INTERNATIONAL STANDARD

ISO 17215-2

First edition
2014-04-15

Road vehicles — Video communication interface for cameras (VCIC) —

Part 2: Service discovery and control

Véhicules routiers — Interface de communication vidéo pour caméras (ICVC)

Reference number
ISO 17215-2:2014(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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electric and electronic equipment*.

ISO 17215 consists of the following parts, under the general title *Road vehicles — Video communication interface for cameras (VCIC)*:

— *Part 1: General information and use case definition*

— *Part 2: Service discovery and control*

— *Part 3: Camera message dictionary*

— *Part 4: Implementation of communication requirements*

Introduction

Driver assistance systems are more and more common in road vehicles. From the beginning, cameras were part of this trend. Analogue cameras were used in the beginning, because of lower complexity of the first systems. With increasing demand for more advanced functionality, digital image processing has been introduced. So-called one box design cameras (combining a digital image sensor and a processing unit) appeared in the vehicles.

Currently, the market demands such systems with multiple functions. Even different viewing directions are in use. It seems to be common sense that 6 up to 12 cameras in a single vehicle will be seen in the next future. Out of this and the limitation in size, power consumption, etc. it will lead to designs where the cameras are separated from the processing unit. Therefore, a high performance digital interface between camera and processing unit is necessary.

This International Standard has been established in order to define the use cases, the communication protocol, and the physical layer requirements of a video communication interface for cameras which covers the needs of driver assistance applications.

The video communication interface for cameras

— incorporates the needs of the whole life cycle of an automotive grade digital camera,

— utilizes existing standards to define a long-term stable state-of-art video communication interface for cameras usable for operating and diagnosis purpose,

— can be easily adapted to new physical data link layers including wired and wireless connections by using existing adaption layers, and

— is compatible with AUTOSAR.

This part of ISO 17215 is related to the general information and use case definition. This is a general overview document which is not related to the OSI model.

To achieve this, it is based on the open systems interconnection (OSI) basic reference model specified in ISO/IEC 7498-1 and ISO/IEC 10731 which structures communication systems into seven layers. When mapped on this model, the protocol and physical layer requirements specified by this International Standard, in accordance with Table 1, are broken into following layers:

— application (layer 7), specified in ISO 17215-3;

— presentation layer (layer 6), specified in ISO 17215-2;

— session layer (layer 5), specified in ISO 17215-2;

— transport protocol (layer 4), specified in ISO 17215-4, ISO 13400-2;

— network layer (layer 3), specified in ISO 17215-4, ISO 13400-2;

— data link layer (layer 2), specified in ISO 17215-4, ISO 13400-3;

— physical layer (layer 1), specified in ISO 17215-4, ISO 13400-3.

Table 1 — Specifications applicable to the OSI layers

Applicability	OSI 7 layers	Video communication interface for cameras		Camera diagnostics
Seven layers according to ISO 7498-1 and ISO/IEC 10731	Application (layer 7)	ISO 17215-3		
	Presentation (layer 6)	ISO 17215-2		
	Session (layer 5)	ISO 17215-2		
	Transport (layer 4)	ISO 17215-4	Other future interface standards	ISO 13400-2
	Network (layer 3)			
	Data link (layer 2)	ISO 17215-4		ISO 13400-3
	Physical (layer 1)			

ISO 17215-1 has been established in order to define the use cases for vehicle communication systems implemented on a video communication interface for cameras; it is an overall document not related to the OSI model.

ISO 17215-3 covers the application layer implementation of the video communication interface for cameras; it includes the API.

ISO 17215-2 covers the session and presentation layer implementation of the video communication interface for cameras.

ISO 17215-4, being the common standard for the OSI layers 1 to 4 for video communication interface for cameras, complements ISO 13400-2 and ISO 13400-3 and adds the requirement for video transmission over Ethernet.

ISO 17215-2 and ISO 17215-3 (OSI layer 5 to 7) services have been defined to be independent of the ISO 17215-4 (OSI layer 1 to 4) implementation. Therefore ISO 17215-4 could be replaced by other future communication standards.

Road vehicles — Video communication interface for cameras (VCIC) —

Part 2: Service discovery and control

1 Scope

This part of ISO 17215 specifies how services can be discovered and controlled. This functionality is located mainly in layer 5 of the OSI model. Both discovery and control are implemented using the scalable service oriented middlewire over IP (SOME/IP). Figure 1 shows a diagram of these aspects and their relation to other parts of this International Standard.

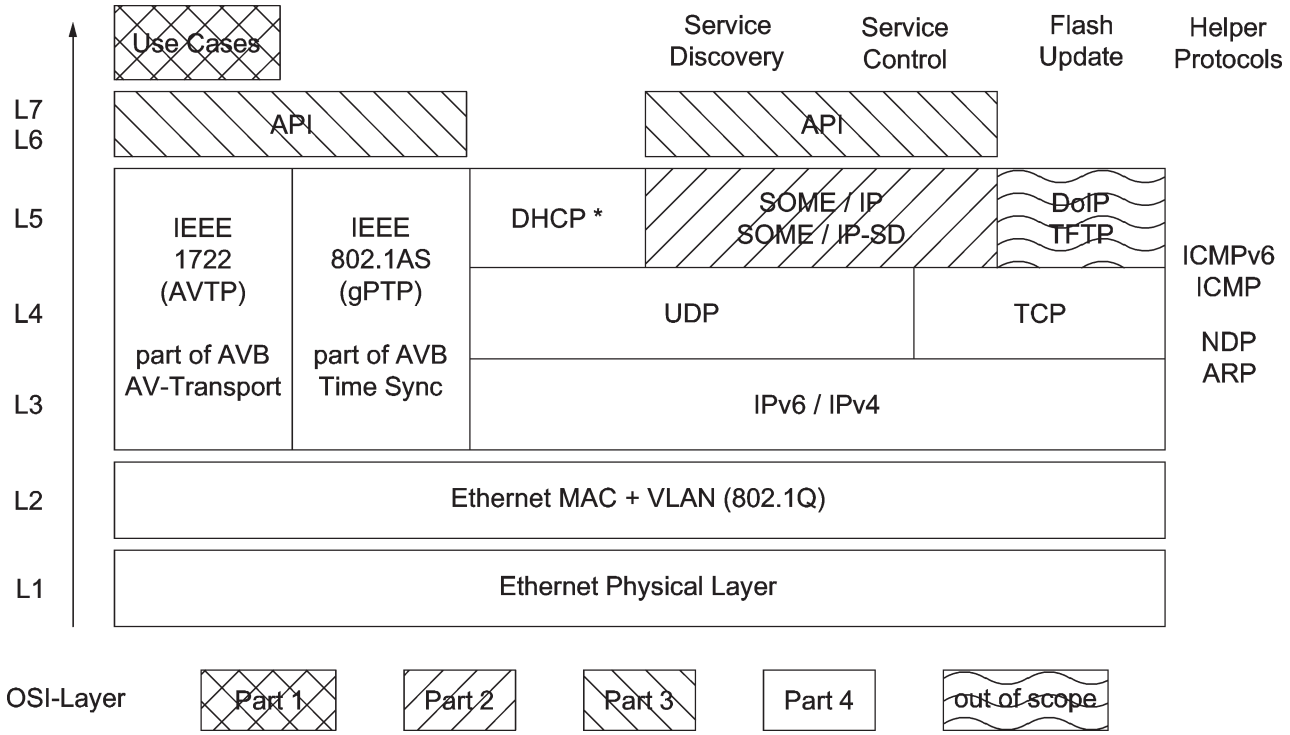

Figure 1 — Overview of ISO 17215

The general terminology defined in ISO 17215-1 is also used in this part of ISO 17215.

2 Normative references

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

ISO 7498-1, *Information processing systems — Open Systems Interconnection — Basic Reference Model: The Basic Model — Part 1*

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

ISO 17215 (all parts), *Road vehicles — Video communication interface for cameras (VCIC)*

NOTE The keywords shall, should, etc. as defined in [IETF RFC 2119] are used in this part of ISO 17215 to indicate requirement levels. Capitalization of those keywords is not required.

If an RFC referenced by this part of ISO 17215 has been updated by one or several RFCs, the update is fully applicable for the purpose of implementing this International Standard. This presumes the additional document describes an implementation which is compatible with implementation described by document referred to herein.

If one or more errata for an RFC referenced by this part of ISO 17215 have been published, all of these errata documents are fully applicable for the purpose of implementing this International Standard.

It is assumed that future implementations of this International Standard will use the most recent versions of the referenced RFCs, but maintain backward compatibility to existing implementations.

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 17215-1 and the following apply.

3.1.1
AUTOSAR
open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers, and tool developers

3.1.2
client
software component that uses a service instance, e.g. by invoking a method

3.1.3
event
fire and forget message invoked on changes or cyclic and sent from server to client

3.1.4
event group
logical grouping of events or notifications within a service in order to ease subscription

3.1.5
field
representation of a remote property which has up to one getter, up to one setter, and up to one notifier

Note 1 to entry: A getter/setter is a method to get/set the value of a field.

3.1.6
fire and forget communication
RPC call that consists only of a request message

3.1.7
interface definition
concrete specification of a service interface (e.g. in IDL or PDU notation)

Note 1 to entry: In the case of ISO 17215, the interface definition is contained in ISO 17215-3.

3.1.8
method
procedure, function, or subroutine that can be called by a client

3.1.9
notification
fire and forget message that is sent on defined status changes or periodically by the notifier of an event or a field

Note 1 to entry: Field messages cannot be distinguished from an event message; therefore, when referring to an event message, this shall also be true for the messages of notifiers of fields.

3.1.10
parameters
input, output, or input/output arguments of a method

3.1.11
remote procedure call
method call between two processes that is transmitted using messages

3.1.12
request
message from the client to the server invoking a method

3.1.13
request/response communication
RPC that consists of a request and a response

3.1.14
response
message from the server to the client transporting results of a method invocation

3.1.15
server
software component that offers a service instance, e.g. by providing a method

3.1.16
service
logical combination of zero or more methods, zero or more fields, and zero or more events

3.1.17
service instance
instantiation of the service interface, which can exist more than once in the vehicle or on an ECU

3.1.18
service interface
abstract specification of a service including its methods, events, and fields

3.1.19
union
data structure that can dynamically assume different data types (also known as variant)

3.2 Abbreviated terms

Term	Description
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
RPC	Remote Procedure Call
ECU	Electronic Control Unit
IDL	Interface Description Language
AUTOSAR	AUTomotive Open System ARchitecture

4 Conventions

ISO 17215 is based on the conventions specified in the OSI service conventions (ISO/IEC 10731) as they apply for physical layer, protocol, network and transport protocol, and diagnostic services.

5 Overview

5.1 General

ISO 17215 has been established in order to implement a standardized video communication interface for cameras in vehicles.

The focus of ISO 17215 is using existing protocols.

Figure 1 specifies the relation to the other parts of the standard.

Figure 2 specifies the relation of ISO 17215 to existing protocols.

5.2 Document overview and structure

This International Standard consists of a set of four sub-documents, which provide all references and requirements to support the implementation of a video communication interface for cameras according to the standard at hand.

— ISO 17215-1:. This part provides an overview of the document set and structure along with use case definitions and a common set of resources (definitions, references) for use by all subsequent parts.

— ISO 17215-2:. This part specifies the discovery and control of services provided by a VCIC camera.

— ISO 17215-3:. This part specifies the standardized camera messages and data types used by an VCIC camera (OSI layer 7).

— ISO 17215-4:. This part specifies standardized low-level communication requirements for implementation of the physical layer, data link layer, network layer, and transport layer (OSI layers 1 to 4).

5.3 Open Systems Interconnection (OSI) model

This International Standard is based on the Open Systems Interconnection (OSI) basic reference model as specified in ISO/IEC 7498 which structures communication systems into seven layers.

All parts of this International Standard are guided by the OSI service conventions as specified in ISO/IEC 10731 to the extent that they are applicable to diagnostic services. These conventions define the interaction between the service user and the service provider through service primitives.

The aim of this subclause is to give an overview of the OSI model and show how it has been used as a guideline for this part of ISO 17215. It also shows how the OSI service conventions have been applied to this International Standard.

The OSI model structures data communication into seven layers called (from top to bottom) the application layer (layer 7), presentation layer, session layer, transport layer, network layer, data link layer, and physical layer (layer 1). A subset of these layers is used in ISO 17215.

The purpose of each layer is to provide services to the layer above. The active parts of each layer, implemented in software, hardware or any combination of software and hardware, are called *entities*. In the OSI model, communication takes place between entities of the same layer in different nodes. Such communicating entities of the same layer are called peer entities.

The services provided by one layer are available at the Service Access Point (SAP) of that layer. The layer above can use them by exchanging data parameters.

This International Standard distinguishes between the services provided by a layer to the layer above it and the protocol used by the layer to send a message between the peer entities of that layer. The reason for this distinction is to make the services, especially the application layer services and the transport layer services, reusable also for other types of networks than the video communication interface for cameras. In this way, the protocol is hidden from the service user and it is possible to change the protocol if demanded by special system requirements.

5.4 Document reference according to OSI model

Figure 2 illustrates the document references.

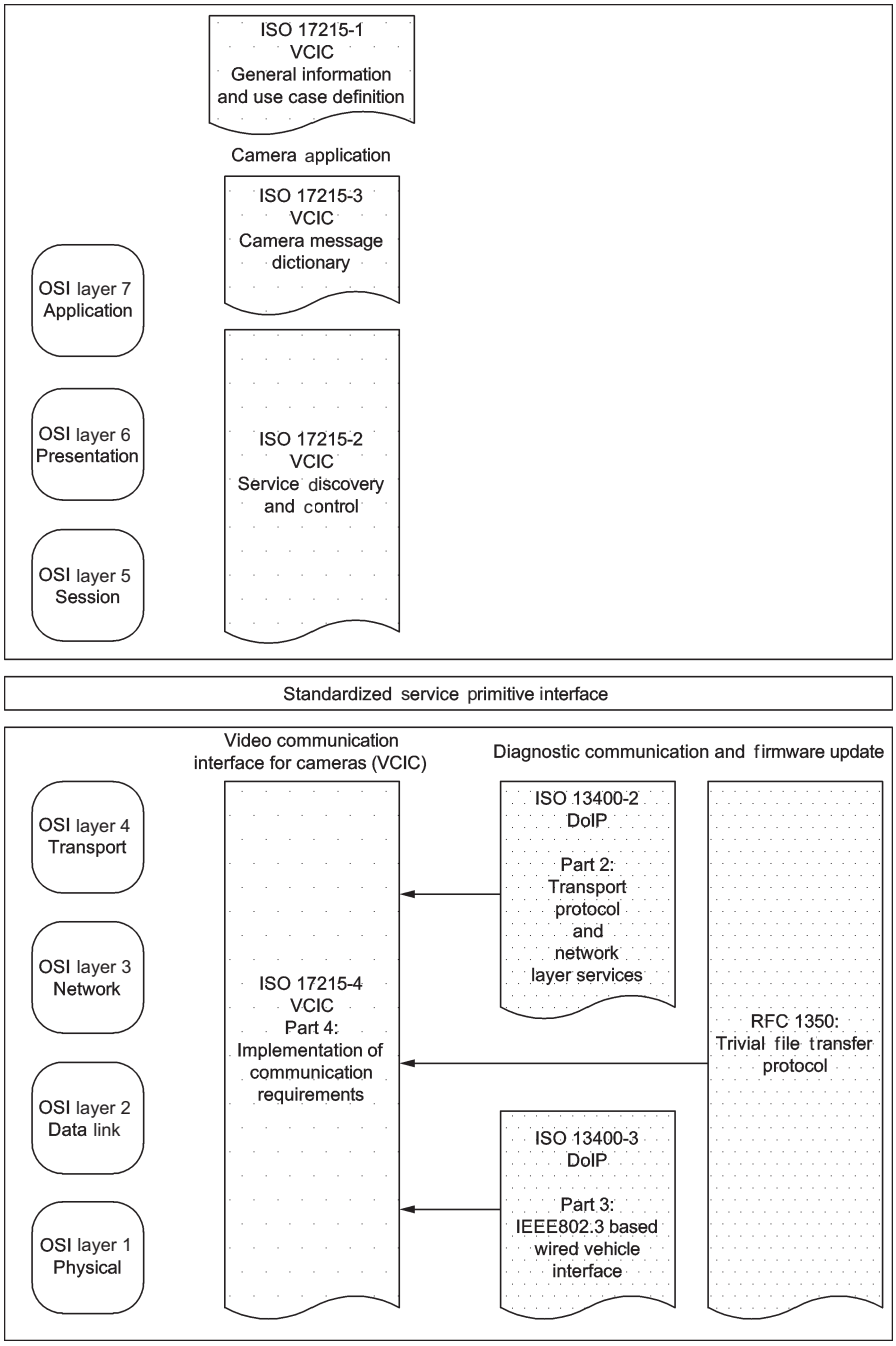

Figure 2 — Video communication interface for camera's document reference according to OSI model

6 SOME/IP

6.1 General

Service discovery as well as command and control is performed using the Scalable service-Oriented MiddlewarE over IP. SOME/IP is a lightweight RPC protocol that defines an AUTOSAR-compatible method for describing interfaces and marshalling data over automotive Ethernet and IP networks. The basic feature set of the SOME/IP wire format is already supported by AUTOSAR. This allows AUTOSAR to parse the RPC PDUs and transport the signals to the application.

6.2 Header

This subclause defines the header structure.

For interoperability reasons, the header layout shall be identical for all implementations of SOME/IP and is shown in Figure 3.

— The header-fields are presented in transmission order; i.e. the header-fields on the top left are transmitted first. In the following sections, the different header-fields and their usage is being described.

— All RPC header fields shall use network byte order (big endian) [IETF RFC 791].

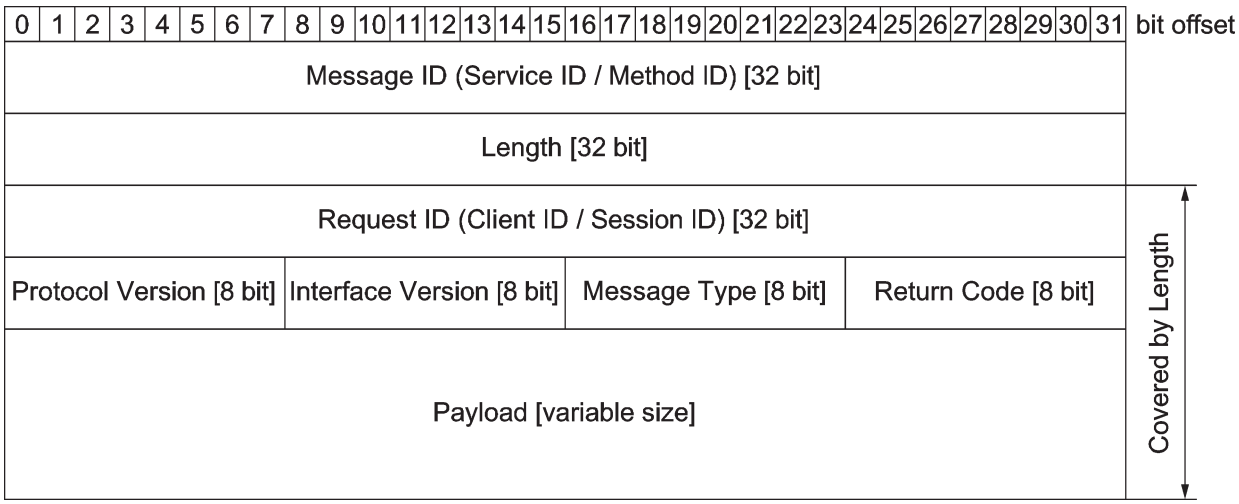

Figure 3 — General SOME/IP header layout

6.2.1 Message ID [32-bit]

The Message ID is a 32-bit identifier that is used to dispatch the RPC call to the method of an application and to identify an event.

The assignment of the Message ID is up to the user. The next section describes how to structure the Message IDs in order to ease the organization of Message IDs.

— The Message ID shall uniquely identify a method or an event and the format of its associated PDUs.

In order to structure the different methods and events, they are clustered into services. Services have a set of methods and events as well as a Service ID, which is used for exactly one service. Events are in addition clustered into event groups, which simplify the subscription for multiple events.

— The message ID for RPC calls shall consist of a 16-bit Service ID (high bytes) and a 16-bit Method ID (low bytes).

— The Service ID and the Method ID shall be defined in the interface specification.

— The highest bit of the Event ID shall always be one.

This scheme allows for up to 65 536 services with up to 32 767 methods and 32767 events each.

— The message ID for events shall consist of a 16-bit Service ID (high bytes) and a 16-bit Notification ID (low bytes).

— The Eventgroup ID and Event ID shall be specified in the interface specification.

— The highest bit of the Eventgroup ID shall always be one.

6.2.2 Length [32-bit]

Length is a field of 32 bits containing the length (in bytes) of the payload, beginning with the Request ID/Client ID and ending with the end of the SOME/IP message. The length does not include the Message.

6.2.3 Request ID [32-bit]

The Request ID allows a client to differentiate multiple calls to the same method.

— The Request ID shall be unique for a single client and server combination.

— When generating a response message, the server shall copy the Request ID from the request to the response message. This allows the client to map a response to the issued request even with more than one request outstanding.

The Request ID needs to be restructured.

— The Request ID shall consist of a 16-bit Client ID (high bytes) and a 16-bit Session ID (low bytes).

The Client ID is the unique identifier for the calling client inside the ECU. The Session ID is a unique identifier chosen by the client for each call.

— The Client ID within each ECU shall be configured by the manufacturer.

— If no response to a message is expected, e.g. for events or notifications, the Session ID shall be set to 0x0000.

— If response messages are expected the Session ID shall be incremented for each method call within the same life cycle (starting with 0x0001, also after wrapping).

— The client shall be responsible for invalidating responses to outdated requests if necessary.

6.2.4 Protocol version [8-bit]

Protocol version is an 8-bit field containing the SOME/IP protocol version.

— The protocol version shall be set to 0x01.

6.2.5 Interface major version [8-bit]

Interface major version is an 8-bit field that contains the major version of the service interface. This is required to catch mismatches in service definitions and allow debugging tools to identify the service interface used (see ISO 17215-3).

6.2.6 Message type [8-bit]

The message type field is used to differentiate between different types of messages.

— The message type shall be set to one of the values listed in Table 2.

— A REQUEST shall be answered by a RESPONSE if no error occurred.

— A REQUEST shall be answered by an ERROR if errors occurred.

— A REQUEST_NO_RETURN, a NOTIFICATION or any ACK message shall not be answered.

Table 2 — SOME/IP Message types

Number	Value	Description
0x00	REQUEST	A request expecting a response (even void)
0x01	REQUEST_NO_RETURN	A fire and forget request (client to server)
0x02	NOTIFICATION	A notification/event callback expecting no response (server to client)
(0x40)	REQUEST ACK	Acknowledgement for REQUEST (optional)
(0x41)	REQUEST_NO_RETURN ACK	Acknowledgement for REQUEST_NO_RETURN (optional)
(0x42)	NOTIFICATION ACK	Acknowledgement for NOTIFICATION (optional)
0x80	RESPONSE	The response message
0x81	ERROR	The response containing an error
(0xC0)	RESPONSE ACK	Acknowledgement for RESPONSE (optional)
(0xC1)	ERROR ACK	Acknowledgement for ERROR (optional)

Acknowledgement messages (ACK) may be used in cases where the transport protocol (i.e. UDP) does not guarantee message delivery.

All ACKs are optional and do not need to be implemented.

6.2.7 Return code [8-bit]

The return code is used to signal whether a request was processed successfully. For simplification of the header layout, every message transports the field, whether it is applicable or not.

— Response messages (message type 0x80) shall use the return code field to transmit a return code to the request they answer.

— Error messages (message type 0x81) shall use the return code field to transmit a return code to the request they answer, but shall not use the return code 0x00.

— Acknowledgement messages shall reuse the return code of the message they acknowledge.

— All other messages shall set the return code field to 0x00.

— The return codes are based on an 8-bit Std_returnType of AUTOSAR. The two most significant bits are reserved and shall be set to zero (0). The receiver of a return code shall ignore the values of the two most significant bits.

— The currently defined return codes are listed in Table 3 and shall be implemented as described.

Table 3 — SOME/IP return codes

ID	Name	Description
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	E_UNKNOWN_SERVICE	The requested Service ID is unknown
0x03	E_UNKNOWN_METHOD	The requested Method ID is unknown. Service ID is known.
0x04	E_NOT_READY	Service ID and Method ID are known. Application not running.
0x05	E_NOT_REACHABLE	System running the service is not reachable
a These errors will be specified in future versions of this part of ISO 17215.		
b These error codes are specified by the interface specification for each service.		

Table 3 *(continued)*

ID	Name	Description
0x06	E_TIMEOUT	A timeout occurred
0x07	E_WRONG_PROTOCOL_VERSION	Version of SOME/IP protocol not supported
0x08	E_WRONG_INTERFACE_VERSION	Interface version mismatch
0x09	E_MALFORMED_MESSAGE	Deserialization error (i.e. length or type incorrect)
0x10 - 0x1F	RESERVED	Reserved for generic SOME/IP errors[a]
0x20 - 0x3F	RESERVED	Reserved for application-specific errors[b]
[a] These errors will be specified in future versions of this part of ISO 17215.		
[b] These error codes are specified by the interface specification for each service.		

6.2.8 Payload [variable size]

In the payload field, the parameters are carried. The mechanism for serialization of the parameters is specified in the following section.

6.3 Wire format

The wire format describes data representation in PDUs transported over an IP-based automotive in-vehicle network.

6.3.1 Transport

SOME/IP directly supports the two most common transport protocols of the Internet: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). While UDP is a very lean transport protocol supporting only the most important features (multiplexing and error detecting using a checksum), TCP adds additional features for achieving reliable communication.

— The port numbers for SOME/IP services shall be specified by the OEM.

— UDP should be used if there are hard latency requirements (<100 ms) in case of errors.

— TCP should be used only if large chunks of data need to be transported (>>1 kb) and no hard latency requirements in the case of errors exist.

— The choice of the transport protocol shall be documented for every method in the interface specification.

— Methods, events, and fields should not support the use of more than one transport protocol.

In combination with regular Ethernet, IPv4 and UDP can transport packets with up to 1 472 bytes of data without fragmentation. The IPv6 header requires additional 20 bytes. Fragmentation shall be avoided especially in small systems, so the SOME/IP header and payload should be of limited length. The possible usage of security protocols further limits the maximum size of SOME/IP messages. TCP allows segmented payloads, but AUTOSAR currently limits messages to 4 095 bytes.

Other transport mechanisms (Network File System, IEEE 1722, and so forth) can also be used when they are more suitable for the use case. In this case, SOME/IP is only used to transport a file handle or similar.

6.3.1.1 UDP binding

Many applications inside the vehicle require short timeouts to enable fast reaction. These requirements are better met using UDP, because the application itself can handle the unlikely case of errors in a flexible manner. For example, in use cases with cyclic data, it is often the best approach to just wait for the next

data transmission instead of trying to repair the last one. The major disadvantage of UDP is that it does not handle segmentation, and thus, can only transport small chunks of data.

— SOME/IP messages over UDP can use up to 1 416 bytes for the SOME/IP header and payload.

— The header format allows transporting more than one SOME/IP message in a single UDP packet. The SOME/IP implementation shall identify the end of a SOME/IP message by means of the SOME/IP length field. Based on the UDP lengths field, SOME/IP shall determine if there are additional SOME/IP messages in the UDP packet. The SOME/IP messages can be unaligned.

— Each SOME/IP payload shall have its own SOME/IP header.

— If specified by the interface definition, SOME/IP shall send an acknowledgement message before the result of a long-running operation becomes available (processing acknowledgement).

6.3.1.2 TCP binding

The TCP binding of SOME/IP is heavily based on the UDP binding. In contrast to the UDP binding, the TCP binding allows much larger SOME/IP messages and the batch transport of a large number of SOME/IP messages (pipelining). TCP cannot only handle bit errors, but also packet segmentation, loss, duplication, reordering, and network congestion.

— SOME/IP messages over TCP, including the SOME/IP header, shall not exceed 4 095 Bytes.

— Every SOME/IP payload shall have its own SOME/IP header.

— In order to lower latency and reaction time, Nagle's algorithm should be turned off (TCP_NODELAY).

— When the TCP connection is lost, outstanding requests shall be handled as timeouts. Otherwise, TCP guarantees packet delivery, so additional measures for robustness are not necessary.

— The TCP connection shall be opened by the client, when the first method call is transport or the client tries to receive the first notifications depending on the first occurrence.

— In order to allow resynchronization to SOME/IP over TCP in testing and integration scenarios, the SOME/IP magic cookie message (Figure 4) shall be used between SOME/IP messages over TCP.

— The magic cookie message uses a Service ID of 0xFFFF and a Method ID of 0x0000 (client to server) respectively 0x8000 (server to client).

— The magic cookie message has a length of 8 bytes, a fixed Client ID of 0xDEAD and a fixed Session ID of 0xBEEF. The protocol version is 0x01, the interface version 0x01, the message type 0x01 (client to server) respectively 0x02 (server to client). The return code is 0x00.

— Before the first SOME/IP message is transported in a TCP segment, the SOME/IP magic cookie message shall be included.

— The implementation shall only include up to one SOME/IP magic cookie message per TCP segment.

— If the implementation has no appropriate access to the message segmentation mechanisms and therefore cannot fulfil the two previous requirements, the implementation shall include SOME/IP magic cookie messages once every 10 s in the TCP connection as long as messages are transmitted over this TCP connection.

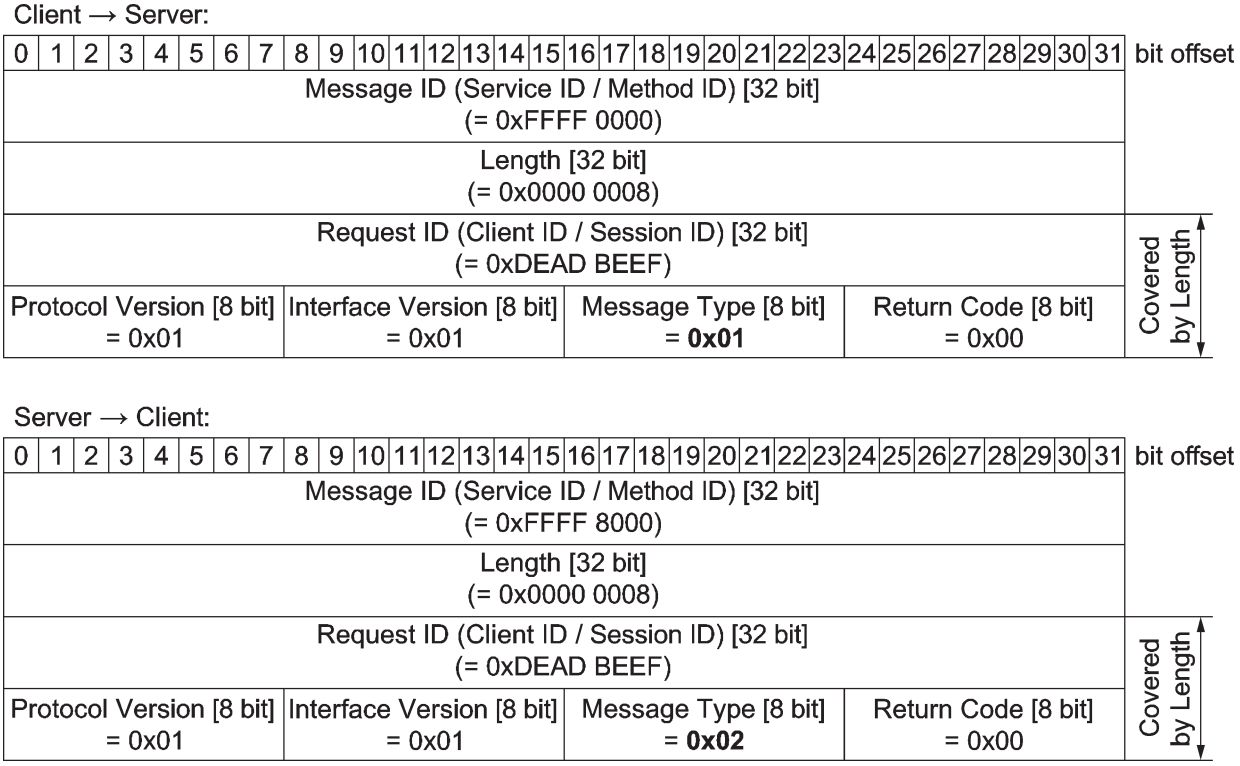

Figure 4 — SOME/IP magic cookie

6.3.2 Mapping of IP addresses and ports

For response and error messages, the IP addresses and port number of the transport protocol shall match the request message. In particular, this means:

— The source IP address of the response message shall be the destination IP address of the corresponding request message.

— The destination IP address of the response message shall be the source IP address of the corresponding request message.

— The source port of the response message shall be the destination port of the corresponding request message.

— The destination port of the response message shall be the source port of the corresponding request message.

6.4 Parameter serialization

An important aspect of the serialization process is the memory alignment of the parameters. This means that the storage address of a data type should be an integer multiple of its size in bytes, e.g. a float32 parameter should be placed at an offset divisible by four.

— The serialization shall not try to automatically align parameters, but shall align them as specified in the interface specification.

— If a SOME/IP implementation encounters an interface specification that leads to a PDU that is not correctly aligned (e.g. because of an unaligned structure), a SOME/IP implementation shall warn about a misaligned structure but shall not fail in generating the code.

— The parameters shall be marked as IN, OUT, or INOUT in the interface specification.

— For messages from client to server, OUT parameters shall be skipped in the serialization/deserialization.

— For messages from server to client, IN parameters shall be skipped in the serialization/deserialization.

— The byte order for each parameter shall be specified by the interface definition. Whenever possible, network byte order (big endian) should be used.

6.4.1 Basic data types

The following basic data types shall be supported:

Table 4 — SOME/IP data types

Type	Description	Size [bit]	Remark
boolean	TRUE/FALSE value	8	FALSE (0), TRUE (1)
uint8	unsigned integer	8	
uint16	unsigned integer	16	
uint32	unsigned integer	32	
sint8	signed integer	8	
sint16	signed integer	16	
sint32	signed integer	32	
float32	floating point number	32	IEEE 754 binary32 (single precision)
float64	floating point number	64	IEEE 754 binary64 (double precision)

6.4.2 Structured data types

The serialization of a structure is based on its in-memory layout. Especially for structures it is important to consider the memory alignment. Insert padding elements in the interface definition if needed for alignment. An example is given in Figure 5.

— The interface specification can add a length field of 8 bits, 16 bits, or 32 bits in front of the structure.

— The length field describes the number of bytes which are transmitted. If the size of the structure as specified in the interface is smaller than the given length, the additional bytes at the end shall be skipped in the deserialization. This is to ease the migration of interfaces.

— If the size of the length field is not specified, a length of 0 has to be assumed and no length field is in the message.

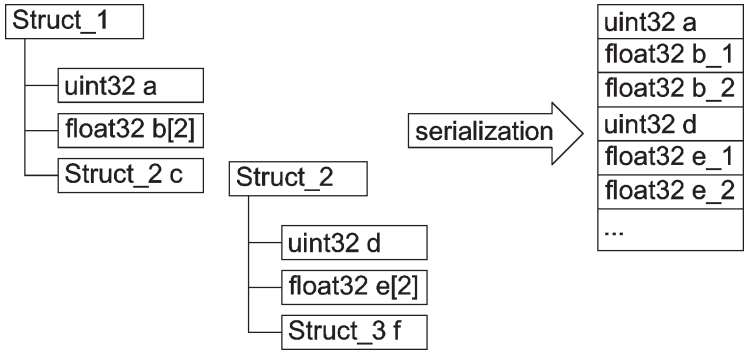

Figure 5 — Serialization of structures

6.4.3 Arrays

Arrays are simply seen as repeated elements. Multidimensional arrays are supported. SOME/IP supports both fixed-size and dynamic-size arrays. Fixed-size arrays are less flexible, but can be more efficient and have better support in earlier versions of AUTOSAR.

NOTE The serialization of multidimensional arrays shall follow the in-memory layout of multidimensional arrays in the C programming language (row-major order).

6.4.3.1 Fixed-size arrays

The size (in bytes) of fixed-size arrays shall be defined by the interface definition. The example layout of a two-dimensional array is illustrated in Figure 6.

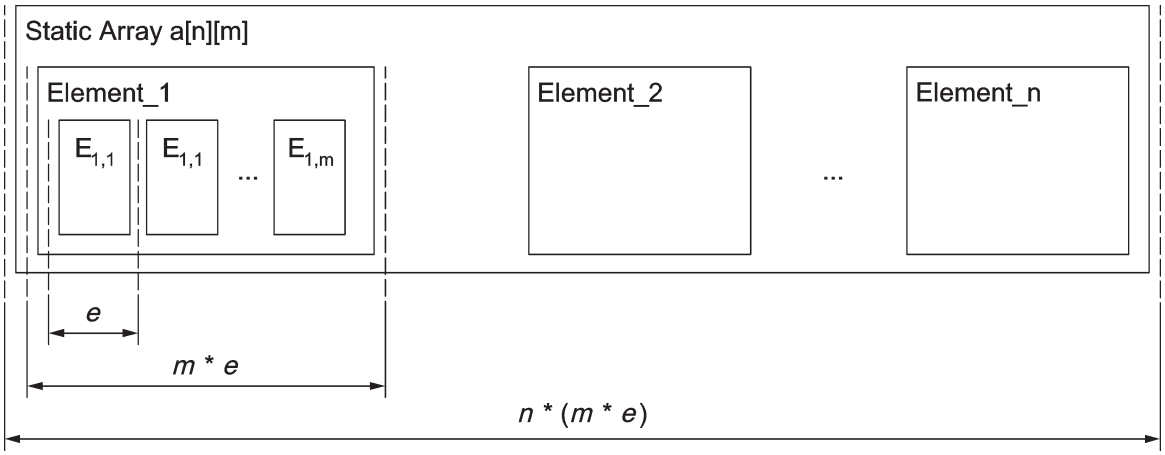

Figure 6 — Fixed-size multidimensional array memory layout

6.4.3.2 Dynamic-size arrays

The layout of arrays with dynamic length is the same as for fixed-length arrays, with additional explicit length fields.

— To determine the size of the array, the serialization shall add a length field of 8 bits, 16 bits, or 32 bits in front of the data, which counts the total bytes of the array.

— The length shall not include the size of the length field.

— Each row shall have its own length field.

— If not specified, otherwise, in the interface specification, the size of the length field is 32 bits.

— The interface definition shall define the maximum length of each dimension to allow static buffer size allocation.

Thus, when transporting an array with zero elements, the length is set to zero. The memory layout of dynamic arrays is shown in Figure 7.

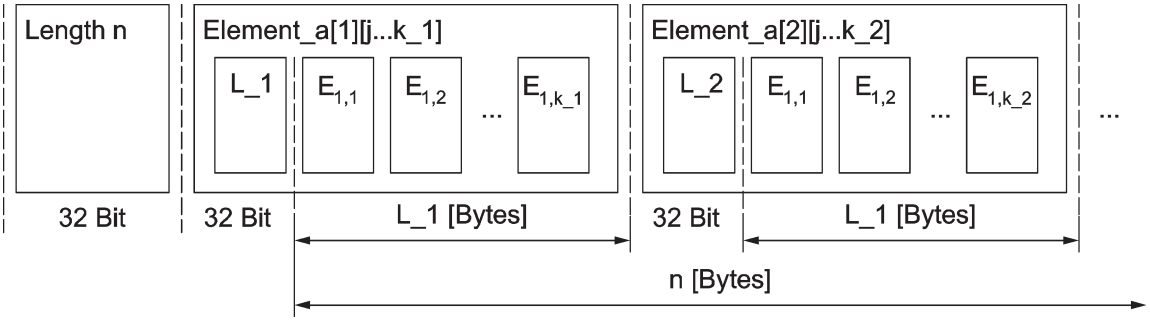

Figure 7 — Dynamic-size multidimensional array memory layout

6.4.3.2.1 Optional fields

Optional fields shall be specified in the interface definition as a dynamic-size array. Its length will then be either 0 or 1, depending on whether the parameter is present or not.

6.4.3.2.2 Map/Dictionary

Maps or dictionaries shall be described as a dynamic-size array of key-value-pair structures. An example of this important data structure can be seen in Table 5.

Table 5 — Serialized dictionary example

Length = 12 bytes	
key0 [uint16]	value0 [uint16]
key1 [uint16]	value1 [uint16]
key2 [uint16]	value2 [uint16]

6.4.4 Strings

Strings are simply one-dimensional byte arrays in UTF-8 encoding. Since this encoding uses a variable number of bytes per character, the length of a string in bytes is different from the number of characters. In the worst case, three times the number of characters plus 4 bytes for the termination with a "\0" and the Byte Order Mark (BOM) might be needed for UTF-8 strings. All strings shall start with a BOM.

6.4.4.1 Strings (fixed length)

The following rules apply for strings with fixed length.

— Strings shall be encoded using UTF-8, UTF-16 BE, or UTF-16 LE and terminated with a NULL character.

— The length of the string (including the terminator) in bytes shall be specified in the interface definition.

— Unused space shall be filled with NULL characters.

6.4.4.2 Strings (dynamic length)

The following requirements apply for strings with dynamic length.

— Dynamic-length strings shall start with a length field.

— The size of the length field shall be either 8 bits, 16 bits, or 32 bits. Fixed-length strings might be seen as having a 0-bit length field.

— If not specified, otherwise, in the interface specification, the size of the length field is 32 bits.

— Strings shall be encoded using UTF-8, UTF-16 BE, or UTF-16 LE and shall be terminated with a NULL character.

— The length field shall be set to the number of bytes occupied by the string (including the terminator).

— The maximum length of the string (including the terminator) in bytes shall be specified in the interface definition.

6.4.5 Union/Variant

A union (also called variant) is a parameter whose contents can be interpreted in different data formats.

— The serialization layout of unions shall consist of a length field and a type field, followed by the actual storage space.

— The length field shall define the size of the parameter storage space, including padding, in bytes and does not include the size of the length field and type field.

— The type field shall describe the type of the parameter. Possible values of the type field are defined by the interface specification for each union separately. The types are encoded as in the interface specification in ascending order starting with 1. The 0 is reserved for the NULL type, i.e. an empty union. The usage of NULL shall be allowed by the interface definition.

— The size of the length field shall be defined by the interface specification and shall be 32 bits, 16 bits, 8 bits, or 0 bit.

— The value of the length field shall be defined by the interface specification. The value shall be high enough to accommodate the largest possible type.

— A length field of 0 bit means that no length field shall be written to the PDU. In this case, all types in the union shall be of the same length.

— If the interface specification does not specify the length of the length field for a union, 32-bit length field shall be used.

— The size of the type field shall be defined by the interface specification and shall be 32 bits, 16 bits, or 8 bits.

— If the interface specification does not specify the length of the type field of a union, 32-bit length of the type field shall be used.

— If an invalid type flag is encountered, an E_MALFORMED_MESSAGE error shall be raised.

— The parameters shall be serialized/deserialized according to their type.

— The serializer shall set the type field and add padding behind the element, if necessary, in accordance with the length field.

— The deserializer shall skip any remaining bytes if the given length is more than the length of the type.

In the following, an example is given for union of uint8/uint16 both padded to 32 bits. In the interface specification, a union of uint8 and uint16 is specified. Both are padded to the 32-bit boundary (length = 4).

Table 6 — Serialization of a uint8 (type 0 is the undefined union)

Length [32 bits] = 4 bytes			
Type [32 bits] = 1			
uint8	Padding	Padding	Padding

Table 7 — Serialization of a uint16

Length [32 bits] = 4 bytes		
Type [32 bits] = 2		
uint16	Padding	Padding

6.4.6 Enumeration

The interface definition might specify an enumeration based on unsigned integer datatypes (uint8, uint16, uint32, uint64).

7 Service discovery protocol specification

7.1 General

In the closed environment of the vehicle, the implemented services and their location are already known a-priori; thus, the service discovery is not used for discovering the location of a service, but its availability. This can be used to communicate whether an Electronic Control Unit (ECU) is running and a service is ready. In addition, the service discovery enables service migration, a feature that can prove valuable for handling optional equipment and implementing power saving strategies.

7.2 Definitions

Offering a service instance shall mean that one ECU implements an instance of a service and tells other ECUs using SOME/IP-SD that they can use it.

Finding a service instance shall mean to send a SOME/IP-SD message in order to find a needed service instance

Requiring a service instance shall mean to send a SOME/IP-SD message to the ECU implementing the required service instance to signal that this service instance is needed by the other ECU. In case the service is not running, the receiving ECU should start it.

Releasing a service instance shall mean to send a SOME/IP-SD message to the ECU hosting this service instances, with the meaning that the service instance is no longer needed.

Publishing an eventgroup shall mean to offer an eventgroup of a service instance to other ECUs using a SOME/IP-SD message.

Subscribing an eventgroup shall mean to require an eventgroup of a service instance using a SOME/IP-SD message.

A service status of *up* shall mean that a service instance is available; thus, it can be accessed using the communication method specified and is able to fulfil its specified function.

A service status of *down* shall mean the opposite of the service status up.

A service status of *required* shall mean that service instance is needed by at least one other software component in the system to function.

A service status of *released* shall mean the opposite of the service status required.

7.3 General requirements

It is assumed that IP addresses for all devices have already been assigned at this stage. See ISO 17215-4 for details on the address assignment process. Dynamic address assignment is done via DHCP. Special emphasis is placed on keeping the start-up time of the system as short as possible.

— Service discovery messages shall be supported over UDP.

— Service discovery messages shall use the Service ID of 0xFFFF.

— Service discovery messages shall use the Method ID of 0x8100.

— Service discovery messages shall have a Client ID and handle it based on SOME/IP rules.

— Service discovery messages shall have a Session ID and handle it based on SOME/IP rules.

— Service discovery messages shall have a protocol version of 0x01 and an interface version of 0x01.

— Service discovery messages shall use the message type notification (0x02) and the corresponding return code 0x00.

— Different instances of the same service within the same vehicle shall have different Instance IDs.

— Instance IDs shall be 16-bit unsigned integers (uint16). For further details, see 8.2.3.

— Instance ID 0x0000 shall not be used

— Instance ID 0xFFFF shall mean “All service instances”.

— An event group shall be identified using the Eventgroup ID.

— Eventgroup IDs shall be 16-bit unsigned integers (uint16).

— Different event groups within the same service shall have different Eventgroup IDs.

Event groups are logical groupings of zero or more events. There are only relevant for SOME/IP-SD to allow easy subscription of multiple events. However, in the actual notifications sent later, only the EventID is present.

Sequence charts for an example use cases can be seen in Figure 8 and Figure 19.

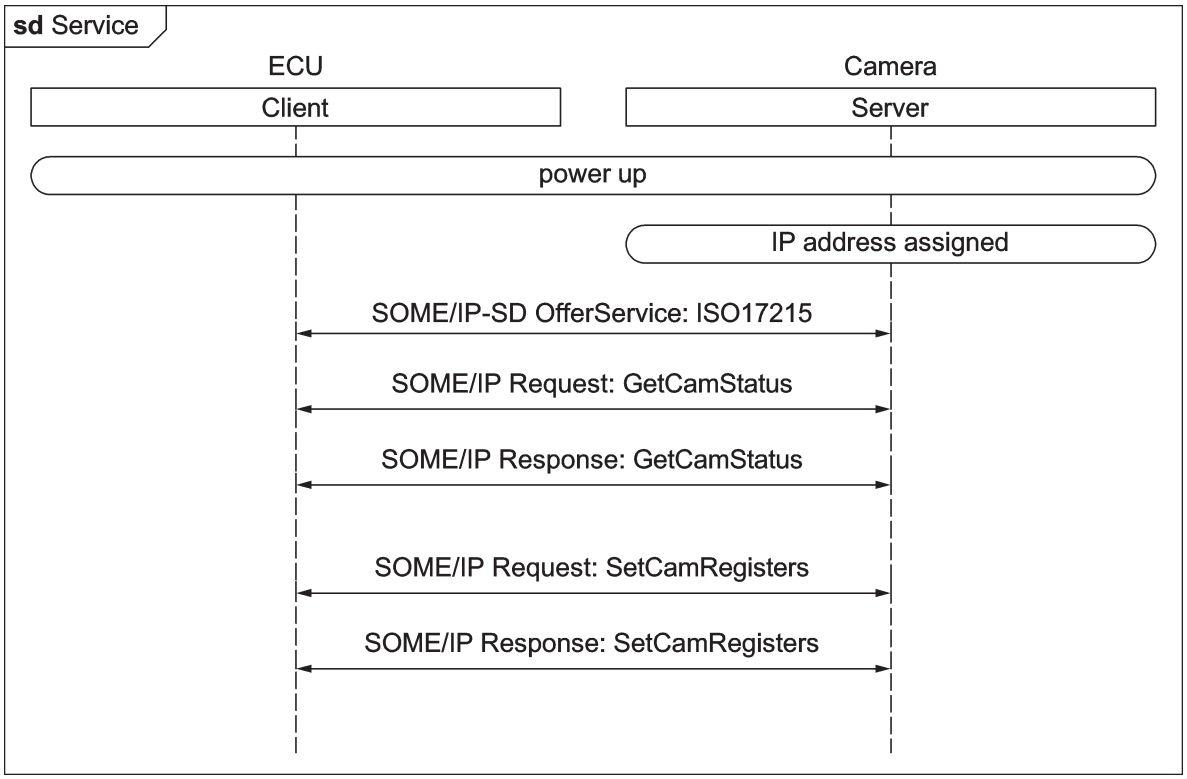

Figure 8 — Service call example

7.4 Service discovery ECU-internal interface

Inside each SOME/IP-SD capable ECU, one SOME/IP-SD implementation shall be running and provide certain functionality to local services.

— The service discovery interface shall inform local software components about the status of remote services (up/down).

— The service discovery interface shall offer the option to local software component to require or release a remote service instance (and then send the required SOME/IP-SD messages).

— The service discovery interface shall inform local software components of the require/release status of local services.

— The service discovery interface shall offer the option to local software component to set a local service status (up/down).

— Eventgroup status shall be defined in the same way the service status is defined.

— In addition to the operations of the services the eventgroups shall start sending events on being required.

— The service discovery shall be informed of link-up and link-down events of logical, virtual, and physical communication interfaces that the service discovery is bound to.

— The service discovery of an ECU shall keep the current state of the service instances this ECU offers as well as of the service instances other ECU offer as well as service instance it needs from other ECUs.

— As optimization, the service discovery shall support keeping only state of service instances that are configured and/or dynamically signalled inside the local ECU.

7.5 Packet format

Service discovery is performed with help of RPC requests to a reserved SOME/IP message ID in conjunction with an extended SOME/IP header including entries and options. This extended header is shown in Figure 9, the entry layout in Figure 11, and example option layouts in Figure 12.

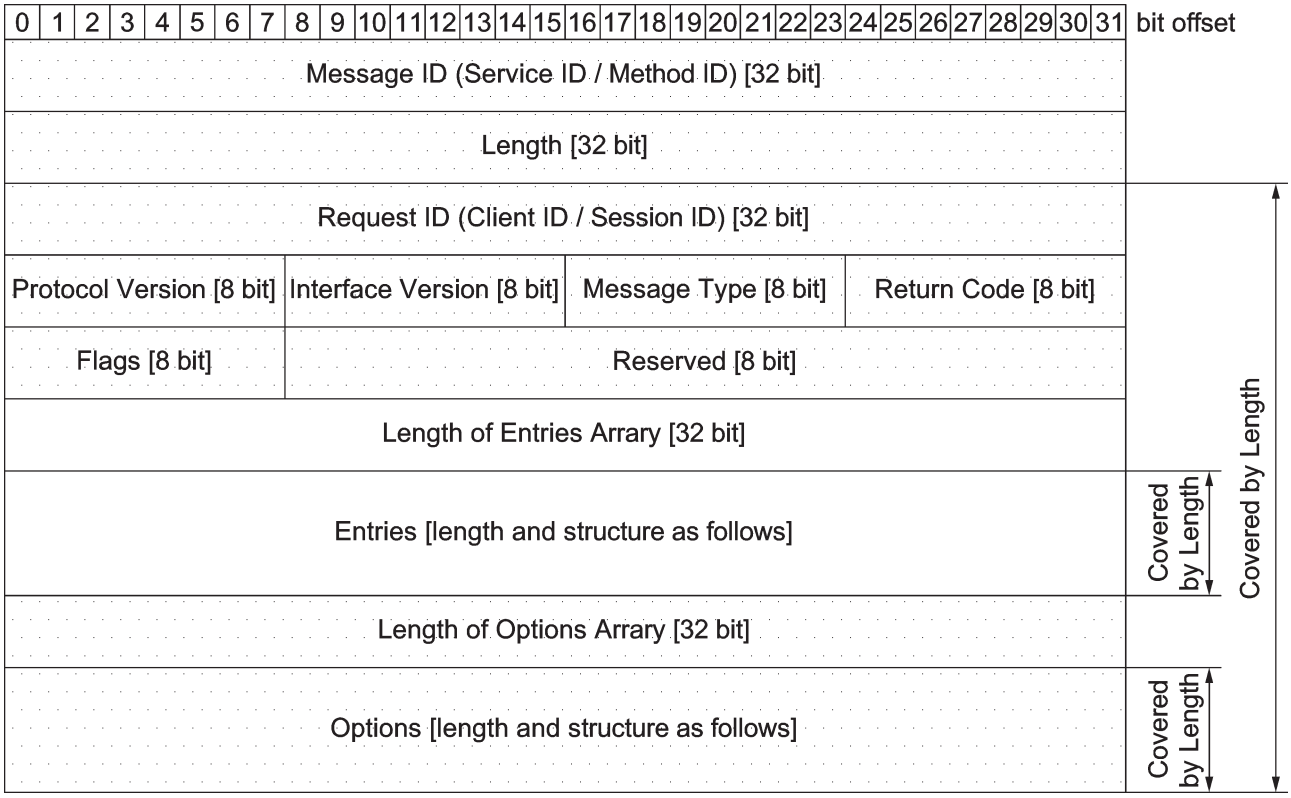

Figure 9 — SOME/IP service discovery packet format

For service discovery, the standard SOME/IP header is followed by the SOME/IP-SD header.

— The SOME/IP-SD header shall begin with an 8-bit field called flags. Only the two highest bits are currently used.

  — Bit 7 of the flag field is the reboot bit. It shall be 1 when the ECU boots and 0 once the Session ID has wrapped around.

  — Bit 6 of the flag field is the unicast bit. It indicates that it is acceptable to reply with a unicast message (see 8.2.1 for further details).

— The unicast flag shall be set to 1 for request messages, subscribe messages, and SubscribeEventgroupAck messages.

— After the flags, the SOME/IP-SD header shall contain 24 reserved bits.

— The SOME/IP-SD header shall be followed by the entries array and the options array.

— The entries array and the options array shall be encoded like SOME/IP arrays, and thus, start with length fields as uint32.

— The length field shall be encoded as the array length field, counting the number of bytes of the array without the length field itself.

7.5.1 Entry records

One SOME/IP service discovery message can contain multiple entries. Each entry record can reference up to two option arrays. The first run is meant to be used for options common to multiple entries, while the second run is meant to be used for options that are only relevant for this one entry. Multiple options

can, for example, be used to announce IPv4 and IPv6 addresses simultaneously, or to include additional configuration data with an announcement.

— Every entry of the array shall be of type A (concerning services) (see Figure 10) or type B (concerning eventgroups) (see Figure 11).

— A type A entry shall be 16 bytes in size and include the following fields in this order:

  — Type field [uint8]: Encodes FindService (0x00),OfferService (0x01) and RequestService (0x02).

  — Index first option run [uint8]: Index of the first option of the first option run in the option array.

  — Index second option run [uint8]: Index of the first option of the second option run in the option array.

  — Number of options 1 [uint4]: Describes the number of options the first option run uses. Zero means no options (empty run).

  — Number of options 2 [uint4]: Describes the number of options the second option run uses. Zero means no options (empty run).

  — If the first option run is empty, the second option run shall be empty as well.

  — If an option run is empty, the index of the first option can be non-zero.

  — Service ID [uint16]

  — Instance ID [uint16]

  — Major version [uint8]

  — TTL [uint24]: Describes the lifetime of the entry in seconds.

  — Minor version [uint32]

— A type B entry shall be 16 bytes in size and include the following fields in this order:

  — Type field [uint8]: Encodes FindEventgroup (0x04), Publish (0x05), SubscribeEventgroup (0x06), SubscribeEventgroupAck (0x07).

  — Index first option run [uint8]: Index of the first option of the first option run in the option array.

  — Index second option run [uint8]: Index of the first option of the second option run in the option array.

  — Number of options 1 [uint4]: Describes the number of options the first option run uses. Zero means no options (empty run).

  — Number of options 2 [uint4]: Describes the number of options the second option run uses. Zero means no options (empty run).

  — If the first option run is empty, the second option run shall be empty as well.

  — If an option run is empty, the index of the first option can be non-zero.

  — Service ID [uint16]

  — Instance ID [uint16]

  — Major version [uint8]: Encodes the major version of the service instance this eventgroup is part of.

  — TTL [uint24]: Describes the lifetime of the entry in seconds.

— Reserved [uint16]: Shall be set to 0x0000.

— Eventgroup ID [uint16]

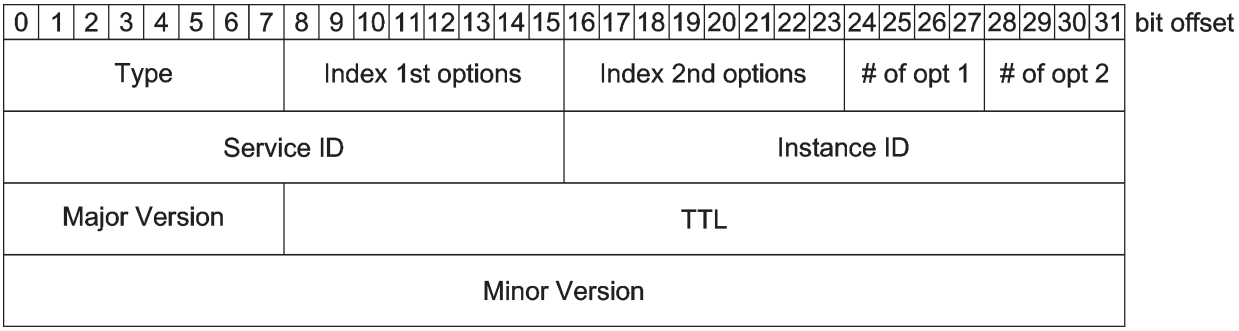

Figure 10 — SOME/IP SD entry layout type A

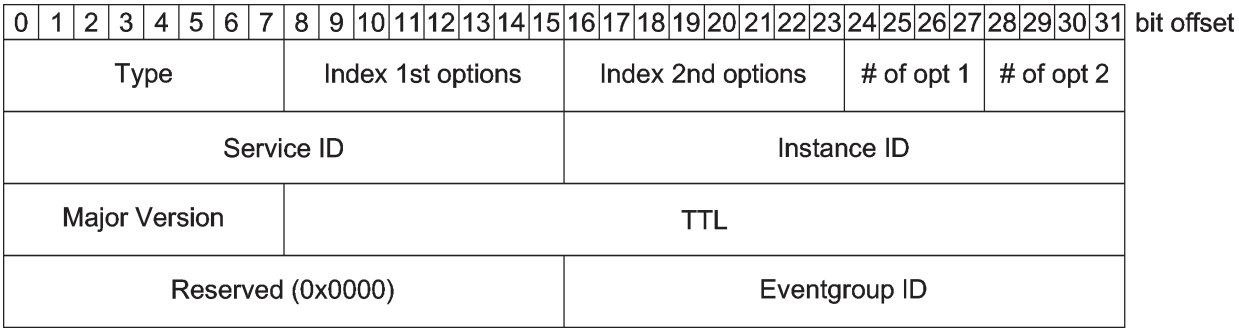

Figure 11 — SOME/IP SD entry layout type B

7.5.1.1 FindService and StopFindService (optional)

The Find Service entry type shall be used for finding service instances and shall only be sent if the current state of a service is unknown (no current service offer was received and is still being valid). If the service in question is not available, there is no response.

— Type shall be set to 0x00 (FindService).

— Service ID shall be set to the Service ID of the service that shall be found.

— Instance ID shall set to 0xFFFF, if all service instances shall be returned. If set to the Instance ID of a specific service instance, just this single service instance shall be returned.

— Major version shall be set to 0xFF. That means that services with any version shall be returned. If set to value different than 0xFF, services with this specific major version shall be returned only.

— Minor version shall be set to 0xFFFF FFFF. That means that services with any version shall be returned. If set to a value different to 0xFFFF FFFF, services with this specific minor version shall be returned only.

— TTL shall be set to the lifetime of the find service entry. After this lifetime, the find service entry shall be considered as not existing.

— If set to 0xFFFFFF, the find service entry shall be considered valid for ever.

— The StopFindService feature is optional.

— The Stop Find Service entry type shall be used to stop finding service instances.

— Stop find service entries shall be used in communication with service directories.

— Stop find service entries shall set the header fields exactly like the find service entry they are stopping, except that TTL shall be set to 0x000000.

7.5.1.2 OfferService and StopOfferService

The Offer Service entry type shall be used to offer a service to other communication partners.

— Type shall be set to 0x01 (OfferService).

— Service ID shall be set to the Service ID of the service instance offered.

— Instance ID shall be set to the Instance ID of the service instance offered.

— Major version shall be set to the major version of the service instance offered.

— Minor version shall be set to the minor version of the service instance offered.

— TTL shall be set to the lifetime of the service instance. After this lifetime, the service instance shall be considered as not offered.

— If set to 0xFFFFFF, the Offer Service entry shall be considered valid forever.

— The Stop Offer Service entry type shall be used to stop offering service instances.

— Stop Offer Service entries shall set the header fields exactly like the Offer Service entry they are stopping, except that TTL shall be set to 0x000000.

7.5.1.3 RequestService (optional) and StopRequestService (optional)

The Request Service entry type shall be used to indicate that a service instance is required.

— The RequestService and the StopRequestService features are optional.

— An ECU shall consider a Request Service entry as a reason to start the specified service instance if configured to do so.

— Type shall be set to 0x02 (RequestService).

— Service ID shall be set to the Service ID of the service instance requested.

— Instance ID shall be set to the Instance ID of the service instance requested.

— Major version shall be set to 0xFF (any version).

— Minor version shall be set to 0xFFFF FFFF (any version).

— TTL shall be set to the lifetime of the request. After this lifetime, the service request shall be considered non-existing. This can lead to an ECU shutting down a service previously requested.

— If set to 0xFFFFFF, the Request Service entry shall be considered valid for ever.

— The Stop Request Service entry type shall be used to stop requests.

— Stop Offer Request entries shall set the header fields exactly like the Request Service entry they are stopping, except that TTL shall be set to 0x000000.

7.5.1.4 FindEventgroup (optional) and StopFindEventgroup (optional)

The Find Eventgroup entry type shall be used for finding eventgroups.

— The FindEventgroup and the StopFindEventgroup features are optional.

— Type shall be set to 0x04 (FindEventgroup).

— Service ID shall be set to the Service ID of the service that includes the eventgroup that shall be found.

— Instance ID shall set to 0xFFFF, if eventgroups of all service instances shall be returned. It shall be set to the Instance ID of a specific service instance, if just the eventgroup of a single service instance shall be returned.

— Major version shall be set to 0xFF, meaning that eventgroups of services with any version shall be returned. If set to value different than 0xFF, eventgroups of services with this specific major version shall be returned only.

— Minor version shall be set to 0xFFFF FFFF, meaning that eventgroups of services with any version shall be returned. If set to a value different to 0xFFFF FFFF, eventgroups of services with this specific minor version shall be returned only.

— TTL shall be set to the lifetime of the Find Eventgroup entry. After this lifetime, the Find Eventgroup entry shall be considered as not existing.

— If set to 0xFFFFFF, the FindEventgroup entry shall be considered valid for ever.

— Eventgroup ID shall set to the ID of the eventgroup that is being looked for. Setting the Eventgroup ID to 0xFFFF (all eventgroups) is currently not recommended, but when receiving an Eventgroup ID of all, the ECU shall answer for all Eventgroups.

— The Stop Find Eventgroup entry type shall be used to stop finding eventgroups.

— Stop Find Eventgroup entries shall be used in communication with service directories.

— Stop Find Eventgroup entries shall set the header fields exactly like the Find Eventgroup entry they are stopping, except that TTL shall be set to 0x000000.

7.5.1.5 PublishEventgroup (optional) and StopPublishEventgroup (optional)

The Publish Eventgroup entry type shall be used to offer a eventgroup to other communication partners.

— The PublishEventgroup and the StopPublishEventgroup features are optional.

— Type shall be set to 0x05 (PublishEventgroup).

— Service ID shall be set to the Service ID of the service instance that includes the eventgroup published.

— Instance ID shall be set to the Instance ID of the service instance that includes the eventgroup published.

— Major version shall be set to the major version of the service instance that includes the eventgroup published.

— Minor version shall be set to the minor version of the service instance that includes the eventgroup published.

— TTL shall be set to the lifetime of the eventgroup. After this lifetime, the eventgroup shall considered not been published.

— In most use cases, eventgroups will have the same lifetime as the service instance they are part of. A longer lifetime of the eventgroup than the lifetime of the service instance it is part of shall not be allowed.

— If set to 0xFFFFFF, the Publish Eventgroup entry shall be considered valid for ever.

— Eventgroup ID shall be set to the ID of the eventgroup.

— The Stop Publish Eventgroup entry type shall be used to stop publishing eventgroups.

— Stop Publish Eventgroup entries shall set the header fields exactly like the Publish Eventgroup entry they are stopping except that TTL shall be set to 0x000000.

7.5.1.6 SubscribeEventgroup and StopSubscribeEventgroup

The Subscribe Eventgroup entry type shall be used to subscribe to an eventgroup. This can be considered comparable to the Request Service entry type. Subscribe Eventgroup entries shall reference one or two IPv4 and/or one or two IPv6 endpoint options (one for UDP, one for TCP).

— Type shall be set to 0x06 (SubscribeEventgroup).

— Service ID shall be set to the Service ID of the service instance that includes the subscribed eventgroup.

— Instance ID shall be set to the Instance ID of the service instance that includes the subscribed eventgroup.

— Major version shall be set to the major version of the service instance that includes the subscribed eventgroup.

— Minor version shall be set to the minor version of the service instance that includes the subscribed eventgroup.

— TTL shall be set to the lifetime of the subscription. After this lifetime, the eventgroup shall be considered not subscribed.

— If set to 0xFFFFFF, the SubscribeEventgroup entry shall be considered valid for ever.

— Eventgroup ID shall be set to the Eventgroup ID of the eventgroup subscribed to.

— The Stop Subscribe Eventgroup entry type shall be used to stop subscribing to eventgroups.

— Stop Subscribe Eventgroup entries shall set the header fields exactly like the Subscribe Eventgroup entry they are stopping, except that TTL shall be set to 0x000000.

7.5.1.7 SubscribeEventgroupAck and SubscribeEventgroupNack

The SubscribeEventgroupAcknowledgement entry type shall be used to indicate that SubscribeEventgroup was accepted.

— Type shall be set to 0x07 (SubscribeEventgroupAck).

— Service ID, Instance ID, major version, minor version, and TTL shall be the same value as in the subscription that is being answered.

— The Subscribe Eventgroup Negative Acknowledgement entry shall be used to indicate that a SubscribeEventgroup entry was not accepted.

— SubscribeEventgroupNack entries shall set the header fields exactly like SubscribeEventgroupAck entries, except that TTL shall be set to 0x000000.

7.5.2 Option records

Option records have a different layout depending on their type. Options are used to transport additional information to the entries. There are options for text-based configuration parameters and IPv4/IPv6 endpoints. In order to identify the option type, every option shall start with:

— Length [uint16]: Specifies the length of the option in bytes.

— Type [uint8]: Specifying the type of the option.

— Based on the SOME/IP Union format, the length field shall not cover the size of the length field and type field.

7.5.2.1 Configuration option

The configuration option specifies a set of key-value pairs. The format of the configuration option shall be as follows:

— Length [uint16]: Shall be set to the total number of bytes occupied by the configuration option, excluding the 16-bit length field and the 8-bit type flag.

— Type [uint8]: Shall be set to 0x01.

— Reserved [uint8]: Shall be set to 0x00.

— Each key-value pair shall start with an 8-bit length field that contains the number of bytes occupied by the pair (maximum length for a single pair is thus 255 bytes).

— After each byte sequence, another length field and following byte sequence are expected until a length field is set to 0x00.

— A byte sequence shall encode a key and optionally value.

— The separator between the name and value shall be the equal sign ("=", 0x3D).

— The key shall consist of only printable ASCII characters (0x20 to 0x7F). It shall not include an equal sign and shall contain at least one non-whitespace character.

— For a byte sequence without an ' = ' sign, that key shall be interpreted as present.

— For a byte sequence ending on an ' = ' sign, that key shall be interpreted as present with empty value.

— The individual key-value pairs shall not be zero-terminated (but the complete configuration string is).

The configuration option shall also be used to provide a hostname, servicename, and an instancename (if needed).

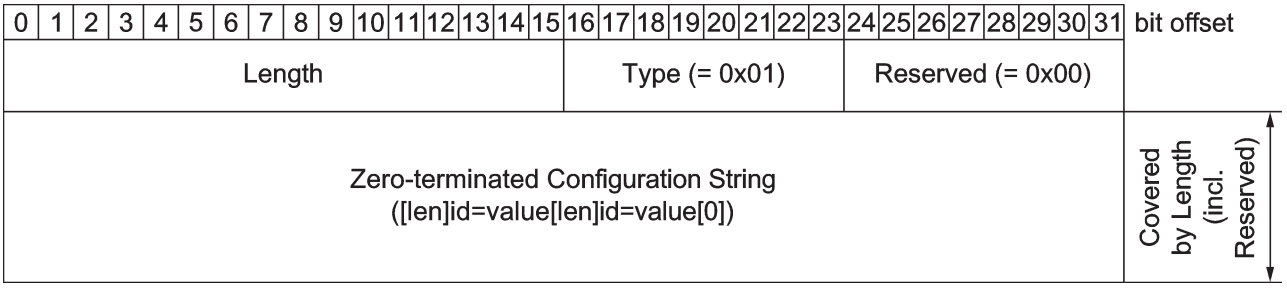

Figure 12 — SOME/IP SD configuration option layout

7.5.2.2 IPv4 endpoint option

If IPv4 is required for the service, the IPv4 endpoint option shall be used by a SOME/IP-SD instance to signal the relevant endpoint(s). Endpoints include the local IP address, the transport layer protocol (UDP or TCP), and the port number of the sender.

These ports shall be used for the events and notification events as well. When using UDP, the server uses the announced port as source port; and with TCP, the client needs to open a connection to this port before subscription, so that the server can use this TCP connection.

The format of the IPv4 endpoint option shall be as follows.

— Length [uint16]: Shall be set to 9.

— Type [uint8]: Shall be set to 0x04.

— Reserved [uint8]: Shall be set to 0x00.

— IPv4-address [uint32]: Shall transport the IP address as 4 bytes.

— Reserved [uint8]: Shall be set to 0x00.

— L4-Proto [uint8]: Shall be set to the relevant layer 4 protocol based on the IANA/IETF protocol numbers (6: TCP, 17:UDP).

— L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

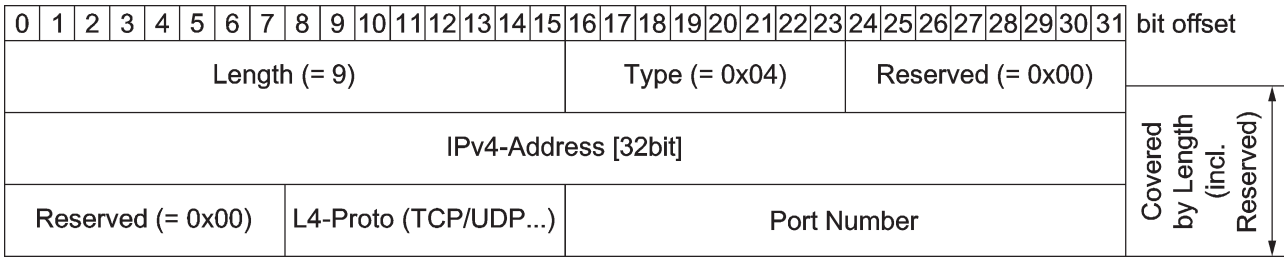

Figure 13 — SOME/IP SD IPv4 endpoint option layout

7.5.2.3 IPv6 endpoint option

The IPv6 endpoint option shall be used by a SOME/IP-SD instance to signal the relevant endpoint(s). Endpoints include the local IP address, the transport layer protocol (e.g. UDP or TCP), and the port number of the sender.

These ports shall be used for the events and notification events as well. When using UDP, the server uses the announced port as source port; and with TCP, the client needs to open a connection to this port before subscription, so that the server can use this TCP connection.

The format of the IPv6 endpoint option shall be as follows.

— Length [uint16]: Shall be set to 21.

— Type [uint8]: Shall be set to 0x06.

— Reserved [uint8]: Shall be set to 0x00.

— IPv6 address [uint128]: Shall transport the IP address as 16 bytes.

— Reserved [uint8]: Shall be set to 0x00.

— L4-Proto [uint8]: Shall be set to the relevant layer 4 protocol based on the IANA/IETF protocol numbers (6: TCP, 17:UDP).

— L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

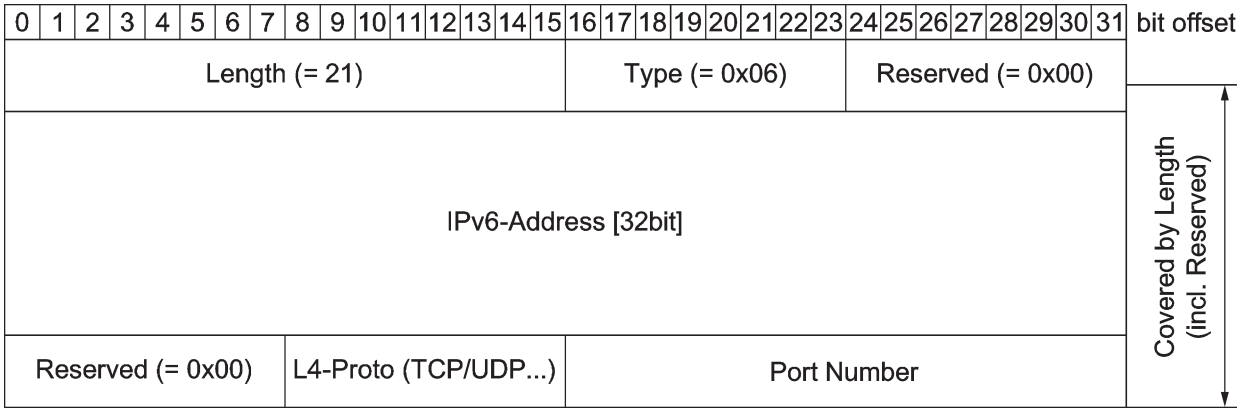

Figure 14 — SOME/IP SD IPv6 endpoint option layout

7.5.2.4 IPv4 multicast option

The IPv4 multicast option is used by the server to announce the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4), and the port number the multicast events and multicast notification events are sent to. As transport layer protocol, currently, only UDP is supported.

The format of the IPv4 multicast option shall be as follows.

— Length [uint16]: Shall be set to 0x0009.

— The IPv4 multicast option shall use the type 20.

— Reserved [uint8]: Shall be set to 0x00.

— IPv4 address [uint32]: Shall transport the multicast IP address as 4 bytes.

— L4-Proto [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).

— L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

The IPv6 multicast option and not the IPv6 endpoint option shall be referenced by SubscribeEventgroupAck messages.

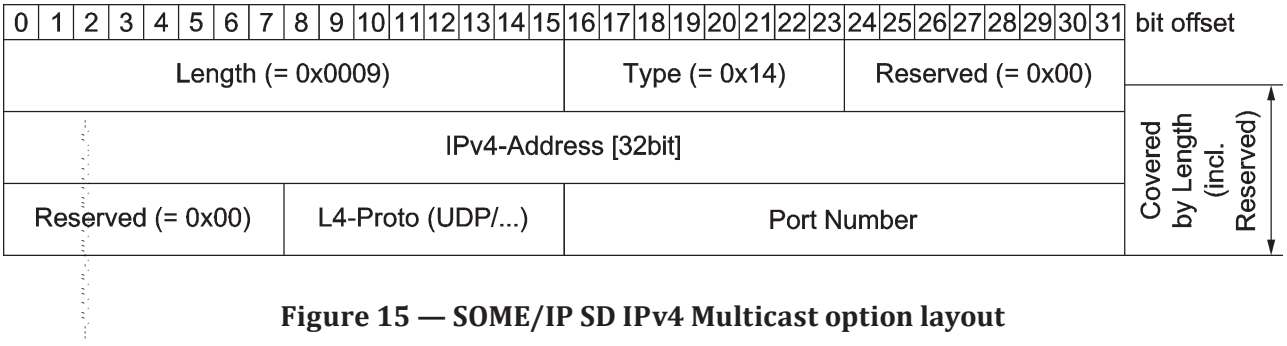

Figure 15 — SOME/IP SD IPv4 Multicast option layout

7.5.2.5 IPv6 multicast option

The IPv4 multicast option is used by the server to announce the IPv6 multicast address, the transport layer protocol (ISO/OSI layer 4), and the port number the multicast events and multicast notification events are sent to. As transport layer protocol, currently, only UDP is supported.

The format of the IPv6 multicast option shall be as follows.

— Length [uint16]: Shall be set to 0x0021.

— The IPv4 multicast option shall use the type 22.

— Reserved [uint8]: Shall be set to 0x00.

— IPv6 address [uint32]: Shall transport the multicast IP address as 16 bytes.

— L4-Proto [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).

— L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

The IPv4 multicast option and not the IPv4 endpoint option shall be referenced by SubscribeEventgroupAck messages.

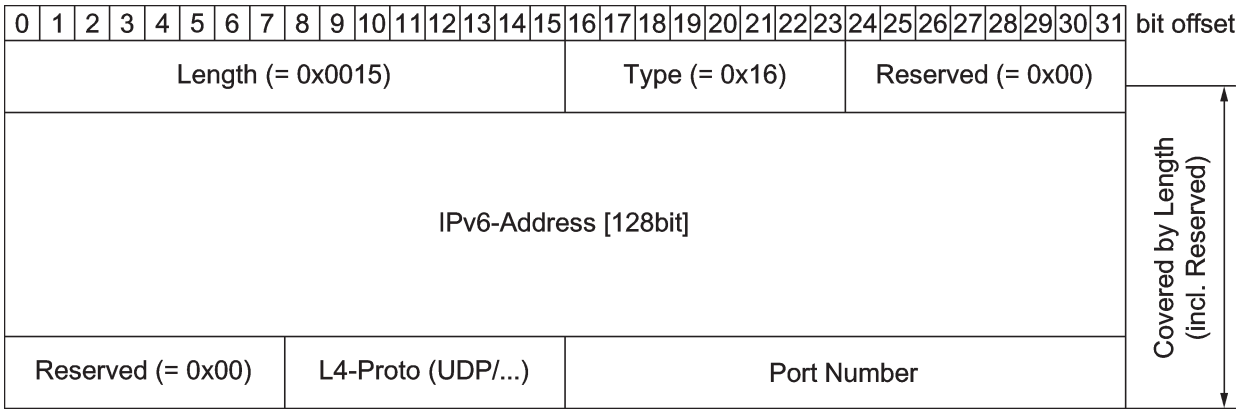

Figure 16 — SOME/IP SD IPv4 Multicast option layout

7.5.3 Example PDU

Figure 17 shows an example service discovery packet containing two entries and one option. The IP header and the UDP header are only shown in abstract form.

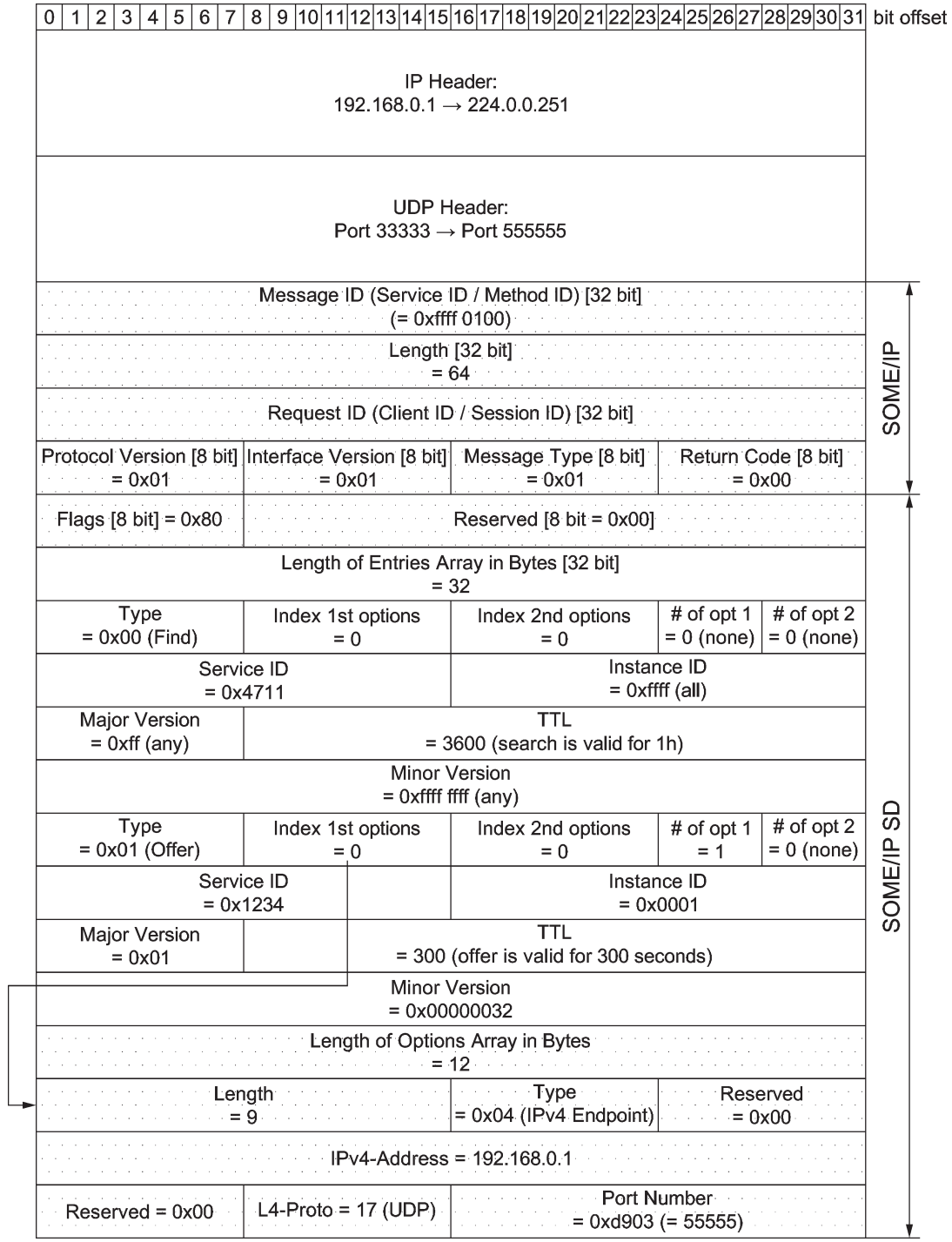

Figure 17 — Example SOME/IP SD PDU

8 Runtime behaviour

8.1 General

This clause describes the runtime behaviour of SOME/IP and SOME/IP-SD.

8.2 SD communication

8.2.1 General

— Whenever possible, the implementation shall combine multiple operations (e.g. FindService, OfferService, Publish, Subscribe) into one packet to reduce network load.

— All delay values used below shall be specified as a minimum and maximum delay. Each service instance shall randomly choose a concrete delay value from the interval.

— The delay and timeout values shall be chosen by the system integrator. Generally, timeout values should be below 2 s to avoid lengthy “hangs”.

— Caching can be implemented. The service discovery implementation can temporarily store information locally to reduce the amount of network queries. To allow for cleanup of stale client registrations (to avoid that the list of listeners fills over time), a cleanup mechanism might be required.

— Query for Service ID shall produce a result (the answers can come from different hosts), the result being a list of records for all service instances, which have the given Service ID.

— Query for the combination (Instance ID and Service ID) shall produce a result (the answers can come from different hosts), the result being a list of records for all service instances, which have the requested (Instance ID and Service ID) in common.

— The server shall delay the answer to FindService and FindEventgroup messages transported by multicast/broadcast for a delay based on REQUEST_RESPONSE_DELAY. This is to avoid flooding the network. The server shall not delay unicast messages that are answered with unicast messages.

— RequestService, SubscribeEventgroup, SubscribeEventgroupAck, and SubscribeEventgroupNack messages shall be sent as unicast.

— FindService, OfferService, FindEventgroup, and PublishEventgroup messages should be sent as multicast.

— FindService/FindEventgroup (respectively RequestService, SubscibeEventgroup) messages received with the unicast flag set to 1, shall be answered with a unicast response if the last announcement was sent less than 1/2 CYCLIC_OFFER_DELAY (respectively 1/2 CYCLIC_REQUEST_DELAY) ago.

— FindService/FindEventgroup (respectively RequestService, SubscibeEventgroup) messages received with the unicast flag set to 1, shall be answered with a multicast response if the last announcement was 1/2 CYCLIC_OFFER_DELAY (respectively 1/2 CYCLIC_REQUEST_DELAY) or more ago.

— FindService/FindEventgroup messages received with unicast flag set to 0 (multicast), shall be answered with a multicast response.

8.2.2 Startup

The startup behaviour of the particular service instances or Eventgroups has three phases: initial, repetition, and main. The phases are illustrated in Figure 18.

— After the network link has been established, the service discovery implementation shall be in the initial phase and do nothing for INITIAL_DELAY milliseconds.

NOTE After sending the first message, the system enters the repetition phase.

— The service discovery shall send out up to REPETITIONS_MAX announcements during the repetitions phase with a delay of REPETITIONS_BASE_DELAY*2^N, where N is increasing from 0 to (REPETITIONS_MAX-1).

— Sending find entries shall be stopped after receiving the corresponding offer entries by jumping to the main phase in which no find entries are sent.

NOTE After the repetitions phase, the main phase is entered.

— In the main phase, offer messages shall be sent periodically if a CYCLIC_OFFER_DELAY is configured.

— After entering the main phase, CYCLIC_OFFER_DELAY is waited before sending the first message.

— After a message for a specific service instance, the service discovery waits for CYCLIC_OFFER_DELAY before sending the next message for this service instance.

— For requests/subscriptions, the same cyclic behaviour as for the offers shall be implemented with the parameter CYCLIC_REQUEST_DELAY instead of CYCLIC_OFFER_DELAY.

— For find entries (Find Service and Find Eventgroup) no cyclic messages shall be sent in the main phase.

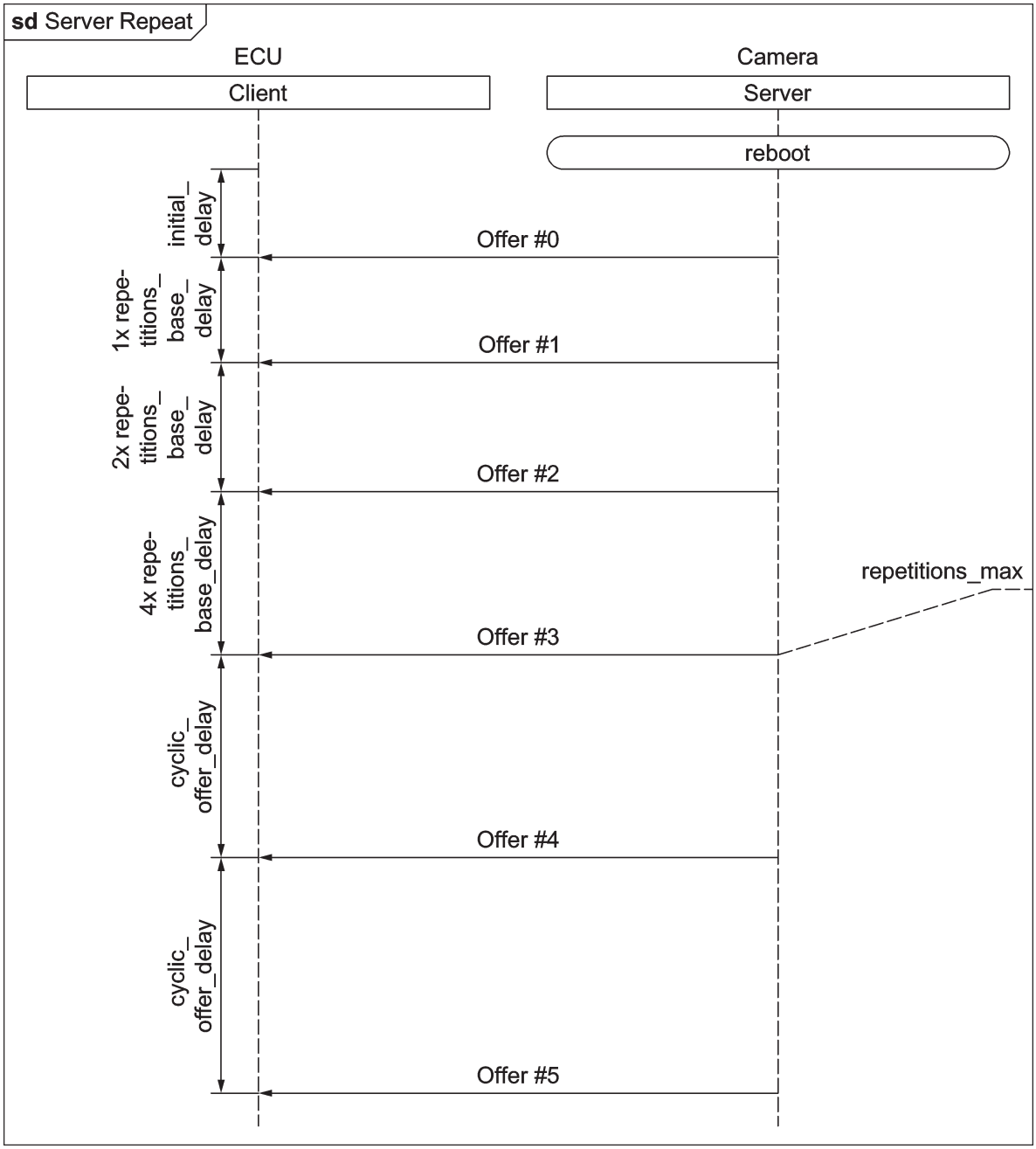

Figure 18 — Startup behaviour

8.2.3 Multiple service instances

Instances of the same service are identified through different Instance IDs. Multiple service instances can reside on different ECUs and also multiple service instances of one or more services can reside on one single ECU. The Instance IDs are used for service discovery, but are not contained in the RPC header. Multiple instances of the same service on a single ECU shall listen on different ports. A service instance can be identified through the socket (i.e. the combination of IP address, transport protocol, and port number). It is recommended that instances use the same port number for UDP and TCP.

— The Instance ID shall be chosen so that it is guaranteed to be unique in the local network (in combination with the Service ID).

— The Service Instance IDs of 0x0000 and 0xFFFF shall not be used for a service, since 0x0000 is reserved and 0xFFFF is used to describe all service instances.

— If a service instance uses UDP port x, exactly this instance should use exactly TCP port x for its services.

Possibilities to obtain unique Instance IDs include computing them from the unique IP address directly, or using a lookup table mapping IP addresses to Instance IDs. This table has to be flashed into the camera but it can support multiple instances per IP address.

To avoid confusion, the following paragraphs highlight some of the differences between the ClientID and the InstanceID fields.

The Client ID is configured by the ECU manufacturer. It is unique within each ECU (independent of the Service ID) and used in every SOME/IP header. Its purpose is to allow SOME/IP to relay incoming PDUs to the correct process, as the socket information has been stripped away by AUTOSAR at this point.

The Instance ID is configured by the OEM. In combination with the Service ID, it is unique within the vehicle and it is used in SOME/IP-SD messages only. It is needed for SOME/IP-SD to differentiate between multiple instances of the same service.

8.2.4 Subscription handling

If a client wishes to subscribe to an Eventgroup, it shall send a SubscribeEventgroup message to the server offering the Eventgroup.

— Upon receiving a subscription message, the server shall reply with a SubscribeEventgroupAck if it accepts the subscription.

— Upon receiving a subscription message, the server shall reply with a SubscribeEventgroupNack if it does not accept the subscription, e.g. because the eventgroup is not available or resources are exhausted.

— Upon receiving a subscription message, the server offering the notifications shall send “initial value” notifications for all relevant events.

— If a client wishes to subscribe to an Eventgroup but does not know which servers offer it, it shall send a FindService message.

— A server shall stop the publishing of notifications by sending a StopOfferService message for the corresponding Eventgroup.

— A client shall unsubscribe from a notification by sending a StopSubscribe message for the corresponding Eventgroup.

— The SOME/IP-SD on the server shall cancel the subscription if a relevant SOME/IP error is received after sending a notification.

— If the server loses its Ethernet link, it shall delete all the registered subscriptions.

— If the Ethernet link of the server comes up again, it shall trigger a SOME/IP-SD OfferService message.

— If the notification client does not receive the expected notification message for a certain time, it shall send out a new SubscribeEventgroup to the server. This timeout is to be specified in the interface definition for each eventgroup.

— A link-up event on the clients Ethernet link shall trigger a SOME/IP-SD SubscribeEventgroup message.

An example sequence chart can be seen in Figure 19.

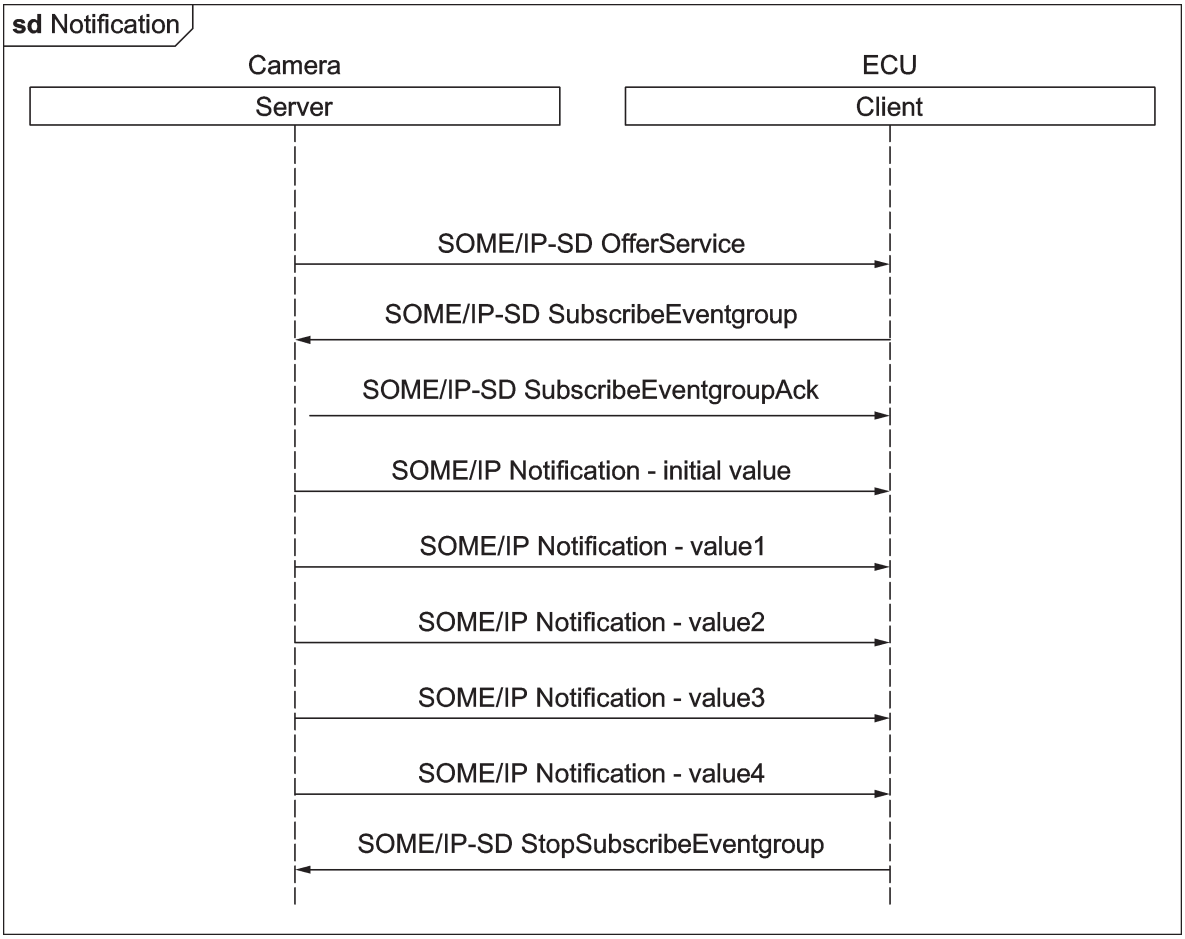

Figure 19 — Notification subscription

8.2.5 Endpoint handling

This section describes how the endpoints encoded in the endpoint and multicast options shall be set and used. The service discovery shall overwrite IP addresses and port numbers with those transported in endpoint and multicast options if the statically configured values are different from those in these options.

8.2.5.1 Service endpoints

— Offer service entries shall reference up to one UDP endpoint option and up to one TCP endpoint option. Both shall be of the same version Internet Protocol (IPv4 or IPv6).

— The referenced endpoint options of the offer service entries denote the IP address and port numbers the service instance can be reached at the server.

— The referenced endpoint options of the offer service entries also denote the IP address and port numbers the service instance sends the events from. Events of this service instance shall be sent only from this IP address and ports.

— If an ECU offers multiple service instances, SOME/IP messages of these service instances shall be differentiated by the information transported in the endpoint options referenced by the offer service entries. Therefore, transporting an Instance ID in the SOME/IP header is not required.

8.2.5.2 Eventgroup endpoints

— Subscribe Eventgroup entries shall reference up to one UDP endpoint option and up to one TCP endpoint option for the Internet Protocol used (IPv4 or IPv6).

— The endpoint options referenced in the subscribe eventgroup entries is used to send unicast UDP or TCP SOME/IP events for this service instance to, i.e. the IP address and the port numbers on the client side.

— TCP events are transported using the TCP connection the client has opened to the server before sending the Subscribe Eventgroup entry.

— The initial events shall be transported using unicast from server to client.

— Subscribe Eventgroup Ack entries shall reference up to one multicast option for the Internet Protocol used (IPv4 or IPv6).

— The multicast option shall be set to UDP as transport protocol.

If the server has to send multicast events very shortly (<5 ms) after sending the Subscribe Eventgroup Ack entry, the server shall try to delay these events, so that the client is not missing it. If this event was sent as initial event anyhow, the server can send this event using unicast as well.

Figure 20 below shows an example for the usage of eventgroup endpoints for the subscription.

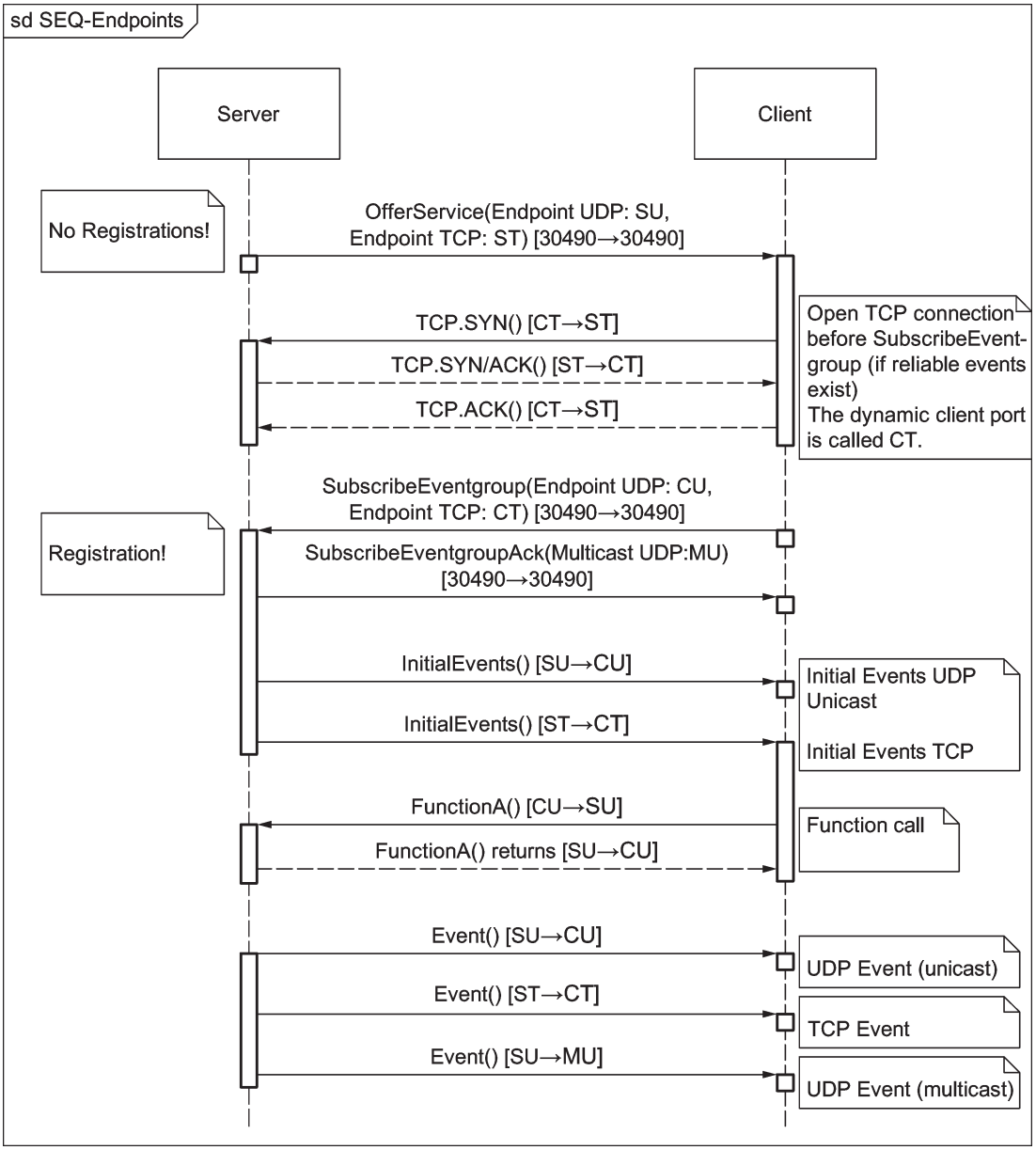

Figure 20 — Publish/Subscribe example for endpoint options and the usage of ports

8.2.6 Announcing non-SOME/IP protocols with SOME/IP-SD

Besides SOME/IP, other communication protocols are used within the vehicle; e.g. for network management, diagnosis, or flash updates. Such communications protocols might need to communicate a service instance or have eventgroups as well. For non-SOME/IP protocols, a special Service ID shall be used and further information shall be added using the configuration option.

— Service ID for non-SOME/IP services shall be set to 0xFFFE (reserved).

— Instance ID shall be used as described for SOME/IP services and eventgroups.

— The configuration option shall be added and shall contain at least an entry with key “otherserv” and a configurable non-empty value that is determined by the system integrator.

— SOME/IP services shall not use the otherserv-string in the configuration option.

8.2.7 Conflict resolution

Theoretically, there can be conflicts over IP addresses, Instance IDs, or even Service IDs. This is a sign of a misconfigured vehicle and should not occur in the field.

— In case of a conflict, the server shall announce that its service is being removed (TTL = 0 in a normal announcement message).

— The reason for the removal of a service shall be accessible to developers and system integrators by inspection of the local error log.

8.3 RPC communication

8.3.1 Request/Response communication

One of the most common communication patterns is the request/response pattern. One party (the client) sends a request message, which is answered by another party (the server). To construct the SOME/IP header of the request message, the client has to perform the following steps:

— construct the payload;

— set the Message ID based on the method the client wants to call;

— set the length field to 8 bytes + length of the serialized payload;

— optionally, set the Request ID to a unique number (shall be unique for client only);

— set the protocol version to 0x01;

— set the interface version according to the interface definition;

— set the message type to 0x00 (REQUEST);

— set the return code to 0x00 (reserved).

For the reply, the server has to

— construct the payload,

— set the Message ID to the value received in the client's request,

— set the length to the 8 Bytes + new payload size,

— set the Request ID to the value received in the client's request,

— set the protocol version to 0x01,

— set the interface version according to the interface definition,

— set the message type to 0x80 (RESPONSE) or 0x81 (ERROR), and

— set the return code (see Table 4 for possible values).

8.3.2 Fire and forget communication

Requests without response message are called fire and forget. The implementation is basically the same as a request/response with the following differences:

— The message type is set to 0x01 (REQUEST_NO_RETURN).

— There is no response message.

Fire and forget messages do no return an error. Error handling and return codes shall be implemented by the application when needed.

8.3.3 Notifications

Notifications describe a general Publish/Subscribe concept. Usually, the server publishes a service to which a client subscribes (using SOME/IP-SD). On certain events, the server will send the client an event, which could be, for instance, an updated value or an event that occurred.

— No responses shall be sent upon receiving notification messages.

— The frequency with which updates will be sent shall be specified in the interface definition. Different modes are possible, e.g. periodic update or update on change/delta change.

When more than one subscribed client on the same ECU exists, the system should handle the replication of notifications in order to save transmissions on the bus. This is especially important when notifications are transported using multicast messages.

8.3.4 Fields

A field is a representation of a remote property, which has up to one getter, up to one setter, and up to one notifier. A field does represent a status, and thus, has a valid value at all times on which getter, setter, and notfier can act upon. The getter is a request/response call that allows read access to a field. The setter is a request/response call that allows write access to a field. Thus, the field mechanism is a combination of methods to get/set this property value and a notification for this property.

— A field shall be a combination of an optional getter, an optional setter, and an optional notification event.

— A field shall have at least one getter, or one setter, or one notification event.

— The getter of a field shall be a request/response call that has an empty payload in the request message and the value of the field in the payload of the response message.

— The setter of a field shall be a request/response call that has the desired value of the field in the payload of the request message and the value that was set to field in the payload of the response message.

— The notifier shall send an event message that transports the value of a field on change and follows the rules for events.

8.3.5 Error message format

— The error message (type 0x81) shall be used to carry specific fields for error handling, e.g. an exception string.

— The return code shall be set to the value that best describes the error condition (see Table 3).

— Error message shall copy over the fields of the SOME/IP header (i.e. Message ID, Request ID, protocol version, and interface version) but not the payload.

— Error messages are sent instead of response messages.

For more flexible error handling, SOME/IP allows the user to specify a message layout specific for errors instead of using the message layout for response messages. This is defined by the API specification and can be used to transport exceptions of higher level programming languages. The layout of the exception message shall contain at least

— a union of specific exceptions. At least one generic exception without fields shall be defined (this comes automatically with the default union type 0), and

— a dynamic-length string for exception description.

The union gives the flexibility to add new exceptions in the future in a type-safe manner.

8.3.6 Communication errors

If an application is waiting for a response to a message, but does not receive it within a certain amount of time, it can try to resend the message. The number of retries, the timeout value, and the timeout behaviour (constant or exponential back off) are outside of the SOME/IP specification and can be added to the interface definition.

Bibliography

[1] IEEE 754-2008, *IEEE Standard for Binary Floating-Point Arithmetic*

ICS 43.040.15

Price based on 41 pages