International Telecommunication Union

ITU-T Y.4500.20

(03/2018)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

Internet of things and smart cities and communities – Frameworks, architectures and protocols

oneM2M – WebSocket protocol binding

Recommendation ITU-T Y.4500.20

ITU-T Y-SERIES RECOMMENDATIONS

GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

GLOBAL INFORMATION INFRASTRUCTURE	
General	Y.100–Y.199
Services, applications and middleware	Y.200–Y.299
Network aspects	Y.300–Y.399
Interfaces and protocols	Y.400–Y.499
Numbering, addressing and naming	Y.500–Y.599
Operation, administration and maintenance	Y.600–Y.699
Security	Y.700–Y.799
Performances	Y.800–Y.899
INTERNET PROTOCOL ASPECTS	
General	Y.1000–Y.1099
Services and applications	Y.1100–Y.1199
Architecture, access, network capabilities and resource management	Y.1200–Y.1299
Transport	Y.1300–Y.1399
Interworking	Y.1400–Y.1499
Quality of service and network performance	Y.1500–Y.1599
Signalling	Y.1600–Y.1699
Operation, administration and maintenance	Y.1700–Y.1799
Charging	Y.1800–Y.1899
IPTV over NGN	Y.1900–Y.1999
NEXT GENERATION NETWORKS	
Frameworks and functional architecture models	Y.2000–Y.2099
Quality of Service and performance	Y.2100–Y.2199
Service aspects: Service capabilities and service architecture	Y.2200–Y.2249
Service aspects: Interoperability of services and networks in NGN	Y.2250–Y.2299
Enhancements to NGN	Y.2300–Y.2399
Network management	Y.2400–Y.2499
Network control architectures and protocols	Y.2500–Y.2599
Packet-based Networks	Y.2600–Y.2699
Security	Y.2700–Y.2799
Generalized mobility	Y.2800–Y.2899
Carrier grade open environment	Y.2900–Y.2999
FUTURE NETWORKS	Y.3000–Y.3499
CLOUD COMPUTING	Y.3500–Y.3999
INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES	
General	Y.4000–Y.4049
Definitions and terminologies	Y.4050–Y.4099
Requirements and use cases	Y.4100–Y.4249
Infrastructure, connectivity and networks	Y.4250–Y.4399
Frameworks, architectures and protocols	**Y.4400–Y.4549**
Services, applications, computation and data processing	Y.4550–Y.4699
Management, control and performance	Y.4700–Y.4799
Identification and security	Y.4800–Y.4899
Evaluation and assessment	Y.4900–Y.4999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Y.4500.20

oneM2M – WebSocket protocol binding

Summary

Recommendation ITU-T Y.4500.20 specifies the binding of Mca and Mcc primitives on to the WebSocket binding.

Recommendation ITU-T Y.4500.20 specifies:

- procedures and message formats for operating and closing of WebSocket connections;
- how request and response primitives are mapped into the payload of the WebSocket protocol.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Y.4500.20	2018-03-01	20	11.1002/1000/13511

Keywords

oneM2M, WebSocket.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

NOTE – This Recommendation departs slightly from the usual editorial style of ITU-T Recommendations to preserve existing cross-referencing from external documents.

Table of Contents

		Page
1	Scope	1
2	References	1
3	Definitions	1
	3.1 Terms defined elsewhere	1
	3.2 Terms defined in this Recommendation	2
4	Abbreviations and acronyms	2
5	Conventions	2
6	Overview on WebSocket Binding	3
	6.1 Use of WebSocket	3
	6.2 Binding Overview	3
7	Protocol Binding	6
	7.1 Introduction	6
	7.2 WebSocket connection establishment	7
	7.3 Closing WebSocket connection	10
	7.4 Registration procedure	10
	7.5 Handling of Non-Registration Request	10
	7.6 Use of proxy servers	10
8	Security Aspects	11
Annex A – oneM2M Specification Update and Maintenance Control Procedure		12
Appendix I – Example Procedures		13
	I.1 AE Registration and creation of a Container child resource	13
Bibliography		16

Recommendation ITU-T Y.4500.20

oneM2M – WebSocket protocol binding

1 Scope

This Recommendation specifies the binding of Mca and Mcc primitives on to the WebSocket binding.

This Recommendation specifies:

- procedures and message formats for operating and closing of WebSocket connections;
- how request and response primitives are mapped into the payload of the WebSocket protocol.

The Recommendation contains oneM2M Release 2 specification – oneM2M WebSocket Protocol Binding V2.0.0 – and is equivalent to standards of oneM2M partners including ARIB, ATIS [b-ATIS.oneM2M.TS0020V200], CCSA, ETSI [b-ETSI TS 118 120], TIA, TSDSI [b-TSDSI STD T1.oneM2M TS-0020-2.0.0], TTA [b-TTAT.MM-TS.0020 v2.0.0] and TTC [b-TTC TS-M2M-0020v2.0.0].

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Y.4500.1] Recommendation ITU-T Y.4500.1 (2018), *oneM2M Functional architecture*.

[ITU-T Y.4500.4] Recommendation ITU-T Y.4500.4 (2018), *oneM2M – Service layer core protocol*.

[ETSI TS 118 103] ETSI TS 118 103 V2.4.1 (2016), *oneM2M; Security solutions (oneM2M TS-0003 version 2.4.1 Release 2).*

[IETF RFC 6455] IETF RFC 6455 (2011), *The WebSocket protocol*.

[IETF RFC 7230] IETF RFC 7230 (2014), *Hypertext transport protocol (HTTP/1.1): Message syntax and routing*.

[IETF RFC 7692] IETF RFC 7692 (2015), *Compression extensions for WebSocket*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 application entity (AE): [b-ITU-T Y.4500.11] Represents an instantiation of application logic for end-to-end M2M solutions.

3.1.2 common services entity (CSE): [b-ITU-T Y.4500.11] Represents an instantiation of a set of common service functions of the M2M environments. Such service functions are exposed to other entities through reference points.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 oneM2M WebSocket client (WS client): WebSocket client associated with an application entity or a common services entity capable of establishing the WebSocket connections.

3.2.2 oneM2M WebSocket server (WS server): WebSocket server associated with a common services entity which accepts requests to establish WebSocket connections.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

ADN	Application Dedicated Node
AE	Application Entity
ASN	Application Service Node
CBOR	Concise Binary Object Representation
CMDH	Communication Management and Delivery Handling
CRUDN	Create-Retrieve-Update-Delete-Notify
CSE	Common Services Entity
FQDN	Fully Qualified Domain Name
GUID	Globally Unique Identifier
HTTP	Hypertext Transport Protocol
IN-CSE	Infrastructure Node-Common Services Entity
IP	Internet Protocol
JSON	JavaScript Object Notation
MN-CSE	Middle Node-Common Services Entity
NAT	Network Address Translator
SAEF	Security Association Establishment Framework
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
WS	WebSocket
WSS	WebSocket Secure
XML	Extensible Markup Language

5 Conventions

The keywords "shall", "shall not", "may", "need not", "should", "should not" in this Recommendation are to be interpreted as follows.

Shall/shall not:

Requirements:

1) effect on this Recommendation: this Recommendation needs to describe the required feature (i.e., specify a technical solution for the requirement);

2) effect on products: every implementation (M2M solution that complies with this Recommendation) must support it;

3) effect on deployments: every deployment (M2M service based on this Recommendation) must use the standardized feature where applicable – otherwise interoperability problems with other services could arise, for example.

Should/should not:

Recommendation

1.) effect on this Recommendation: this Recommendation needs to describe a solution that allows the presence and the absence of the feature;

2) effect on products: an implementation may or may not support it; however, support is recommended;

3) effect on deployments: a deployment may or may not use it; however, usage is recommended.

May/need not:

Permission/option

1) effect on this Recommendation: this Recommendation needs to describe a solution that allows the presence and the absence of the required feature;

2) effect on products: an implementation may or may not support it;

3) effect on deployments: a deployment may or may not use it.

6 Overview of WebSocket binding

6.1 Use of WebSocket

This binding makes use of the WebSocket protocol [IETF RFC 6455] to transport serialized representations of oneM2M request and response primitives over the Mca or Mcc reference points.

Establishment of a WebSocket connection shall be initiated by a WebSocket client by sending a handshake to a WebSocket server as specified in section 4 of [IETF RFC 6455]. Once the WebSocket connection is established, both the oneM2M request and response primitives can be exchanged bi-directionally between the two endpoints of the connection. Serialized representations of the request and response primitives shall be mapped in the Payload Data field of the WebSocket base framing protocol, as defined in section 5.2 of [IETF RFC 6455].

A WebSocket connection employs either a transmission control protocol/Internet protocol (TCP/IP) or transport layer security (TLS) over a TCP/IP connection. The underlying TCP and TLS connections are established as the first step of the WebSocket handshake.

6.2 Binding overview

WebSocket binding may be employed for communication between any two endpoints that can be connected over the Mca, Mcc or Mcc' interface reference points supported by the oneM2M architecture as shown in Figure 6.1-1 of oneM2M TS-0001 [ITU-T Y.4500.1].

When using the WebSocket protocol, one communication endpoint shall act as the WebSocket server. The WebSocket server listens for inbound handshake messages arriving from any WebSocket client to which a WebSocket connection is not yet established. Whether a communication endpoint takes the role of the client or the server shall depend on the registration relationship between the communicating entities as follows: the registree shall always use a WebSocket client, while the associated registrar shall always use a WebSocket server on the respective reference point.

This implies that the application dedicated node (ADN) and application service node (ASN) always take the role of a WebSocket client when WebSocket binding is employed. A middle node-common

services entity (MN-CSE) uses a WebSocket server to communicate with its registrees and a WebSocket client to communicate with its own registrar [which can be another MN-CSE or an infrastructure node-common services entity (IN-CSE)].

The IN-CSE provides a WebSocket server functionality to communicate with all its registrees, i.e., within a service provider's domain. On the Mcc' reference points, i.e., for communication between IN-CSEs of different Service Provider domains, the IN-CSE shall provide both WebSocket client and server functionality. This enables any IN-CSE to open a WebSocket connection to any IN-CSE of another Service Provider's domain.

Figure 6.2.-1 shows an example of an applicable system configuration.

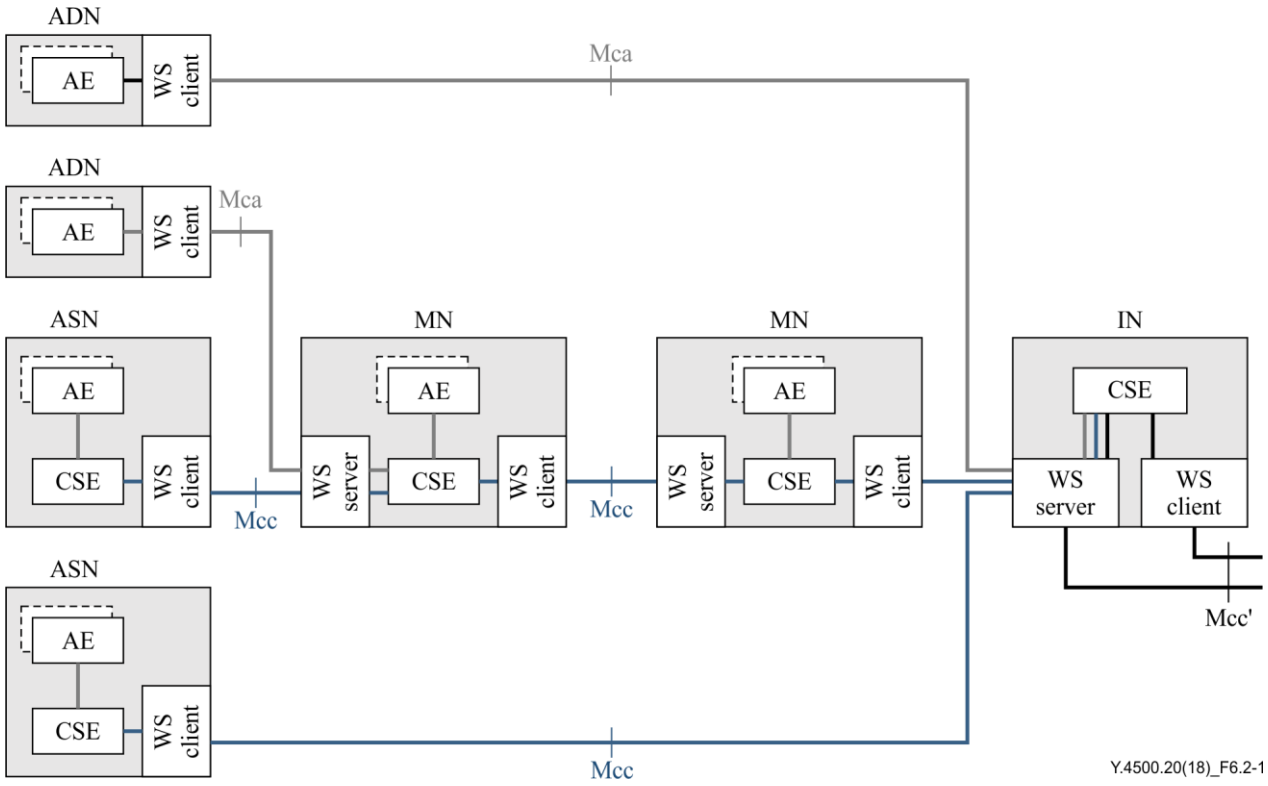

Figure 6.2-1 – Example scenarios of WebSocket client and server configurations

There exists a maximum of one WebSocket connection between two nodes. A WebSocket connection is established for the first time when the initial registration procedure of an entity to its registrar is performed. On an established WebSocket connection, request and response primitives can be exchanged in both directions. Any connection may be closed by either the WebSocket client or the server, depending on the communication schedule of either entity. However, the connection can be reopened from the client side only.

If the connection is closed temporarily, it shall be reopened when the next request primitive is sent from the client to the server side, or when the time to become reachable configured at <schedule> resource. If the WebSocket connection with the next-hop entity is not opened, and the WebSocket connection cannot be established due to lack of *pointOfAccess* address for the entity, a sending CSE may enable buffering of primitives that should be sent to the entity until the connection is reopened or their expiry time is reached. See Annex H of oneM2M TS-0004 [ITU-T Y. 4500.4] about buffering of primitives by communication management and delivery handling (CMDH) functionality.

Figure 6.2-2 shows an example of a message flow for a scenario where an ADN-AE registers on its registrar MN-CSE using an unsecured TCP connection without proxy and then continues exchanging non-registration request and response primitives.

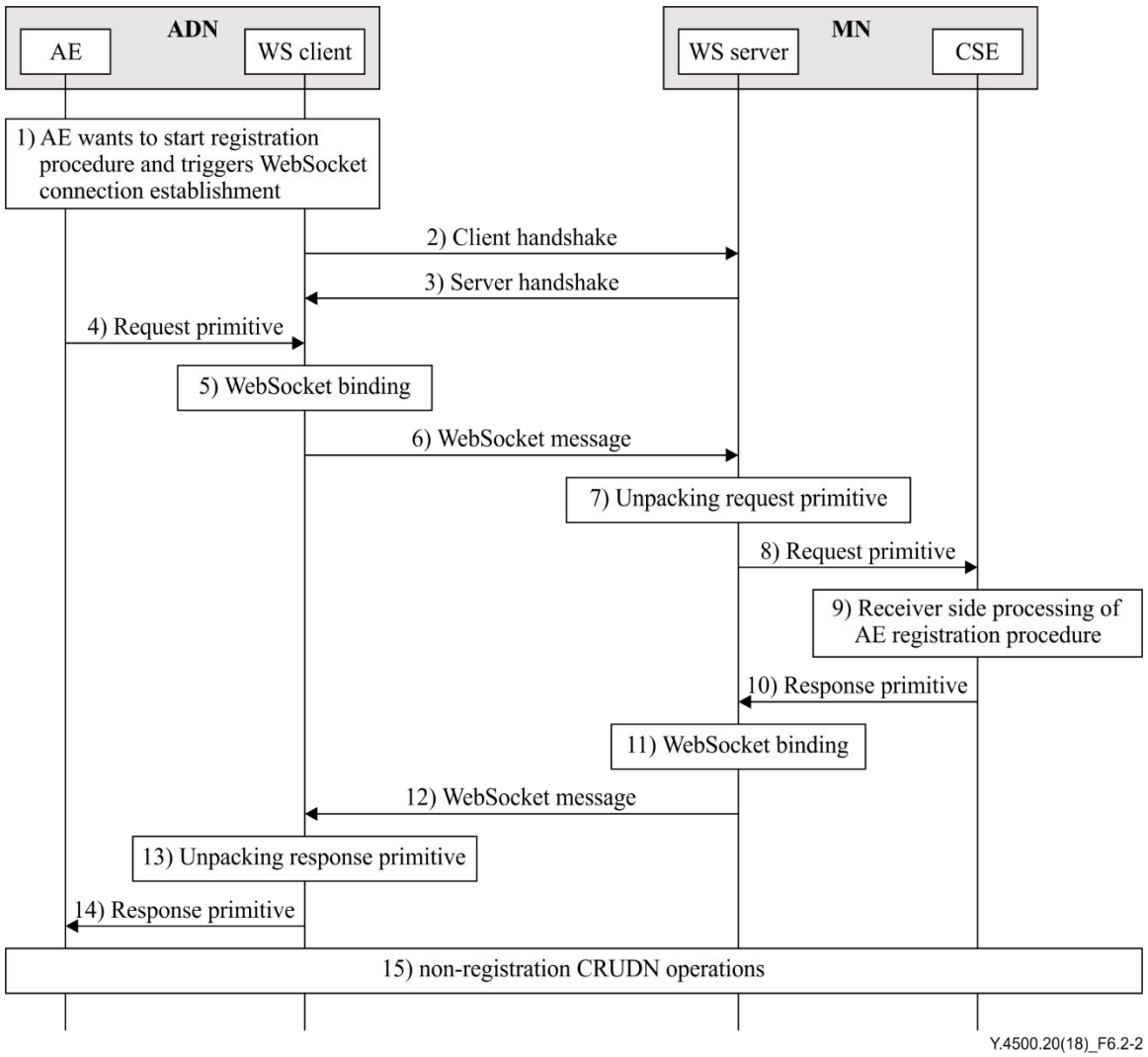

Figure 6.2-2 – Example of a message flow with WebSocket binding

1) The ADN-AE wants to register on its registrar MN-CSE. If a WebSocket connection does not exist, it is established by the steps 2) and 3). It is assumed that the ADN-AE knows the point of access (i.e., a WebSocket uniform resource identifier (URI) specified in [IETF RFC 6455]) under which the registrar CSE can be reached with a WebSocket binding.

2) The WebSocket client opens a handshake to the server with the subprotocol name "oneM2M-pro-v1.0" following [IETF RFC 6455].

   If the server can be reached under the WebSocket URI ws://example.net:9000/, the client handshake may look as follows:

```
GET / HTTP/1.1
Host: mncse1234.net:9000
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==
Sec-WebSocket-Protocol: oneM2M-pro-v1.0
Sec-WebSocket-Version: 13
```

3) The WebSocket server replies with a handshake to the client. In the successful case, the status-line of this hypertext transport protocol (HTTP) response may look as follows:

```
Request-Version:  HTTP/1.1
Status-Code: 101
Response-Phrase: Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Protocol: oneM2M-pro-v1.0
Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=
```

4) The ADN-AE issues a registration request primitive. The request primitive may look, for example, like a representation serialized by JavaScript object notation (JSON) as follows:

```
{"op":1,"to":"//example.net/mncse1234","rqi":"A1234","pc":{"m2m:ae":{"api":"a56",
"apn":"app1234"}},"ty": 2}
```

NOTE – The WebSocket client associated with an ADN-AE does not need to be reachable for WebSocket Server handshake messages.

5) The WebSocket Binding process, which transforms a single oneM2M primitive into one or more data frames of the WebSocket Framing protocol, is specified in [IETF RFC 6455]. When transmitting a JSON-serialized primitive in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment will be set to x1 ("text frame").

6) The WebSocket message (consisting of one or more frames) shall be sent to the WebSocket (WS) server.

7) The original request primitive shall be unpacked from the WebSocket message by the WS server.

8) The request primitive is delivered to the MN-CSE.

9) The MN-CSE performs the receiver side operations of AE registration as specified in oneM2M TS-0001 [ITU-T Y.4500.1].

10) The response primitive is issued to the WebSocket server.

11) The WebSocket binding process for the response primitive is performed.

12) The WebSocket message (consisting of one or more frames) is sent to the client.

13) The response primitive is unpacked.

14) The response primitive is to the ADN-AE.

15) After successful completion of AE registration, any other create-retrieve-update-delete-notify (CRUDN) requests and response primitives can be exchanged over the existing WebSocket connection in both directions. If the ADN-AE has no other requests to send, the WebSocket connection may be closed temporarily. When the WebSocket connection is closed after registration and later reopened, the registration procedure as outlined in steps 4 to 14 is omitted. In this case, any non-registration request primitives can be sent directly.

7 Protocol binding

7.1 Introduction

The WebSocket protocol enables two-way communication between client and server even when a firewall or a network address translator (NAT) is present between them. This means, once a WebSocket connection is established, request (and response) primitives can be exchanged in both directions, from the client to the server and vice versa. However, AEs may be capable of handling notification request primitives only or no request primitives at all.

WebSocket binding applied by oneM2M entities or nodes shall be fully compliant with [IETF RFC 6455]. After establishment of a WebSocket connection between two nodes, at the transmitter side, each individual request and response primitive is mapped into one or several WebSocket frames. See Appendix I for examples of procedures.

7.2 WebSocket connection establishment

7.2.1 General

A WebSocket connection is opened by the client side as specified in section 4 of [IETF RFC 6455] with the sending of a client handshake. The server responds with a server handshake.

The client handshake consists of an HTTP upgrade request, along with a list of required and optional header fields.

The handshake shall be a valid HTTP request as specified by [IETF RFC 7230]. The server handshake consists of an HTTP status-line and a list of header fields.

The applicable format of the request-line, status-line and the applicable header fields are specified in clauses 7.2.2 and 7.2.3.

HTTP headers fields have case-insensitive field names.

CSEs capable of supporting WebSocket shall indicate the schemes WebSocket (WS) or WebSocket secure (WSS) together with the applicable host name and port numbers in the pointOfAccess attribute of their <CSEBase> resource, i.e., as `ws://host:port1` and `wss://host:port2`.

7.2.2 Client handshake

7.2.2.1 Format of request-line

The request-line of a client handshake shall begin with the method token "`GET`", followed by the request target "/" and the HTTP version set to "HTTP/1.1" as follows:

```
GET / HTTP/1.1
```

If the client is configured to use a proxy when using the WebSocket Protocol, a connection to the proxy server shall be established prior to sending the client handshake in the foregoing. This is described in clause 7.6.

7.2.2.2 Host header

The host header shall be present in each client handshake.

The host header indicates the fully qualified domain name (FQDN) or IP address of the receiver CSE of the next hop. If the originator of the client handshake is an oneM2M field entity, the host header represents the registrar CSE of the originator.

When no proxy is used, the host header shall be set as one of the pointOfAccess attribute values associated with the receiver. Selection of the appropriate receiver is described in oneM2M TS-0004 [ITU-T Y.4500.4].

If the client is configured to use a proxy when using the WebSocket protocol, then the client should connect to that proxy and ask it to open a TCP connection to the host and port rather than to the next hop CSE.

7.2.2.3 Upgrade header

The upgrade header shall be present in each client handshake message with value WebSocket as follows:

```
Upgrade: WebSocket
```

7.2.2.4 Connection header

The connection header shall be present in each client handshake message with value upgrade as follows:

```
Connection: Upgrade
```

7.2.2.5 Sec-WebSocket-Key header

The Sec-WebSocket-Key header shall be present in each client handshake message. The header field includes a base64-encoded representation of a random 16 byte pattern, e.g.:

```
Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==
```

7.2.2.6 Sec-WebSocket-Version header

The Sec-WebSocket-Version header shall be present in each client handshake message with value 13 as follows:

```
Sec-WebSocket-Version: 13
```

7.2.2.7 Sec-WebSocket-Protocol header

The Sec-WebSocket-Protocol header shall be present in a client handshake message. It enables the client to indicate its supported application subprotocols on the server and to be sure that the server agrees to support that subprotocol. It is used by the client to indicate the oneM2M service layer protocol version and supported serialization formats to the server.

The value of the Sec-WebSocket-Protocol header shall be one or more of the registered names defined in clause 7.2.2.9. It shall also be allowed to include multiple Sec-WebSocket-Protocol headers with a value that includes one registered name each as defined in [IETF RFC 6455], e.g.:

```
Sec-WebSocket-Protocol:   oneM2M.R2.0.JSON, oneM2M-R2.0_XML
```

and

```
Sec-WebSocket-Protocol:   oneM2M.R2.0.XML

Sec-WebSocket-Protocol:   oneM2M.R2.0.JSON
```

are equivalent headers, expressing that the WebSocket client supports both application subprotocols, oneM2M.R2.0.XML and oneM2M.R2.0.JSON. The order of names indicated in the Sec-WebSocket-Protocol header specifies the client's preference.

7.2.2.8 Sec-WebSocket-Extensions header

The Sec-WebSocket-Extensions header may be used to negotiate the use of per-message compression as specified in [IETF RFC 7692].

If the client handshake includes the header, e.g.

```
Sec-WebSocket-Extensions: permessage-deflate
```

it indicates the client's preference to apply the compression mechanism defined in [IETF RFC 7692] to the server. The header may include additional parameters as specified in [IETF RFC 7692].

When the server accepts use of message compression, it responds with a Sec-WebSocket-Extensions header in the server handshake message as specified in clause 7.2.3.6, and in this case, compression is applied in both transmission directions. If the server handshake message does not include a Sec-WebSocket-Extensions header, compression shall not be applied.

7.2.2.9 Subprotocol names and serialization formats

The Sec-WebSocket-Protocol header in the opening handshake is used to negotiate the application protocol layered on top of WebSocket. The application protocol addressed in this specification is the Release-2 version of the oneM2M service layer.

The oneM2M service layer protocol consists of the exchange of serialized representations of request and response primitives as defined in oneM2M TS-0001 [ITU-T Y.4500.1] and oneM2M TS-0004 [ITU-T Y.4500.4]. This version of the specification allows use of the serialization formats listed in Table 7.2.2.9-1. Both protocol version and serialization format are associated with a specific subprotocol name.

Table 7.2.2.9-1 lists the serialization formats associated subprotocols names and opcode settings of the WebSocket Frame protocol applicable for this specification.

Table 7.2.2.9-1 – Applicable subprotocol names

Serialization Format	Subprotocol Name	WS opcode	Notes
JSON	oneM2M.R2.0.json	x1 ("text frame")	See clause 8.4 in oneM2M TS-0004 [ITU-T Y.4500.4]
Extensible markup language (XML)	oneM2M.R2.0.xml	x1 ("text frame")	See clause 8.3 in oneM2M TS-0004 [ITU-T Y.4500.4]
Concise binary object representation (CBOR)	oneM2M.R2.0.cbor	x2 ("binary frame")	See clause 8.5 in oneM2M TS-0004 [ITU-T Y.4500.4]

7.2.3 Server handshake format

7.2.3.1 Format of status-line

The status-line of a server handshake shall begin with the HTTP version set to "`HTTP/1.1`", followed by the status code and reason phrase as defined in [IETF RFC 6455]. When the WebSocket connection is established successfully, the status-line may look as follows:

```
HTTP/1.1 101 Switching Protocols
```

When connection establishment is unsuccessful, any appropriate HTTP error status code shall be returned with the optional addition of a corresponding reason phrase.

7.2.3.2 Upgrade header

The Upgrade header shall be present in each server handshake message with value WebSocket as follows:

```
Upgrade: WebSocket
```

7.2.3.3 Connection header

The Connection header shall be present in each server handshake message with value Upgrade as follows:

```
Connection: Upgrade
```

7.2.3.4 Sec-WebSocket-Accept header

The Sec-WebSocket-Accept header shall be present in each server handshake message. The header field shall be constructed from the Sec-WebSocket-Key value and the globally unique identifier (GUID) as specified in section 4.2.2 of [IETF RFC 6455]. It may look, for example, as follows:

```
Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=
```

7.2.3.5 Sec-WebSocket-Protocol header

The Sec-WebSocket-Protocol header shall be present in a server handshake message. It indicates to the client that the server accepts (one of) the subprotocol(s) indicated by the client.

The server compliant with this specification shall select one of the subprotocol names indicated in the Sec-WebSocket-Protocol header of the client handshake message and set the value of the Sec-WebSocket-Protocol header of the server handshake message accordingly.

7.2.3.6 Sec-WebSocket-Extensions header

If the optional Sec-WebSocket-Extensions header with value "`permessage-deflate`" was included in the client handshake message, the Sec-WebSocket-Extensions header with same value shall also be included in the server handshake message, if the server accepts usage of message compression, and apply message compression in the transmit direction and message decompression in the receive direction as defined in [IETF RFC 7692].

If the server does not accept message compression, it shall not include the `Sec-WebSocket-Extensions` header.

7.3 Closing WebSocket connection

Compliant with section 7 of [IETF RFC 6455] a WebSocket connection shall be closed by sending a connection close frame (opcode x8). Both, client and server may initiate a closing handshake of an existing WebSocket connection at any time.

WebSocket connections should be kept open for as long as possible considering any constraints due to communication policies and power-saving requirements. Unless communication policies enforce the closing of network access, it is left to implementation to decide when exactly the closing of a WebSocket shall be triggered.

7.4 Registration procedure

A oneM2M entity (AE or CSE) not yet registered on its registrar CSE needs to be preconfigured with various parameters as specified in oneM2M TS-0001 [ITU-T Y.4500.1] and oneM2M TS-0003 [ETSI TS 118 103] in order to be able to send the registration request primitive (i.e., create <*AE*> or create <*remoteCSE*> request primitive). To establish a WebSocket connection, the WebSocket client shall be configured with an applicable point of access of its registrar CSE, which includes the FQDN or IP address and the port number.

After the registration procedure has been successfully completed, the WebSocket Server (e.g., Registrar CSE for WebSocket Client) shall enable routing of any incoming oneM2M primitives to this registree.

Thus until the before the registration procedure is successfully completed, any incoming oneM2M primitives to the WebClient shall be rejected by the receiver (e.g., registrar CSE).

Closing of the WebSocket connection after registration does not impact the registration status of an AE or CSE on its registrar, unless an explicit de-registration procedure is performed by deletion of the respective <*AE*> or <*remoteCSE*> resource instance.

7.5 Handling of a non-registration request

Registered entities (AE and CSE) are allowed to send and receive non-registration request primitives. A WebSocket connection should support any of the transfer modes defined in clause 8.2 of oneM2M TS-0001 [ITU-T Y.4500.1], i.e., blocking requests, and non-blocking requests for both synchronous and asynchronous cases.

When sending blocking requests, the WebSocket connection shall not be closed before the response is received or before any configured timeout period has expired.

When sending non-blocking requests, the WebSocket connection shall not be closed before the acknowledgment response is received or before any configured timeout period has expired. If the entities' communication policies and power-saving requirements allow, the connection should be kept open at least until an ongoing procedure has fully completed, i.e., requesting of the result in synchronous mode or completion of a notify procedure in asynchronous mode.

7.6 Use of proxy servers

The connection to a proxy shall be requested by sending a request-line with the method token "`CONNECT`", followed by the request target host and port of the WebSocket server and the HTTP version set to "`HTTP/1.1`" as follows:

```
CONNECT WSserver.example.com:80 HTTP/1.1
```

8 Security aspects

Authentication and TLS can be established when the oneM2M entity that hosts the WebSocket Server can be addressed with the WSS URI scheme. When using the WSS URI scheme, one of the security association establishment frameworks (SAEFs) as defined in oneM2M TS-0003 [ETSI TS 118 103] shall be applied to provide mutually authenticated TLS between the communicating entities prior to sending the WebSocket client handshake.

The SAEF is accomplished by successful completion of a TLS handshake procedure before the client sends its opening handshake message. The details of SAEFs and any remote security provisioning frameworks required are specified in oneM2M TS-0003 [ETSI TS 118 103].

In special deployment scenarios, e.g., when the communicating oneM2M entities using WebSocket binding are located in a secure environment or implemented on the same device, TLS may not be required. In such scenarios, unsecured WebSocket communication addressed by the WS URI scheme may be adequate.

Annex A

oneM2M specification update and maintenance control procedure

(This annex forms an integral part of this Recommendation.)

The provisions of Annex L of [ITU-T Y.4500.1] entitled "oneM2M specification update and maintenance control procedure" shall apply to this Recommendation.

Appendix I

Example procedures

(This appendix does not form an integral part of this Recommendation.)

I.1 AE registration and creation of a container child resource

Figure I.1-1 illustrates a message flow for registration of an ADN-AE to an IN-CSE as described in clause 7.3.5.2.1 of oneM2M TS-0004 [ITU-T Y.4500.4] with WebSocket mapping and subsequent creation of a <Container> child resource.

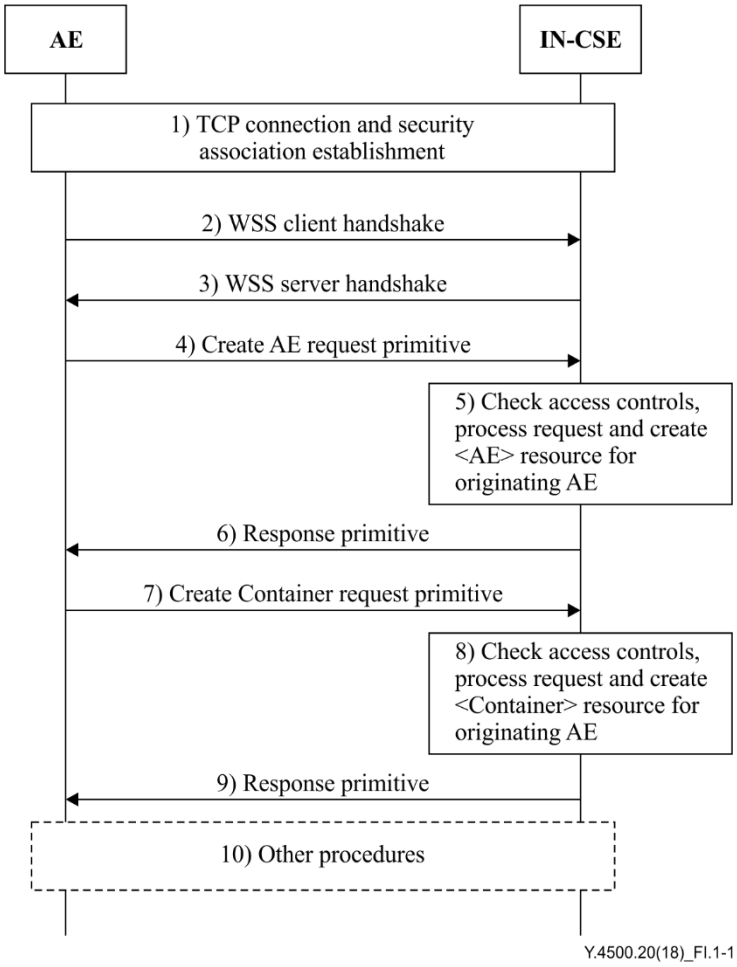

Figure I.1-1 – Message flow for registration of an ADN-AE to an IN-CSE

In the example considered, the WebSocket protocol is used to send a JSON-serialized request and response primitives in text format.

The message flow may look as follows.

1) TCP connection establishment and security association establishment as defined in oneM2M TS-0003 [ETSI TS 118 103] based on a TLS handshake procedure is accomplished.

2) The WSS client sends, for example, the following opening handshake message, offering to use either JSON or XML serialization of primitives:

```
GET / HTTP/1.1

Host: mncse1234.net:9000

Upgrade: WebSocket
```

```
Connection: Upgrade

Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==

Sec-WebSocket-Protocol: oneM2M.R2.0.json, oneM2M.R2.0.xml

Sec-WebSocket-Version: 13
```

3) The WSS server selects the use of JSON serialization and responds with the following handshake message:

```
Request-Version:  HTTP/1.1

Status-Code: 101

Response-Phrase: Switching Protocols

Upgrade: WebSocket

Connection: Upgrade

Sec-WebSocket-Protocol: oneM2M.R2.0.json

Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=
```

4) The AE sends the following request primitive in textual JSON-serialized format:

```
{"op":1,"to":"//example.net/mncse1234","rqi":"A1000",
"rcn":7,"pc":{"m2m:ae":{"rn":"SmartHomeApplication", "api":"Na56", "apn":"app1234"}},"ty":2}
```

The above JSON object is mapped by the WS client into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 ("text frame").

5) The IN-CSE validates the privilege of the originator to create an <AE> resource, and accepts the request to create the resource.

6) The IN-CSE acknowledges the success of the request by responding the following JSON-serialized response primitive. The response primitive includes all attributes of <AE> instance created in Step 5.

```
{"rsc":2001,"rqi":"A1000","pc":{"m2m:ae":{"rn":"SmartHomeApplication","ty":2,"ri":"ae1","api"
:"Na56","apn":"app1234","pi":"cb1","ct":"20160506T153208",
"lt":"20160506T153208","acpi":["acp1","acp2"],"et":"20180506T153208", "aei":"S_SAH25"}}}
```

The above JSON object is mapped by the WS server into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 ("text frame").

7) The AE sends the following request primitive in textual JSON-serialized format to create a <Container> resource as a child resource of the <AE> created in Step 5:

```
{"op":1,"to":"//example.net/mncse1234/SmartHomeApplication",
"rqi":"A1001","rcn":7,"pc":{"m2m:cnt":{"rn":"SmartHomeContainer","mbs":100000,
"mni":500}},"ty":3}
```

The above JSON object is mapped by the WS client into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 ("text frame").

8) The IN-CSE validates the privilege of the originator to create an <Container> resource under the <AE> resource created in step 5, and accepts the request to create the resource.

9) The IN-CSE acknowledges the success of the request by responding the following JSON-serialized response primitive

```
{"rsc":2001,"rqi":"A1001","pc":{"m2m:cnt":{"rn":"SmartHomeContainer",
"ty":3,"ri":"cnt1","pi":"ae1","ct":"20160506T154048",
"lt":"20160506T154048","acpi":["acp1"],"et":"20180506T154048","cr":"
S_SAH25","st":0,"mni":500,"mbs":100000,"cni":0,"cbs":0,"mia":3600}}}
```

The above JSON object is mapped by the WS server into a data frame of the WebSocket framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket base framing protocol of the first message fragment is set to x1 ("text frame").

10) Primitives of further subsequent CRUDN procedures may be transferred to the existing WebSocket connection.

Bibliography

[b-ITU-T Y.4500.11] Recommendation ITU-T Y.4500.11 (2018), *oneM2M – Common terminology*.

[b-ATIS.oneM2M.TS0020V200] ATIS.oneM2M.TS0020V200-2016, *WebSocket protocol binding*.

[b-ETSI TS 118 120] ETSI TS 118 120 (2016), *oneM2M; WebSocket Protocol Binding – (oneM2M TS-0020 version 2.0.0 Release 2).*

[b-TSDSI STD T1.oneM2M TS-0020-2.0.0] TSDSI STD T1.oneM2M TS-0020-2.0.0 V1.0.0 (2017), *WebSocket protocol binding*.

[b-TTAT.MM-TS.0020 v2.0.0] TTAT.MM-TS.0020 v2.0.0 (2017), *oneM2M - WebSocket protocol binding v2.0.0.*

[b-TTC TS-M2M-0020v2.0.0] TTC TS-M2M-0020v2.0.0 (2016), *oneM2M technical specification –- WebSocket protocol binding.*

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	**Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities**
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2018