S3_{Server}$ timer.

6 **Client**: Even if the client has received all expected response messages to a functionally addressed request message, it shall wait until $P3_{Client\_Func}$ times out before it is allowed to transmit the next request message. At the point in time $P3_{Client\_Func}$ times out.

7 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer. The client would send the next request right after the completion of all expected response messages.

8 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$ (if T_DataSOM.ind is supported) or $P6_{Client} = P6_{Client\_max}$) and, furthermore, its timer $P3_{Client\_Func}$.

All Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. All servers start the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$. All servers stop the $S3_{Server}$ timer.

9 **Server#1 T_Data.req**: diagnostic application has prepared the response message and issues a T_Data.req to transport/network layer within $P2_{Server\_max}$. Server#1 stops the $P2_{Server}$ timer. For the request message, it is assumed that only server #1 supports the requested information, which means that there will be no response from server#2. Server#1 is a fast server and can immediately process the received request message and transmits its response within $P2_{Server}$.

10 **Client T_Data.ind**: transport/network layer issues to diagnostic application the completion of the response message. The client only expected a response message from server #1, therefore it stops its P_{Client} timer.

Server#1 T_Data.con: transport/network layer issues to diagnostic application the completion of the response message. Now the server#1 starts its $S3_{Server}$ timer.

11 **Client**: The $S3_{Client}$ timer of the client times out, which forces the client to transmit a functionally addressed TesterPresent (0x3E) request message, not requiring a response message from the addressed server(s). Based on the situation in which the $P3_{Client\_Func}$ timer is still active at this point in time, the transmission of the TesterPresent (0x3E) shall be postponed until the expiration of the $P3_{Client\_Func}$ timer.

12 **Server#2**: Server #2 is a slow server and interprets received requests on a periodic basis (diagnostic service data interpretation rate). In the worst-case, the last check for incoming request a message is prior to the transport/network layer reception of the functionally addressed request message. This would mean that the request would be stored in a buffer and processed at the earliest the next time the scheduler checks for an incoming request. When server#2 processes the request, then it determines that it does not need to answer, because it does not support the requested information (e.g. server sends no response message because the negative response code requestOutOfRange is always suppressed in case of a functionally addressed request message). Now the server#2 stops the $P2_{Server}$ timer and starts its $S3_{Server}$ timer.

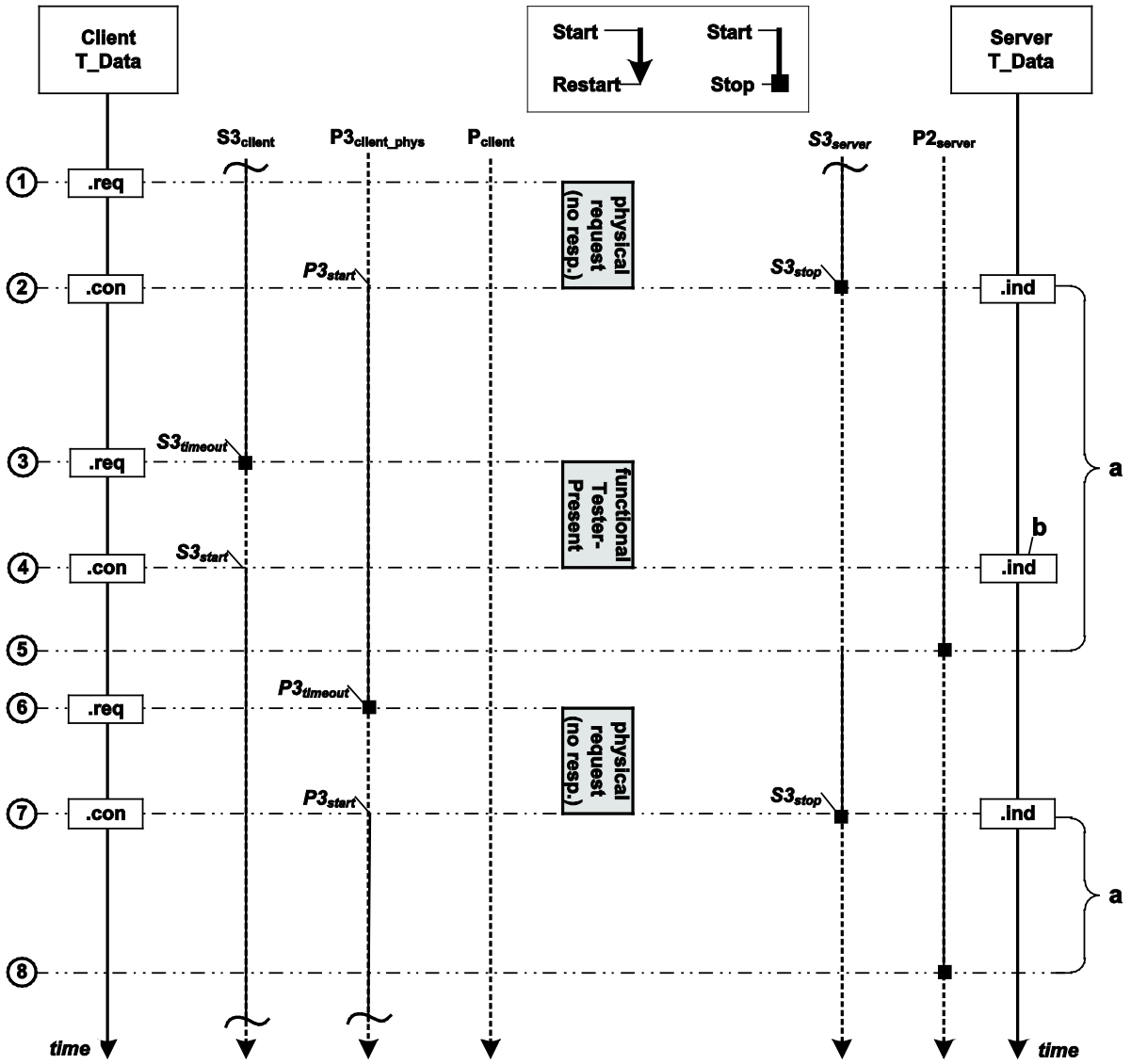
13 **Client T_Data.req**: When the $P3_{Client\_Func}$ timer times out, the functionally addressed TesterPresent (0x3E) request can be transmitted by the client via T_Data.req.

14 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out. Client starts its P_{Client} timer using the default reload value $P2_{Client} = P2_{Client\_max}$ (if T_DataSOM.ind is supported) or $P6_{Client} = P6_{Client\_max}$) and, furthermore, its timer $P3_{Client\_Func}$..

All Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. Any TesterPresent (0x3E) request message that is received during an activated $S3_{Server}$ timer will reload the $S3_{Server}$ timer.

Figure 18 — Minimum time between functionally addressed request messages ($P3_{Client\_Func}$)

Figure 19 graphically depicts the $P3_{Client\_Phys}$ timing handling for the client. The figure shows the handling of a physically addressed request that does not require a response and of the functionally addressed TesterPresent (0x3E) request message in the client when $S3_{Client}$ times out.



Key

a diagnostic service data interpretation rate

b any testerPresent that is received during a disabled S3server timer can be ignored by the server

1 **Client T_Data.req**: diagnostic application issues a request message to transport/network layer that does not require a response.

2 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. Client starts its $P3_{Client\_Phys}$ timer using the default reload value $P3_{Client\_Phys} = P3_{Client\_Phys\_max}$. There is no response required to be transmitted, therefore the client does not need to start its P_{Client} timer.

Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$ and in any non-default session the $S3_{Server}$ timer is now stopped.

3 **Client T_Data.req**: The $S3_{Client}$ timer of the client times out, which forces the client to transmit a functionally addressed TesterPresent (0x3E) request message, not requiring a response message from the addressed server(s). It is assumed that the $P3_{Client\_Func}$ timer is no active at this point in time, which means that the request is transmitted immediately.

4 **Client T_Data.con**: upon the indication of the completed transmission of the TesterPresent (0x3E) request message via T_Data.con of its transport/network layer, the client once again starts its $S3_{Client}$ timer. This means that the functionally addressed TesterPresent (0x3E) request message is sent on a periodic basis every time $S3_{Client}$ times out.

Server T_Data.ind: any TesterPresent (0x3E) request message that is received during processing another request message can be ignored by the server, because it has already stopped its S3Server timer and will restart it once the service that is in progress is processed completely.

5 **Server**: Server interprets received requests on a periodic basis (diagnostic service data interpretation rate). The request is processed the next time the scheduler checks for incoming requests. The completed execution of the service would restart the $S3_{Server}$ timer during any non-default session and stops the $P2_{Server}$ timer.

6 **Client T_Data.req**: when the $P3_{Client\_Phys}$ timer times out in the client, the client can transmit the next physically addressed request message by issuing T_Data.req to its transport/network layer.

7 **Client T_Data.con**: transport/network layer issues to diagnostic application the confirmation of the completion of the request message. The client now starts its $P3_{Client\_Phys}$ timer again. There is no response required to be transmitted, therefore the client does not need to start its P_{Client} timer.

Server T_Data.ind: transport/network layer issues to diagnostic application the completion of the request message. Server starts the $P2_{Server}$ timer using the default value of $P2_{Server} = P2_{Server\_max}$ and in any non-default session the $S3_{Server}$ timer is now stopped.

8 **Server**: Server interprets received requests on a periodic basis (diagnostic service data interpretation rate). The request is processed the next time the scheduler checks for incoming requests. The completed execution of the service would restart the $S3_{Server}$ timer during any non-default session and stops the $P2_{Server}$ timer.

Figure 19 — Minimum time between physically addressed request messages ($P3_{Client\_Phys}$)

Annex A
(normative)

T_PDU interface

Figure A.1 shows the T_PDU (virtual PDU) as an interface between the unified diagnostic services PDU and any communication protocol.

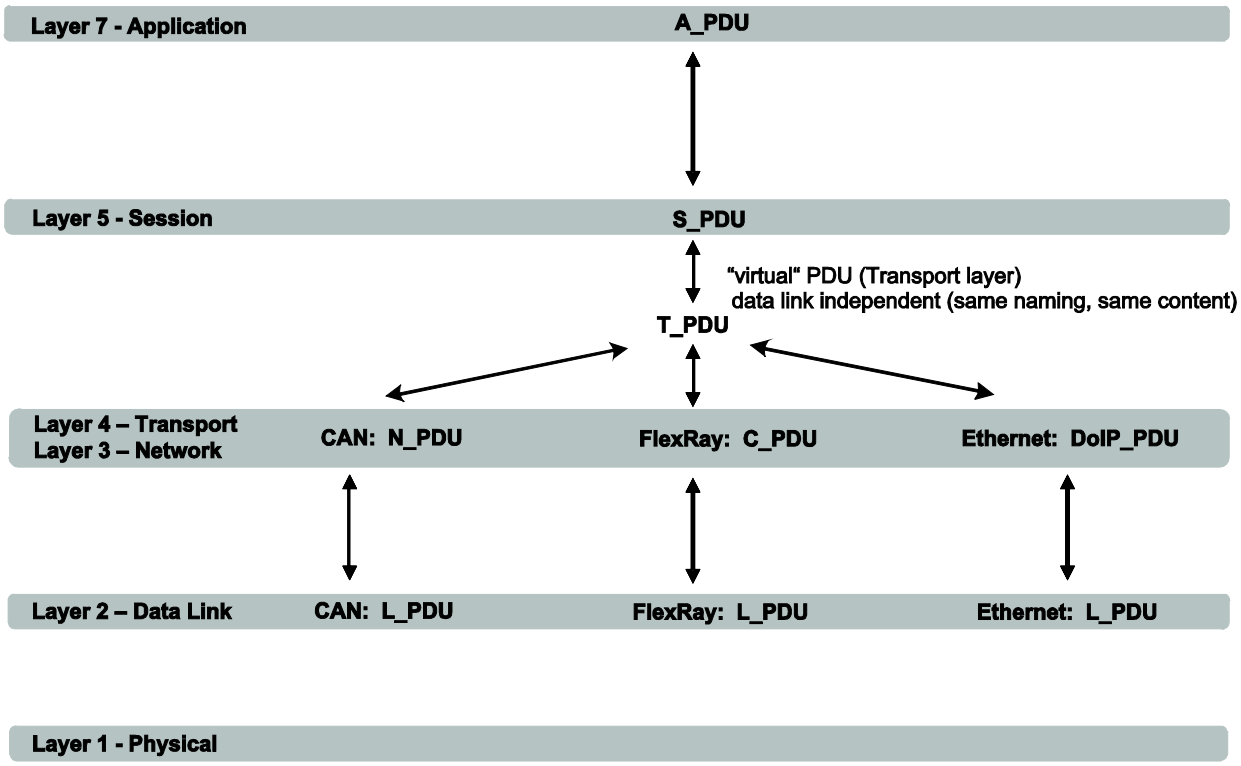


Figure A.1 — T_PDU virtual PDU interface to any communication protocol

Annex B
(informative)

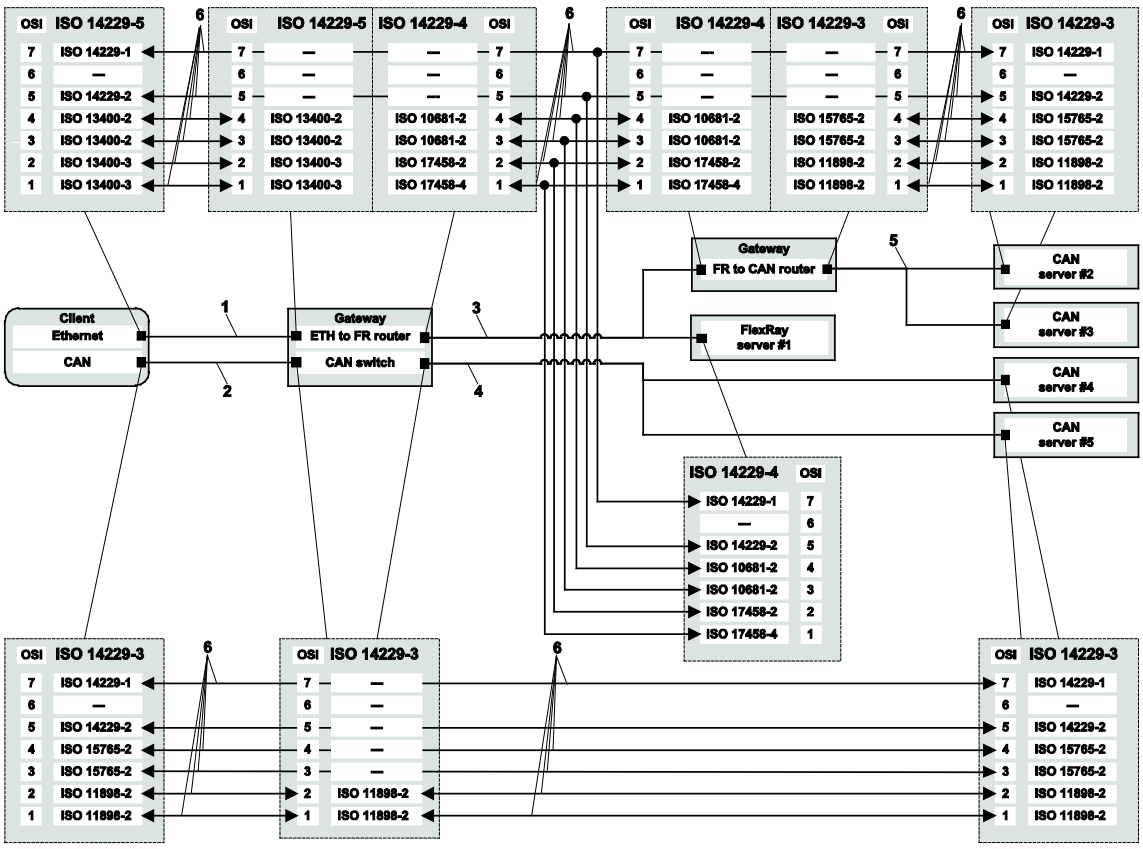
Vehicle diagnostic OSI layer architecture examples

B.1 Vehicle diagnostic OSI layer gateway example

Figure B.1 depicts an example of a vehicle diagnostic network architecture with different network technologies and two gateway instances. The "Ethernet to FlexRay router" implemented in the gateway device is a networking device that transfers the PDU on OSI layers 3 and 4. The "CAN switch" implemented in the gateway device is a networking device that transfers the PDU on OSI layer 2.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Data links key 1 and key 2 are both available for vehicle diagnostic data access. Data links key 3 through key 5 are examples for in-vehicle communication networks, while key 5 is located behind a second gateway (FlexRay to CAN router).



Key

1	Diagnostic Ethernet (ETH)	**5**	CAN
2	Diagnostic CAN	**6**	logical connection
3	FlexRay Communications System (FR)	**---**	not applicable
4	CAN		

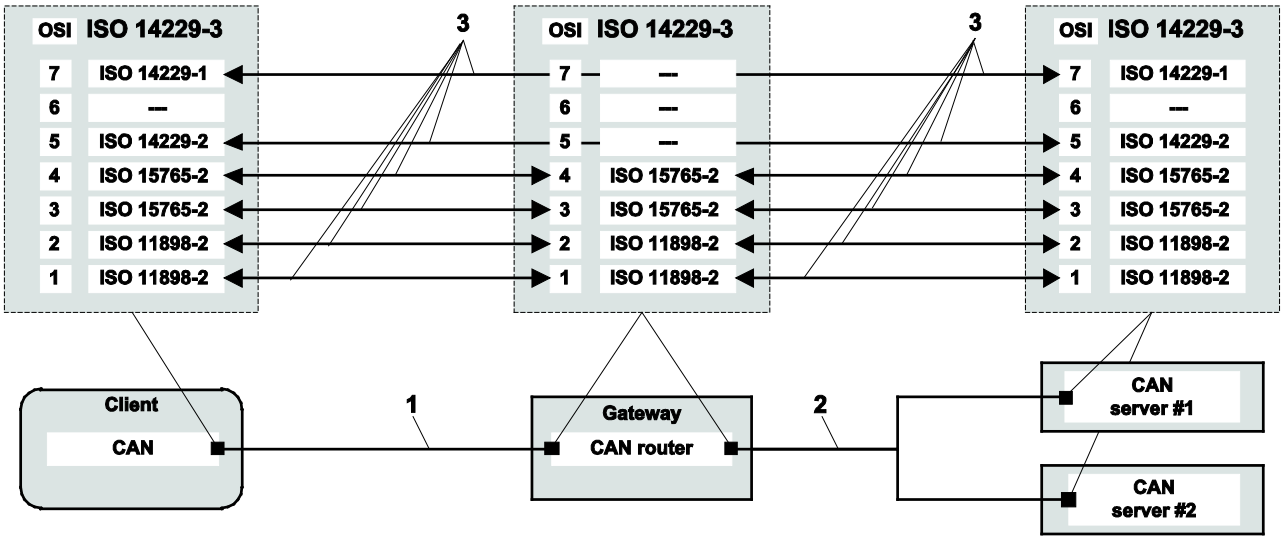
Figure B.1 — Vehicle diagnostic OSI layer gateway example

B.2 Vehicle diagnostic OSI layer CAN router example

Figure B.2 depicts an example of a vehicle diagnostic network architecture with a "CAN router" implemented in a gateway. The "CAN router" implemented in the gateway device is a networking device that transfers the PDU on OSI layers 3 and 4.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Data link key 1 is available for vehicle diagnostic data access. Data link key 2 is an example for an in-vehicle CAN communication network.



Key

1 Diagnostic CAN
2 CAN
3 logical connection
--- not applicable

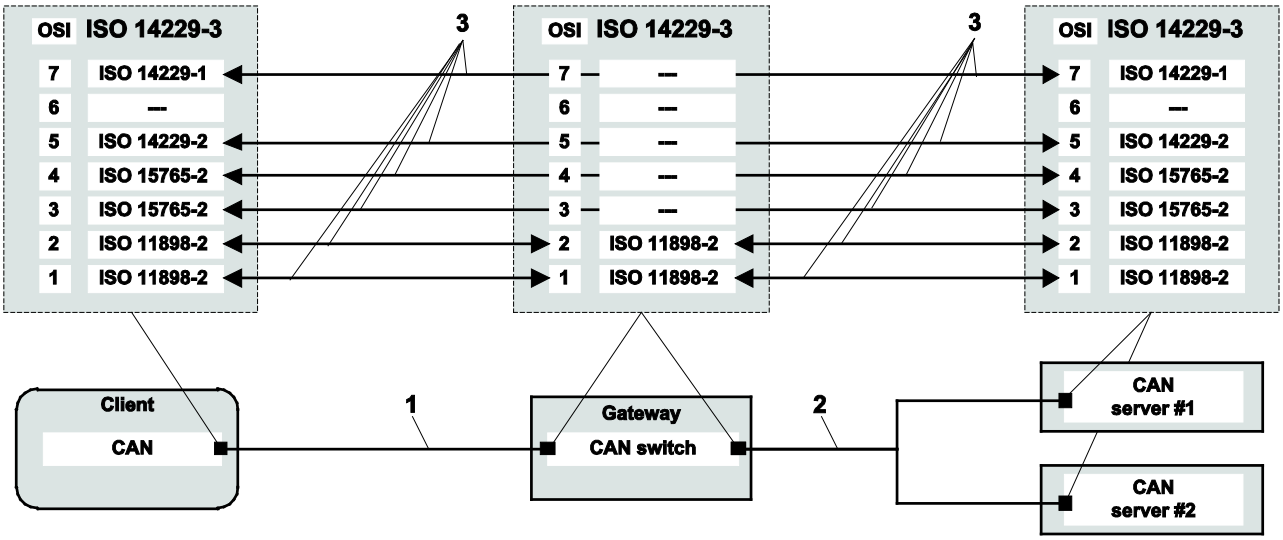
Figure B.2 — Vehicle diagnostic OSI layer CAN router example

B.3 Vehicle diagnostic OSI layer CAN switch example

Figure B.3 depicts an example of a vehicle diagnostic network architecture with a "CAN switch" implemented in a gateway. The "CAN switch" implemented in the gateway device is a networking device that transfers the PDU on OSI layer 2.

The framed boxes (dashed line) provide the ISO/OSI layer model about the different protocols which shall be supported by the corresponding interface.

Data link key 1 is available for vehicle diagnostic data access. Data link key 2 is an example for an in-vehicle CAN communication network.



Key

1 Diagnostic CAN
2 CAN
3 logical connection
--- not applicable

Figure B.3 — Vehicle diagnostic OSI layer CAN switch example

Bibliography

[1] ISO/IEC 7498-1, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*

[2] ISO 10681-2, *Road vehicles — Communication on FlexRay — Part 2: Communication layer services*

[3] ISO/IEC 10731, *Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services*

[4] ISO 11898-1, *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*

[5] ISO 11898-2, *Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit*

[6] ISO 13400-2, *Road vehicles — Diagnostic communication over Internet Protocol (DoIP) — Part 2: Transport protocol and network layer services*

[7] ISO 13400-3, *Road vehicles — Diagnostic communication over Internet Protocol (DoIP) — Part 3: Wired vehicle interface based on IEEE 802.3*

[8] ISO 14229-3, *Road vehicles — Unified diagnostic services (UDS) — Part 3: Unified diagnostic services on CAN implementation (UDSonCAN)*

[9] ISO 14229-4, *Road vehicles — Unified diagnostic services (UDS) — Part 4: Unified diagnostic services on FlexRay implementation (UDSonFR)*

[10] ISO 14229-5, *Road vehicles — Unified diagnostic services (UDS) — Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP)*[1)]

[11] ISO 14229-6, *Road vehicles — Unified diagnostic services (UDS) — Part 6: UDS on K-Line implementation (UDSonK-Line)*[1)]

[12] ISO 14230-1, *Road vehicles — Diagnostic communication over K-Line (DOK-Line) — Part 1: Physical layer*

[13] ISO 14230-2, *Road vehicles — Diagnostic communication over K-Line (DOK-Line) — Part 2: Data link layer*

[14] ISO 15765-2, *Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services*

[15] ISO 17458-2, *Road vehicles — FlexRay communications system — Part 2: Data link layer specification*[1)]

[16] ISO 17458-4, *Road vehicles — FlexRay communications system — Part 4: Electrical physical layer specification*[1)]

[17] ISO 27145-2, *Road vehicles — Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements — Part 2: Common data dictionary (CDD)*

[1)] To be published.

[18] ISO 27145-4, *Road vehicles — Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements — Part 4: Connection between vehicle and test equipment*

[19] IEEE 802.3, *IEEE Standard for Information technology — Telecommunications and information exchange between systems – Local and metropolitan area networks — Specific requirements — Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*[2)]

[2)] Equivalent to ISO/IEC 8802-3.

ICS 43.180

Price based on 48 pages