ETSI TS 103 596-3 V1.1.1 (2021-05)

Methods for Testing and Specification (MTS);
Test Specification for CoAP;
Part 3: Performance Tests

Reference
DTS/MTS-TSTCoAP-3

Keywords
performance, testing

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document is part 3 of a multi-part deliverable covering the Constrained Application Protocol (CoAP), as identified below:

Part 1: "Conformance Tests";

Part 2: "Security Tests";

Part 3: "Performance Tests".

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The present document provides an introduction and possible test specification, i.e. an overall test suite structure and catalogue of performance test purposes for the Constrained Application Protocol (CoAP) protocol. It will be a reference base for both client side test campaigns and server side test campaigns addressing the performance issues.

1 Scope

The present document provides a test specification, i.e. an overall test suite structure and catalogue of test purposes for the Constrained Application Protocol (CoAP) protocol. It will be a reference base for both client side test campaigns and server side test campaigns addressing the performance issues.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] IETF RFC 7252: "The Constrained Application Protocol (CoAP)".

[2] ETSI TS 103 596-1: "Methods for Testing and Specification (MTS); Test Specification for CoAP; Part 1: Conformance Tests".

[3] IETF RFC 8323: "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets".

[4] ETSI ES 203 119-4: "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension)".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IETF RFC 2544: "Benchmarking Methodology for Network Interconnect Devices".

[i.2] ETSI TR 101 577: "Methods for Testing and Specifications (MTS); Performance Testing of Distributed Systems; Concepts and Terminology".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

acknowledgement message: message which acknowledges that a specific Confirmable message arrived as defined in IETF RFC 7252 [1]

NOTE: By itself, an Acknowledgement message does not indicate success or failure of any request encapsulated in the Confirmable message, but the Acknowledgement message may also carry a Piggybacked Response.

benchmark test: procedure by which a test system interacts with a System Under Test to measure its behaviour and produce a benchmark report

benchmark test report: document generated at the conclusion of a test procedure containing the metrics measured during the test

client: originating endpoint of a request; the destination endpoint of a response as defined in IETF RFC 7252 [1]

CoAP-to-CoAP proxy: proxy that maps from a CoAP request to a CoAP request, i.e. uses the CoAP protocol both on the server and the client side

NOTE: Contrast to cross-proxy.

confirmable message: some messages that require an acknowledgement as defined in IETF RFC 7252 [1]

NOTE: These messages are called "Confirmable". When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgement or type Reset.

conformance: extent to which an implementation of a standard satisfies the requirements expressed in that standard

conformance testing: process to verify to what extent the IUT conforms to the standard

content-format: combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the "CoAP Content-Formats" registry as defined in IETF RFC 7252 [1]

NOTE: When the focus is less on the numeric identifier than on the combination of these characteristics of a resource representation, this is also called "representation format".

critical option: option that would need to be understood by the endpoint ultimately receiving the message in order to properly process the message as defined in IETF RFC 7252 [1]

NOTE: The implementation of critical options is, as the name "Option" implies, generally optional: unsupported critical options lead to an error response or summary rejection of the message.

cross-proxy: cross-protocol proxy, or "cross-proxy" for short, proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy

NOTE: While the present document makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

Design Objective Capacity (DOC): largest load an SUT can sustain while not exceeding design objectives defined for a use-case

elective option: option that is intended to be ignored by an endpoint that does not understand it as defined in IETF RFC 7252 [1]

NOTE: Processing the message even without understanding the option is acceptable.

empty message: message with a Code of 0.00; neither a request nor a response as defined in IETF RFC 7252 [1]

NOTE: An Empty message only contains the 4-byte header.

endpoint: entity participating in the CoAP protocol as defined in IETF RFC 7252 [1]

NOTE: Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport-layer multiplexing information that can include a UDP port number and a security association.

forward-proxy: endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations as defined in IETF RFC 7252 [1]

NOTE: Some translations are minimal, such as for proxy requests for "CoAP" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

intermediary: CoAP endpoint that acts both as a server and as a client towards an origin server (possibly via further intermediaries) as defined in IETF RFC 7252 [1]

NOTE: A common form of an intermediary is a proxy; several classes of such proxies are discussed in the present document.

non-confirmable message: As defined in IETF RFC 7252 [1], some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor.

origin server: server on which a given resource resides or is to be created as defined in IETF RFC 7252 [1]

parameter: attribute of a SUT, test system, system load, or traffic set whose value is set externally and prior to a benchmark test, and whose value affects the behaviour of the benchmark test

piggybacked response: included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response as defined in IETF RFC 7252 [1]

proxy: intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process as defined in IETF RFC 7252 [1]

NOTE: As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behaviour based on the nature of each request.

recipient: destination endpoint of a message as defined in IETF RFC 7252 [1]

NOTE: When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

reset message: a specific message (Confirmable or Non-confirmable) is received, but some context is missing to properly process it as defined in IETF RFC 7252 [1]

NOTE: This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g. by sending an Empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

resource discovery: process where a CoAP client queries a server for its list of hosted resources as defined in IETF RFC 7252 [1]

reverse-proxy: endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations as defined in IETF RFC 7252 [1]

NOTE: Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it were the origin server for the target resource.

safe-to-forward option: option that is intended to be safe for forwarding by a proxy that does not understand it as defined in IETF RFC 7252 [1]

NOTE: Forwarding the message even without understanding the option is acceptable.

sender: originating endpoint of a message as defined in IETF RFC 7252 [1]

NOTE: When the aspect of identification of the specific sender is in focus, also "source endpoint".

separate response: when a Confirmable message carrying a request is acknowledged with an Empty message (e.g. because the server does not have the answer right away), a Separate Response is sent in a separate message exchange as defined in IETF RFC 7252 [1]

server: destination endpoint of a request; the originating endpoint of a response as defined in IETF RFC 7252 [1]

test scenario: specific path through a use-case, whose implementation by a test system creates a system load

test suite structure: document defining (hierarchical) grouping of test cases according to some rules

traffic-time profile: evolution of the average scenario over a time interval

unsafe option: option that would need to be understood by a proxy receiving the message in order to safely forward the message as defined in IETF RFC 7252 [1]

NOTE: Not every critical option is an unsafe option.

use case: description of a goal that a user has in interacting with a system, the various actors and the SUT

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DOC	Design Objective Capacity
GTW	Gateway
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IUT	Implementation Under Test
KPI	Key Performance Indicator
LAN	Local Area Network
MTS	Methods for Testing and Specifications
NoS	Number of Subscribers
NoC	Number of Clients
OS	Operating System
PICS	Protocol Implementation Conformance Statement
PING	Packet Internet Groper

NOTE: Send a packet to a computer and wait for its return.

PONG	Ping response packet
RAM	Random Access Memory
SSD	Solid State Drive
SoC	System on a Chip
SUT	System Under Test
TCP	Tranmission Control Protol
TDL	Test Description Language
TDL-TO	Test Description Language - Test Objectives
TP	Test Purpose
TS	Test System
UDP	User Datagram Protocol
URI	Unified Resource Identifier
WLF	WorkLoad Factor

4 Performance metrics

4.0 Introduction

The performance metrics specified herein pertain to the specifics of a CoAP IUT. As such, the objective is to use these metrics in order to determine how well the CoAP component (be it client or server) is performing its functions. As CoAP is a transport protocol, the metrics will be focused on how fast, reliable and efficient the transport is handled. The metrics are designed to fit this purpose while covering multiple use-case scenarios. Following below are the specific messages of the CoAP protocol as defined in IETF RFC 8323 [3] for which the performance metrics are defined.

4.1 Concepts

4.1.1 Concepts introduction

For measuring performance of a given Test System (TS), a comprehensive description of the test environment is required. This includes but it is not limited to:

- TS hardware infrastructure: resource specification, type, capacity and distribution.
- Test environment type and resources (virtualisation technology, allocated resources).
- Measurement equipment hardware/software infrastructure, measurement probe distribution/placement, clock synchronization precision, allocated resources.
- Communication infrastructure: transport network specification, number of switches/hops between TS components, bandwidth capacity.

Additional to the specific characteristics of the SUT, the CoAP protocol [3] specifies sessions as stateful interactions between clients and servers. Because of this, additional performance session-based metrics are considered.

4.1.2 Measurement Preliminary Considerations

In order for the collected measurement data to be useful, special consideration needs to be given to the TS setup. Given that the performance evaluation is targeting one or several IUTs same TS setup characteristics are required in order for the evaluation results to allow valid comparisons between them. Some of the characteristics may refer to infrastructure, hardware, physical or virtual resources as well as network connectivity resources.

4.2 Measurement Methodology

4.2.0 Introduction

This clause presents the test methodology for CoAP performance evaluation. From the performance perspective, all measurable metrics related to the protocol should be considered. Although not exhaustive, these metrics can be categorized as follows:

Powerfulness metrics as defined in ETSI TR 101 577 [i.2] include 3 sub categories: Responsiveness, Capacity and Scalability. From the Responsiveness category the response time, roundtrip time and latency time metrics are used. From the Capacity category the arrival capacity, peak capacity, in progress capacity, streaming capacity and Throughput capacity metrics are used. From the Scalability category the scaling capacity metric is used.

Reliability metrics as defined in ETSI TR 101 577 [i.2] include 6 sub categories: Quality-of-Service, Stability, Availability, Robustness, Recovery, and Correctness. The Quality of Service sub category refers to well defined requirements which may include acceptable values or ranges for metrics from other categories. Stability refers to the capacity of the System to deliver acceptable performance over time. From the Availability sub category, the logical availability metric is used. From the Robustness sub category, the service capacity reduction and service responsiveness deterioration metrics are used. From the Recovery sub category, the service restart characteristics metric is used. Correctness metrics cover the ability of delivering correctly processed requests under high or odd load conditions.

Efficiency metrics as defined in ETSI TR 101 577 [i.2] cover resource utilization. The metrics cover the characteristics of resource usage, linearity, scalability and bottleneck. The efficiency metrics used in the present document are referring to the service level and not covering the platform level.

4.2.1 Metric Post-processing

The collection of metric values from a SUT is performed by multiple agents and/or directly by the IUT. Often a better insight into the IUT performance is gained by post-processing these values in order to get more meaningful results. To this scope, the data samples can be aggregates over time intervals in the experiment. From such common practices, the following are used for the metrics listed in this clause:

- Mean Average: $\frac{1}{n}\sum_1^n x_i$, where n is the number of samples and x is a sample value.
- Standard deviation: $\sqrt{\frac{1}{n}\sum_1^n(x_i - \bar{x_i})^2}$, where n is the number of samples, x is a sample value and $\bar{x}$ is the mean average.
- Minimum: $\min(x_i)$, the smallest sample value, relative to the rest of the samples.
- Maximum: $\max(x_i)$, the greatest sample value, relative to the rest of the samples.

4.2.2 Message Types

Table 1 contains the set control packet messages specified by the CoAP standard [1].

Table 1: Message Types

Control Packet Name	Description	Client -> Server	Server -> Client	Payload
GET	Retrieves a representation for the information that currently corresponds to the resource identified by the request URI.	✓		Required
POST	Requests that the representation enclosed in the request be processed.	✓		None
PUT	Requests that the resource identified by the request URI be updated or created with the enclosed representation.	✓		Required
DELETE	Requests that the resource identified by the request URI be deleted.	✓		None
Success 2.xx	This class of Response Code indicates that the clients request was successfully received, understood, and accepted.		✓	None
2.01 Created	Like HTTP 201 "Created", but only used in response to POST and PUT requests.		✓	Optional
2.02 Deleted	This Response Code is like HTTP 204 "No Content" but only used in response to requests that cause the resource to cease being available, such as DELETE and, in certain circumstances, POST.		✓	Optional
2.03 Valid	This Response Code is related to HTTP 304 "Not Modified" but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid.		✓	None
2.04 Changed	This Response Code is like HTTP 204 "No Content" but only used in response to POST and PUT requests.		✓	Optional
2.05 Content	This Response Code is like HTTP 200 "OK" but only used in response to GET requests.		✓	Required
Client Error 4.xx	This class of Response Code is intended for cases in which the client seems to have erred. These Response Codes are applicable to any request method.		✓	Optional
4.00 Bad Request	This Response Code is Like HTTP 400 "Bad Request".		✓	None

Control Packet Name	Description	Client -> Server	Server -> Client	Payload
4.01 Unauthorized	The client is not authorized to perform the requested action.		✓	None
4.02 Bad Option	The request could not be understood by the server due to one or more unrecognized or malformed options.		✓	None
4.03 Forbidden	This Response Code is like HTTP 403 "Forbidden".		✓	None
4.04 Not Found	This Response Code is like HTTP 404 "Not Found".		✓	None
4.05 Method Not Allowed	This Response Code is like HTTP 405 "Method Not Allowed" but with no parallel to the "Allow" header field.		✓	None
4.06 Not Acceptable	This Response Code is like HTTP 406 "Not Acceptable", but with no response entity.		✓	None
4.12 Precondition Failed	This Response Code is like HTTP 412 "Precondition Failed".		✓	None
4.13 Request Entity Too Large	This Response Code is like HTTP 413 "Request Entity Too Large".		✓	None
4.15 Unsupported Content-Format	This Response Code is like HTTP 415 "Unsupported Media Type".		✓	None
Server Error 5.xx	This class of Response Code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These Response Codes are applicable to any request method.		✓	Optional
5.00 Internal Server Error	This Response Code is like HTTP 500 "Internal Server Error".		✓	Optional
5.01 Not Implemented	This Response Code is like HTTP 501 "Not Implemented".		✓	Optional
5.02 Bad Gateway	This Response Code is like HTTP 502 "Bad Gateway".		✓	Optional
5.03 Service Unavailable	This Response Code is like HTTP 503 "Service Unavailable" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry.		✓	Optional
5.04 Gateway Timeout	This Response Code is like HTTP 504 "Gateway Timeout".		✓	Optional
5.05 Proxying Not Supported	The server is unable or unwilling to act as a forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme.		✓	Optional

4.2.3 Test parameters

The benchmark test parameters are used to control the behaviour of the test script. The data elements required to configure the test system are listed in table 2.

Table 2 is a non-exhaustive list of test parameters defined for the benchmark standard. The list is expected to grow over time, as additional subsystems and system configurations are developed.

Table 2: Test parameters

Parameter	Description
Duration	Amount of time that a system load is presented to a SUT
Type of call	Type of messages contained within a workload
NoC	number of clients generating or subscribing to data/control traffic
NoS	Number of servers handling data/control traffic
Transport interface	Underlying transport interface
WLF for GTW	Work load factor for gateway expressed in number messages received per second, by type of message
Payload	Size of the data in Bytes carried within a message
Monitoring Window(s)	The time interval window for which the monitored metrics are recorded. This reflects the measuring accuracy (e.g. per second, minute, hour, etc.)
Validation threshold(s)	The specific metric thresholds used for validating whether a system performs at specifications

Table 3: Test output

Metric	Description
Minimum call duration	The minimum duration of a successful message request/response interaction within a Monitoring Window
Maximum call duration	The maximum duration of a successful message request/response interaction within a Monitoring Window
Average call duration	The average duration of a successful message request/response interaction within a Monitoring Window
Total number of calls	The total number of workload specified request/response type operations executed during the test
Success rate	Percentage number of successful workload operations relative to the total workload operations
Error rate	Percentage number of failed workload operations relative to the total workload operations
Requests processed per time unit	This metric reflects the average number of successfully processed requests per preferred time unit (second/minute/etc.)

4.2.4 Operation Message Flows

The IUT will be evaluated based on the metric values obtain as a result of the service operations using the messages described in clause 4.2.2. The set of messages exchanged triggered by the initial client message are further referred as operations. For the tests, the metrics use operations rather than specific messages because it is easier to handle the measurements. If specific test system network measurements are available, by subtracting the measured network delayed from the duration, the operation processing time can be deducted.

1) GET: This section describes the CoAP operation types and message sequences required for test execution using a Client GET example.

Preconditions:

- Client, Server
- TCP/UDP connection between Client and Server established

Operation sequence:

- Client sends GET message
- Client receives SUCCESS (Content) message

Measurement: Time period expressed in milliseconds between the moment client forwards the GET Message and the moment Client receives SUCCESS message from server.

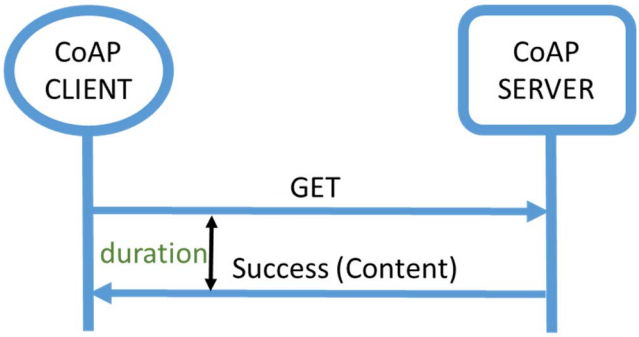

Figure 1: CoAP Server-Client GET message flow

2) PING: This section describes the CoAP operation types and message sequences required for test execution using a Client ping example.

Preconditions:

- Connected Client, Server

Operation sequence:

- Client sends PING message
- Client receives PONG message

Measurement: Time period expressed in milliseconds between the moment client forwards the PING Message and the moment Client receives PONG message from server.

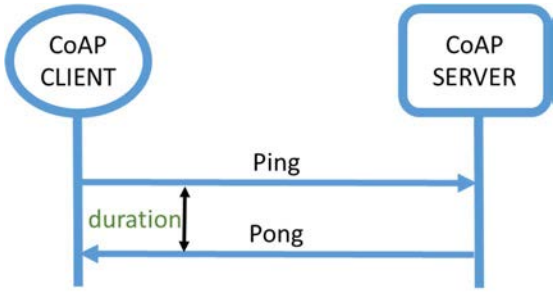

Figure 2: CoAP Server-Client PING message flow

3) POST: This section describes the CoAP operation types and message sequences required for test execution using a Client POST example.

Preconditions:

- Connected Client, Server

Operation sequence:

- Client sends POST message
- Client receives SUCCESS (Updated) message

Measurement: Time period expressed in milliseconds between the moment client forwards the POST Message and the moment Client receives SUCCESS message from server.

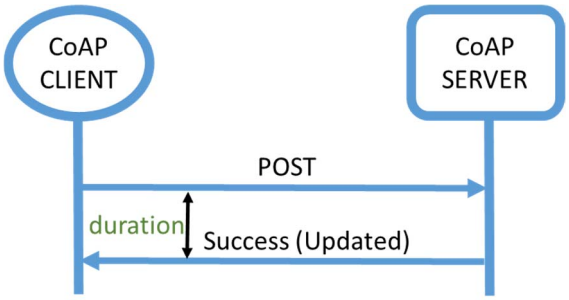

Figure 3: CoAP Server-Client POST message flow

4) PUT: This section describes the CoAP operation types and message sequences required for test execution using a Client PUT example.

Preconditions:

- Connected Client, Server

Operation sequence:

- Client sends PUT message
- Client receives Success (created/updated) message

Measurement: Time period expressed in milliseconds between the moment client forwards the PUT Message and the moment Client receives Success (created/updated) message from server.

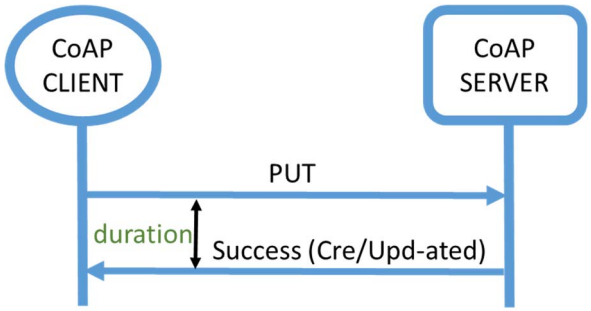

Figure 4: CoAP Server-Client PUT message flow

5) DELETE: This section describes the CoAP operation types and message sequences required for test execution using a Client DELETE example.

Preconditions:

- Connected Client, Server

Operation sequence:

- Client sends DELETE message
- Client receives Success (deleted) message

Measurement: Time period expressed in milliseconds between the moment client forwards the DELETE Message and the moment Client receives Success (deleted) message from server.

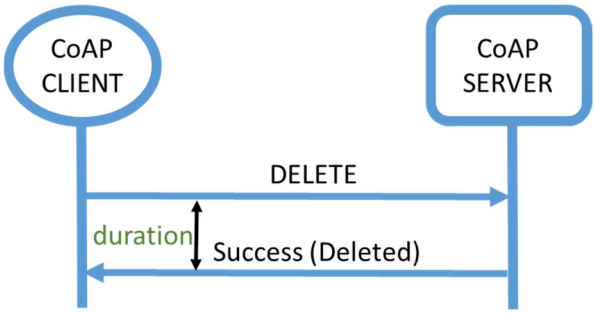

Figure 5: CoAP Server-Client DELETE message flow

4.2.5 Test Campaign Parameters

For test-campaign evaluations, a sequence of control packet exchanges is specified. This allows for defining multiple types of test benchmarks based on client-server interactions. This may address but not be limited to the following:

- Duration
- Control packet sending order
- Number of control packets sent per session
- Number of application message packets sent

The parameters also cover the type, sequence, characteristics and flow of messages sent towards the IUT.

- Type: message types from the ones listed in the previous tables
- Sequence: a logical message flow sequence targeting the IUT behaviour (e.g. GET, POST, PUT, DELETE)
- Characteristics: the specific message header flags and payload size (e.g. Confirmable)
- Flow: the explicit message flow to be sent towards the IUT (e.g. 1xPUT, 1 000x POST, 1xDELETE)

The following metrics listed in clauses 4.3 to 4.5 are derived from ETSI TR 101 577 [i.2]. The measuring intervals (also referred in clause 4.2.3 as "monitoring windows") and metric validation thresholds for all metrics can vary depending on test requirements and are part of the test input parameters.

4.3 Powerfulness metrics

The powerfulness category of performance characteristics contains indicators of speed and quantity of service production. The following metrics are derived from ETSI TR 101 577 [i.2]:

- Capacity: ability to accept incoming service requests per time unit. For this metric set, the type of messages considered for evaluation are GET, POST, PUT and DELETE For each of the messages the input parameters specify number and type of requests per second and duration. The metrics consist of minimum, maximum and average response time, total number of successful calls, failed calls and pending calls.
- Responsiveness: time to handle a service request, e.g. subscription, connect request, etc.; transmission time, roundtrip time; publication (publish) time. Messages used are GET, POST, PUT and DELETE. The metrics consist of minimum, maximum and average response time, total number of successful calls, failed calls and pending calls. The measuring interval and metric validation thresholds can vary depending on user requirements.

4.4 Reliability metrics

The reliability category of performance characteristics contains indicators of how predictable a system's service production is. The performance category has subcategories for Quality-of-Service, Stability, Availability, Robustness, Recovery, and Correctness.

Stability - Stability characteristics are indicators of a system's ability to maintain measured performance figures for powerfulness and efficiency during service delivery regardless of time. This is a performance metric that is usually validated through endurance testing i.e. testing over longer periods of time.

Availability - Availability characteristics are indicators of a system's ability to delivery services over time. Different availability characteristics are applied on hardware (physical availability) and on software (logical availability). This is a performance metric that is usually validated through endurance testing i.e. testing over longer periods of time.

Robustness - Robustness characteristics are indicators of a system's services levels, i.e. services capacity and/or service responsiveness under extreme conditions. Extreme conditions can be caused internally by hardware failure or software malfunctioning, or externally by extreme peak load conditions, or by denial-of-service attacks or other malice attempts. Messages used are GET, POST, PUT and DELETE. The metrics consist of minimum, maximum and average response time, total number of successful calls, failed calls and pending calls.

Recovery - Recovery characteristics are indicators of production disturbances from hardware or software malfunctioning. Recovery covers a large number of different operations. In this context system recovery and service recovery is considered. i.e. the time duration of a system to become operational after a reset.

Correctness - Correctness characteristics are indicators of a system's ability to deliver correctly processed service requests under high or odd load conditions. Messages used are GET, POST, PUT and DELETE. The metrics consist of minimum, maximum and average response time, total number of successful calls, failed calls and pending calls.

4.5 Efficiency metrics

The following metrics are derived from ETSI TR 101 577 [i.2].

The performance category efficiency contains different types of indicators of resource usage and resource utilization. These metrics can be recorded additional to the Powerfulness metrics during the specified tests. The scope of these are to determine system deployment resource requirements as well as scaling capabilities.

Resource usage - amount of CPU, Memory, Disk used by a server/client component during a test. Metric measurements are to be correlated with the other metrics, i.e. CPU usage during a 1 000 PUT messages per second monitoring window, with a min, max and average load as output.

Scalability - amount of resource usage increase through scaling horizontally (multiple client/server instances) or vertically (increase resource usage for CPUs/bandwidth capacity) relative to the load capacity increase. E.g. the CPU usage at 1 000 requests processed per time unit is 1 % and at 10 000 requests per processed time unit is 10 % would express a linear relation between the CPU usage and workload for the given parameters. This varies greatly with the test system characteristics and testing parameters, nevertheless it is very relevant for production services as an operational feature.

5 Configurations

The performance test configurations are derived from the SUT access points and functional test configurations.

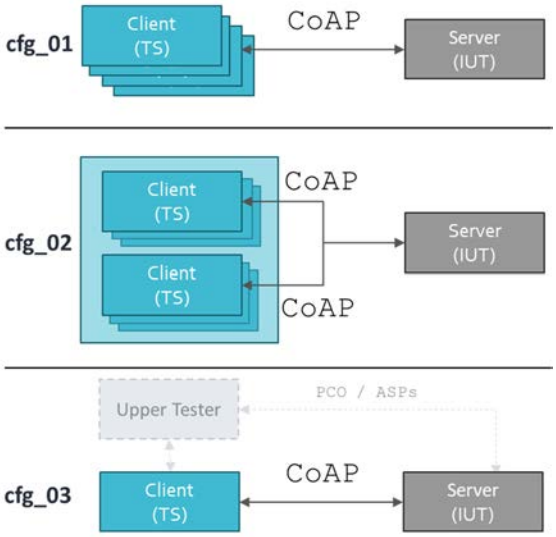

Figure 6: CoAP Server-Client test configurations

Furthermore, there are consideration following possible performance specific load and measurement tools.

6 Benchmarking

6.1 Generic adjustments

This clause specifies the CoAP protocol performance metrics. As a protocol evaluation rationale, the metrics presented herein are directly addressed to IUTs: systems and associated components implementing the protocol, namely clients and servers. No requirements will be expressed regarding specifics of such IUTs in terms of implementation, technology, or proprietary elements. The metrics expressed herein are derived from the protocol specification and assume that both client and server components are conformant with the CoAP protocol specification as defined in IETF RFC 7252 [1]. For conformance testing specifications, please refer to ETSI TS 103 596-1 [2].

6.2 Benchmarking Methodology

This clause aims to describe a viable methodology for benchmarking the performance of an IUT. In the case of CoAP, an IUT may consist of one or several Servers or one or several Clients. The approach is inspired from the examples in IETF RFC 2544 [i.1] from the point of view of measurement preconditions, approach and statistically consistent measurement sampling.

As a general precondition for performance benchmarking, a functionally correct implementation is a prerequisite. For this the general assumption is that the IUT has passed the conformance testing described in ETSI TS 103 596-1 [2].

First, the System under Test is described, including hardware, resource manager (bare-metal/virtualisation technology) and network connectivity (type-wired/ait, latency, throughput capacity). This includes both the resources dedicated to the IUT as well the ones for the test system.

Second, the type of performance benchmarking is established: whether the tests aim to determine the system KPI values or the tests aim to check whether the system meets established performance requirements. Depending on the objective, the approach will differ. In this step the KPIs and metrics w/o thresholds are selected. Two examples are presented in clause 6.4 reflecting the specific approach.

Third, the appropriate tests are selected and the test input parameters are specified. These include the test types, duration, metric threshold requirements, sample size and validation checks. Then, the monitoring system is configured, and the appropriate metrics (see clauses 4.3 to 4.5) are selected for observation.

Fourth, the tests are executed and the metrics are collected. For this stage it is highly relevant that the TS and IUT are not affected by external factors in terms of compute and network resources. As an example, the monitoring system load on both the network and compute resources is commonly not taken into consideration and this leads to skewed results.

Finally, the test results are checked and validated leading to a verdict whether the performance tests are passed or failed.

6.3 Metric examples

This clause presents a series of benchmark metric examples:

- GET delay: the time delay between GET message and server Content response. Value expressed in milliseconds (ms).
- POST-delay: the time interval starts when a Post message is sent and ends when the corresponding Created/Changed message has been received back. Value expressed in milliseconds (ms).
- PUT delay: the time interval starts when a Put message is sent and ends when the corresponding Created/Changed message has been received. Value expressed in milliseconds (ms).
- DELETE delay: the time between Delete and Deleted message. Value expressed in milliseconds (ms).

6.4 Benchmark examples

6.4.0 Introduction

For evaluating the metrics described in clause 4, the benchmark tests can be grouped in 3 main categories.

1) Load Tests: These tests are used for determining or validating the IUT workload range. The workload consists of one or multiple message exchanges between the IUT and the Test System. The aim is to observe the Powerfulness and Efficiency metrics as well as the Correctness (Reliability category) metric in order to determine or validate the maximum operating workload handled by the IUT.

2) Endurance Tests: These tests are used for determining or validating the IUT Reliability and Efficiency. These tests generally consist of exposing the IUT to a variable or high operational workload for long periods of time. The metrics observed are the Reliability and Efficiency ones. This type of testing covers operational aspects such as degradation over time, memory leaks and resource consumption estimates.

3) Stress Tests: These tests are used for determining or validating the IUT Robustness and Recovery (Reliability category) metrics. This is achieved by injecting workload spikes throughout the test and observing the degradation and recovery patterns of the system as well as the maximum workload operational limits.

The following two benchmarking examples reflect the two types of performance benchmarking evaluations specified in clause 6.2.

6.4.1 KPI Determination

The KPI determination benchmark is an exploratory performance evaluation that aims to determine the operational performance of an IUT. The KPIs are specified as an input. The scope of this evaluation is to establish a reliable indication of how the SUT is expected to perform in production. The KPIs are determined according to the intended use-case scenario for the IUT.

As a first example, a device manufacturer has finished a hardware CoAP server prototype for the industrial IoT market. The target objective is to provide communication in small industrial buildings serving a potential capacity of 50 to 5 000 CoAP clients. The expected use-case requires confirmable headers for data transmission and foresees a 1 000 to 10 000 messages per second load. Additionally, the manufacturer wants to determine the system reliability, specifically Stability, Availability and Correctness.

According to clause 7.2, the first step is to specify the SUT hardware and network resources. In this example the underlying hardware is considered to be a bare-metal SoC box running Ubuntu 18.04 OS. It has a dual core 2,4 GHz CPU, 2 Gb or RAM, 120 GB SSD and a 1 Gb Ethernet connection. For simplification, the network is considered wired, with a 1 Gbps throughput running TCP/IP over a 1 000BASE-T Ethernet LAN connection with an estimated 1 ms end-to-end delay for all connections. The Test System (TS) consists of a quad-core power pc with 2,4 GHz CPU, 8 Gb of RAM, 500 GB SSD and 1 Gb Ethernet connection. The TS is deployed in virtual containers running over a Unix environment with direct access to the network (non-virtualised network connection). The monitoring system resource consumption is considered negligible.

The second step is to select the KPIs and metrics of interest. The KPIs selected by the manufacturer are Capacity (max number of POST requests handled per second), Responsiveness (average delay of processing client requests), Number of registered clients, resource usage and stability. The selected associated metrics are as follows:

- GET delay: the time interval starts when a GET message is sent and ends when the corresponding Content message has been received. Value expressed in milliseconds (ms).
- POST delay: the time interval starts when a PUT message is sent and ends when the corresponding Changed message has been received. Value expressed in milliseconds (ms).
- Capacity: number of successfully handled messages per second. Threshold is initially set to 1 000.
- Correctness: percentage of successfully handled messages per second.
- Resource usage - amount of CPU, Memory used by the IUT during the Capacity evaluation. with a min, max and average values.

The third step consists of determining the tests and configuring the monitoring system. According to the KPI requirements, the type of tests required are load testing for determining the IUT Capacity and Resource usage and Endurance testing for determining the system Responsiveness, Correctness and Availability. The monitoring system is configured to record GET delay, POST delay, rate of success for Post messages, CPU, Mem and Network in/out usage. The monitoring window is set to 1 second and post-processing averages are configured to cover 1 minute for load tests and 1 hour for endurance tests.

Load Testing

EXAMPLE: 1 000 Clients connect to the IUT.

Each client starts sending POST messages at a rate of 1 message per second. The rate increases by 1 up to a maximum of 10 every minute. The test duration is set to 10 minutes and executed 10 times. The test is repeated for an incremented number of connected clients up to 5 000 in incremental steps of 1 000 clients. The input parameters are no longer incremented if the test fails. Test duration: 10 minutes per test.

1) Metric: number of connected clients
2) Metric: number of POST messages processed per second
3) Metric: average POST messages processing delay
4) Metric: rate of successfully processed POST messages
5) Metric: CPU load user time %
6) Metric: Memory load (Mb)
7) Metric: Network Packet In (Kb)
8) Metric: Network Packet Out (Kb)

Fail criteria: rate of successfully processed POST messages falls under 90 %

Fail criteria: CPU load goes over 80 %

Fail criteria: Memory load goes over 90 %

Pass criteria: Test ended without fail criteria triggered

Endurance Test

Starting from the highest load successfully passed, the test parameters are noted with maxX where X is the metric from the Load test.

Ex: max Clients connect to the IUT.

Each client starts sending POST messages at a rate of max message per second. The rate remains constant. The test duration is set to 600 minutes and executed 10 times. The test is repeated for an decremented number of connected clients down to 1 000 in decremented steps of 1 000 clients in case of failure. The input parameters are no longer decremented if the test succeeds. Test duration: 600 minutes per test.

1) Metric: number of connected clients
2) Metric: number of POST messages processed per second
3) Metric: average POST messages processing delay
4) Metric: rate of successfully processed POST messages
5) Metric: CPU load user time %
6) Metric: Memory load (Mb)
7) Metric: Network Packet In (Kb)
8) Metric: Network Packet Out (Kb)

Fail criteria: rate of successfully processed POST messages falls under 90 %

Fail criteria: CPU load goes over 80 %

Fail criteria: Memory load goes over 90 %

Pass criteria: Test ended without fail criteria triggered

The fourth step consists of executing the tests. As a general precondition: the SUT is operational - IUT CoAP Server is active. TS is operational and connected to the SUT. Finally the results are collected the Powerfulness, Reliability and Efficiency selected KPI values are determined.

6.4.2 KPI Validation

The KPI validation benchmark is performance evaluation that aims to validate whether the IUT performs according to requirement specifications. The KPIs and their thresholds are specified as an input. The scope of this evaluation is to establish a reliable indication of how the SUT is expected to perform in production. The KPIs are determined according to the intended use-case scenario for the IUT.

In this second example, a factory owner contacts the previous device manufacturer and specifies its requirements. These consist of providing communication in small industrial buildings serving a total of 1 000 CoAP clients. The expected use-case requires confirmable message headers for data transmission and a variable load of 1 000 to 5 000 messages per second load. Additionally, the factory owner requires that the POST delay is no longer than 1 second and that 99 % of the messages are successfully transmitted.

According to clause 7.2, the first step is to specify the SUT hardware and network resources. In this example the underlying hardware is considered to be a bare-metal SoC box running Ubuntu 18.04 OS. It has a dual core 2,4 GHz CPU, 2 Gb or RAM, 120 GB SSD and a 1 Gb Ethernet connection. For simplification, the network is considered wired, with a 1 Gbps throughput running TCP/IP over a 1 000BASE-T Ethernet LAN connection with an estimated 1 ms end-to-end delay for all connections. The Test System (TS) consists of a quad-core power pc with 2,4 GHz CPU, 8 Gb of RAM, 500 GB SSD and 1 Gb Ethernet connection. The TS is deployed in virtual containers running over a Unix environment with direct access to the network (non-virtualised network connection). The monitoring system resource consumption is considered negligible.

The second step is to select the KPIs and metrics of interest. The KPIs selected by the manufacturer are Capacity (max number of POST requests handled per second), Responsiveness (average delay of processing client requests), Number of registered clients, resource usage and stability. The selected associated metrics are as follows:

- POST delay: the time interval starts when a POST message is sent and ends when the corresponding Changed message has been received. Value expressed in milliseconds (ms). Threshold set to 1 000 ms.
- Capacity: number of successfully handled POST messages per second. Threshold is set to 5 000.
- Correctness: percentage of successfully handled POST messages per second. Threshold is set to 99 %.
- Resource usage - amount of CPU, Memory used by the IUT during the Capacity evaluation. with a min, max and average values. Threshold is set to 80 % CPU load, 90 % Mem load.

The third step consists of determining the tests and configuring the monitoring system. According to the KPI requirements, the type of tests required are load testing for validating the IUT Capacity and Resource usage and Endurance testing for determining the system Responsiveness, Correctness and Availability. The monitoring system is configured to record POST delay, rate of success for POST messages, CPU, Mem and Network in/out usage. The monitoring window is set to 1 second and post-processing averages are configured to cover 1 minute for load tests and 1 hour for endurance tests.

Load Testing

EXAMPLE: 1 000 Clients connect to the IUT.

Each client starts sending POST messages at a rate of 1 message per second. The rate increases by 1 up to a maximum of 5 every 2 minutes. The test duration is set to 10 minutes and executed 10 times. Test duration: 10 minutes per test.

1) Metric: number of connected clients
2) Metric: number of POST messages processed per second
3) Metric: average POST messages processing delay
4) Metric: rate of successfully processed POST messages
5) Metric: CPU load user time %
6) Metric: Memory load (Mb)
7) Metric: Network Packet In (Kb)
8) Metric: Network Packet Out (Kb)

Fail criteria: rate of successfully processed POST messages falls under 99 %

Fail criteria: CPU load goes over 80 %

Fail criteria: POST delay max value is greater than 1 000 ms

Fail criteria: Memory load goes over 90 %

Pass criteria: Test ended without fail criteria triggered

Endurance Test

EXAMPLE: 1 000 Clients connect to the IUT.

Each client starts sending POST messages at a rate of 5 message per second. The rate remains constant. The test duration is set to 600 minutes and executed 10 times. Test duration: 600 minutes per test.

1) Metric: number of connected clients
2) Metric: number of POST messages processed per second
3) Metric: average POST messages processing delay

4) Metric: rate of successfully processed POST messages

5) Metric: CPU load user time %

6) Metric: Memory load (Mb)

7) Metric: Network Packet In (Kb)

8) Metric: Network Packet Out (Kb)

Fail criteria: rate of successfully processed POST messages falls under 99 %

Fail criteria: CPU load goes over 80 %

Fail criteria: Memory load goes over 90 %

Fail criteria: POST delay max value is greater than 1 000 ms

Pass criteria: Test ended without fail criteria triggered

The fourth step consists of executing the tests. As a general precondition: the SUT is operational - IUT CoAP Server is active. TS is operational and connected to the SUT. Finally the results are collected the Powerfulness, Reliability and Efficiency selected KPI values are determined.

7 Examples of tests

7.0 Introduction

A test should be presentable as a document, with accompanying data files, that provides a full description of an execution of a performance test on a test system. Description of the test case and objective of the test case, e.g. the definition of the targeted metrics should be contained therein. The following sections should be addressed in general:

- Test procedure: Description of the execution of the test:
  - Test sequence to run the test case: sequence of actions for running the experiment and collect the measurements needed to compute the metrics.
  - Test duration (per iteration).
  - Number of iterations of the experiment. Number of repetitions of the experiments to obtain relevant statistical results.
  - Measurements collected to compute the metrics.
- Procedure for metrics calculation: Description of the procedure applied (statistical aggregation, algorithm, etc.) to compute the metrics based on the raw measurements collected.
- Expected output of the test case:
  - Test report.

A set of Test Purposes (TPs) are provided as reference examples in Annex A.

7.1 Test Objectives

In this clause, 2 formalized test objective examples are presented. A set of test purposes are presented in Annex A.

EXAMPLE 1: Server Post Load Test

Determine if the IUT (server) can handle the given POST message load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.

EXAMPLE 2: Server POST/GET Load Test

Determine if the IUT (broker) can handle the given POST message load and reply the contents them to a given number of clients that make GET requests for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.

7.2 Test Purpose

In this clause, a formalized test purpose example is provided. A set of test purposes are presented in Annex A.

Precondition:

- The IUT is running and configured to accept client connections.
- Client(s) send PUT messages.
- Client(s) start sending POST messages at a specified rate (messages per second) totalling a specified duration (time) or number of messages (e.g. 10 000 messages).
- After finishing the execution, the client(s) send DELETE messages.

Expected behaviour:

- The IUT will reach the given time interval of handling the given load without exceeding the delay/message loss threshold.
- The IUT will not reach the given time interval of handling the given load before exceeding the delay/message loss threshold.

Expected output:

- Minimal, maximal and average measurements of selected metrics.

7.3 Test Report

The test report will include the test execution time, test parameters (e.g. number of messages, rate) and measurement/aggregated measurement results (e.g. the average POST/Updated message sequence latency was 3 ms end-to-end).

Table 4: Performance load test report for GET example

Test Number	**T-01** **Test Category**	**Performance** **Test Type**	**Load Testing**
Test Objective	"Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate."		
Test Description	Test Scenario 1 Test Case 1 Configuration 1: Against Californium Server	Test Scenario 1 Test Case 1 Configuration 2: Against nCoAP Server	Reference 2 ms
Preconditions	the CLIENT having a UDP_CONNECTION to the IUT		

Expected Behaviour	ensure that {
	when {
	(.) at time point t1: the tester entity send multiple GET messages and assure the INCREMENTAL_RATE and
	(!) during the INTERVAL after t1:
	the IUT entity receive multiple GET messages;
	}
	then {
	(!) INTERVAL after t1:
	the IUT entity assure and send the SUCCESS messages and
	the IUT entity assure the packet_loss_limit and
	the IUT entity assure the DELAY;
	}		

Output Measurements	Average GET/ server Content response delay in ms (KPI) Publish Success Rate	Sequence Delay	Reference
Values C1	100 %	0,98 ms	2 ms
Values C2	100 %	0,95 ms	2 ms

Annex A (informative): CoAP Test Purposes (TPs)

TP Id	TP_CoAP_Performance_Server_Load_001
Test Objective	CoAP GET load test for broker: Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	Null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple GET messages and assures the INCREMENTAL_RATE and
(!) during INTERVAL after t1:
the IUT receives several GET messages and // 0.01
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.05;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Load_002
Test Objective	CoAP POST load test for broker: Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple POST messages and assures the INCREMENTAL_RATE and
(!) during INTERVAL after t1:
the IUT receives several POST messages and // 0.01
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.04;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions

TP Id	TP_CoAP_Performance_Server_Load_003
Test Objective	CoAP PUT load test for broker: Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple PUT messages and assures the INCREMENTAL_RATE and
(!) during INTERVAL after t1:
the IUT receives several PUT messages and // 0.03
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2 ;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Load_004
Test Objective	CoAP DELETE load test for broker: Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple DELETE messages and assures the INCREMENTAL_RATE and // 0.04
(!) during INTERVAL after t1:
the IUT receives several DELETE messages containing
uri_path corresponding to DELETE_RESOURCE; and
(!) during INTERVAL after t1:
the IUT sends the Deleted messages containing
code indicating value 2.02;
}
then {
(!) INTERVAL after t1
the IUT assures and sends the response messages and
the IUT assures the DELAY
}
}	

Final Conditions

TP Id	TP_CoAP_Performance_Server_Load_005
Test Objective	CoAP PING load test for broker: Determine if the IUT(server) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple PING messages and assures the INCREMENTAL_RATE and
(!) during INTERVAL after t1:
the IUT receives several Ping messages and // 7.02
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 7.03;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Endurance_001
Test Objective	CoAP GET endurance test for broker: Determine if the IUT(server) can handle the given load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	Null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple GET messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several GET messages and // 0.01
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.05;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Endurance_002
Test Objective	CoAP POST endurance test for broker: Determine if the IUT(server) can handle the given load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	Null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple POST messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several POST messages and // 0.01
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.04;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Endurance_003
Test Objective	CoAP PUT endurance test for broker: Determine if the IUT(server) can handle the given load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple PUT messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several PUT messages and // 0.03
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2 ;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Endurance_004
Test Objective	CoAP DELETE endurance test for broker: Determine if the IUT(server) can handle the given load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple DELETE messages and assures the RATE and // 0.04
(!) during INTERVAL after t1:
the IUT receives several DELETE messages containing
uri_path corresponding to DELETE_RESOURCE; and
(!) during INTERVAL after t1:
the IUT sends the Deleted messages containing
code indicating value 2.02;
}
then {
(!) INTERVAL after t1
the IUT assures and sends the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Endurance_005
Test Objective	CoAP PING endurance test for broker: Determine if the IUT(server) can handle the given load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple PING messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several PING messages and // 7.02
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 7.03;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Stress_001
Test Objective	CoAP GET stress test for broker: Determine if the IUT(server) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple GET messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several GET messages and // 0.01
(.) at time point t2:
the clients sends multiple GET messages and assures the SPIKE_RATE and
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.05;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assures the packet_loss_limit and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Stress_002
Test Objective	CoAP POST stress test for broker: Determine if the IUT(server) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple POST messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several POST messages and // 0.01
(.) at time point t2:
the clients sends multiple POST messages and assures the SPIKE_RATE and
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2.04;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assure the packet_loss_limit and
the IUT assures the DELAY
}
}	

Final Conditions

TP Id	TP_CoAP_Performance_Server_Stress_003
Test Objective	CoAP PUT stress test for broker: Determine if the IUT(server) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	Null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple PUT messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several PUT messages and // 0.03
(.) at time point t2:
the clients sends multiple PUT messages and assures the SPIKE_RATE and
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 2 ;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assure the packet_loss_limit and
the IUT assures the DELAY
}
}	

Final Conditions	

TP Id	TP_CoAP_Performance_Server_Stress_004
Test Objective	CoAP DELETE stress test for broker: Determine if the IUT(server) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple DELETE messages and assures the RATE and // 0.04
(!) during INTERVAL after t1:
the IUT receives several DELETE messages containing
uri_path corresponding to DELETE_RESOURCE; and
(.) at time point t2:
the clients sends multiple DELETE messages and assures the SPIKE_RATE and
(!) during INTERVAL after t1:
the IUT sends the Deleted messages containing
code indicating value 2.02;
}
then {
(!) INTERVAL after t1
the IUT assures and sends the response messages and	

the IUT assure the packet_loss_limit and the IUT assures the DELAY } }
Final Conditions

TP Id	TP_CoAP_Performance_Server_Stress_005
Test Objective	CoAP PING stress test for broker: Determine if the IUT(server) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate.
Reference	IETF RFC 7252 [1] IETF RFC 8323 [3]
PICS Selection	null
Initial Conditions	

with {
the IUT being_in initial_state
}	

Expected Behaviour	

ensure that {
when {
(.) at time point t1:
the clients sends multiple Ping messages and assures the RATE and
(!) during INTERVAL after t1:
the IUT receives several Ping messages and // 7.02
(.) at time point t2:
the clients sends multiple Ping messages and assures the SPIKE_RATE and
(!) during INTERVAL after t1:
the IUT sends multiple messages containing
code indicating value 7.03;
}
then {
(!) INTERVAL after t1
the IUT assures the response messages and
the IUT assure the packet_loss_limit and
the IUT assures the DELAY
}
}	

Final Conditions	

This Test purpose catalogue has been produced using the Test Description Language - Test Objectives (TDL-TO) according to ETSI ES 203 119-4 [4]. The TDL-TO library modules corresponding to the Test purpose catalogue are contained in archive ts_10359603v010101p0.zip which accompanies the present document.

History

Document history		
V1.1.1	May 2021	Publication