MQTT for Sensor Networks (MQTT-SN) Version 2.0

Working Draft 9

05 May 2021

Technical Committee:

[OASIS Message Queuing Telemetry Transport (MQTT) TC](https://www.oasis-open.org/committees/mqtt/)

Chairs:

Ian Craggs ([icraggs@gmail.com](mailto:icraggs@gmail.com)), Individual member

Allan Stockdill-Mander ([asm@uk.ibm.com](mailto:asm@uk.ibm.com)), [IBM](http://www.ibm.com)

Editors:

Ian Craggs ([icraggs@gmail.com](mailto:icraggs@gmail.com)), Individual member

Andrew Banks ([Andrew\_Banks@uk.ibm.com](mailto:Andrew_Banks@uk.ibm.com)), [IBM](http://www.ibm.com)

Simon Johnson ([simon.johnson@u-blox.com](mailto:simon.johnson@u-blox.com)), Ublox AG

Rahul Gupta ([rahul.gupta@us.ibm.com](mailto:rahul.gupta@us.ibm.com)), [IBM](http://www.ibm.com)

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

* XML schemas: (list file names or directory name)
* Other parts (list titles and/or file names)
* (**Note:** Any normative computer language definitions that are part of the Work Product, such as XML instances, schemas and Java(TM) code, including fragments of such, must be (a) well formed and valid, (b) provided in separate plain text files, (c) referenced from the Work Product; and (d) where any definition in these separate files disagrees with the definition found in the specification, the definition in the separate file prevails. Remove this note before submitting for publication.)

Related work:

This specification is related to:

* *MQTT Version 5.0*. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. OASIS Standard. Latest version: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.
* *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta. OASIS Standard. Latest version: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

Abstract:

This specification defines the MQTT for Sensor Networks protocol (MQTT-SN). It is closely related to the MQTT v3.1.1 and MQTT v5.0 standards.

Status:

This [Working Draft](https://www.oasis-open.org/policies-guidelines/oasis-defined-terms-2017-05-26#dWorkingDraft) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](https://www.oasis-open.org/policies-guidelines/tc-process#committeeDraft) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](https://www.oasis-open.org/policies-guidelines/tc-process#standApprovProcess) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

This specification is provided under the [Non-Assertion](https://www.oasis-open.org/policies-guidelines/ipr#Non-Assertion-Mode) Mode of the [OASIS IPR Policy](https://www.oasis-open.org/policies-guidelines/ipr), the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (<https://www.oasis-open.org/committees/mqtt/ipr.php>).

Note that any machine-readable content ([Computer Language Definitions](https://www.oasis-open.org/policies-guidelines/tc-process#wpComponentsCompLang)) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

URI patterns:

Initial publication URI:  
https://docs.oasis-open.org/mqtt/mqtt-sn/v1.3/csd01/mqtt-sn-v1.3-csd01.docx

Permanent "Latest stage" URI:  
https://docs.oasis-open.org/mqtt/mqtt-sn/v1.3/mqtt-sn-v1.3.docx

(**Note:** Publication URIs are managed by OASIS TC Administration; please don't modify. The [OASIS TC Process](https://www.oasis-open.org/policies-guidelines/tc-process#wpComponentsGeneral) requires that Work Products at any level of approval must use the [OASIS file naming scheme](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html), and must include the OASIS copyright notice. The URIs above have been constructed according to the file naming scheme. Remove this note before submitting for publication.)

Copyright © OASIS Open 2020. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](https://www.oasis-open.org/policies-guidelines/ipr) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

[1 Introduction 7](#_Toc71106212)

[1.1 IPR Policy 7](#_Toc71106213)

[1.2 Terminology 7](#_Toc71106214)

[1.3 Normative References 7](#_Toc71106215)

[1.4 Non-Normative References 7](#_Toc71106216)

[1.5 Background 8](#_Toc71106217)

[1.6 MQTT-SN comparison with MQTT 9](#_Toc71106218)

[1.7 MQTT-SN Architecture 10](#_Toc71106219)

[1.7.1 Transparent Gateway 10](#_Toc71106220)

[1.7.2 Aggregating Gateway 11](#_Toc71106221)

[2 Data representation 12](#_Toc71106222)

[2.1 Bits (Byte) 12](#_Toc71106223)

[2.2 Two Byte Integer 12](#_Toc71106224)

[2.3 Four Byte Integer 12](#_Toc71106225)

[2.4 UTF-8 Encoded String 12](#_Toc71106226)

[3 MQTT-SN Control Packet format 14](#_Toc71106227)

[3.1 Structure of an MQTT-SN Control Packet 14](#_Toc71106228)

[3.1.1 Packet Header 14](#_Toc71106229)

[3.1.2 Length 14](#_Toc71106230)

[3.1.3 MQTT-SN Control Packet Type 15](#_Toc71106231)

[3.2 MQTT-SN Control Packet Variable Part 16](#_Toc71106232)

[Publish 16](#_Toc71106233)

[3.2.1 Flags 16](#_Toc71106237)

[3.2.2 Connect Flags 17](#_Toc71106238)

[3.2.3 Subscribe Flags 17](#_Toc71106239)

[3.2.4 Will Flags 17](#_Toc71106240)

[3.2.5 Packet Id 18](#_Toc71106245)

[3.2.6 Protocol Id 18](#_Toc71106246)

[3.2.7 Radius 18](#_Toc71106247)

[3.2.8 Return Code 18](#_Toc71106248)

[3.2.9 Topic Alias 18](#_Toc71106269)

[3.2.10 Topic Name 19](#_Toc71106270)

[3.2.11 Will Packet 19](#_Toc71106271)

[3.2.12 Will Topic 19](#_Toc71106272)

[3.3 MQTT-SN Topic Types 19](#_Toc71106273)

[4 MQTT-SN Control Packets 20](#_Toc71106274)

[4.1 Format of Individual Packets 20](#_Toc71106275)

[4.1.1 ADVERTISE 20](#_Toc71106276)

[4.1.1.1 Length & Packet Type 20](#_Toc71106277)

[4.1.1.2 GwId 20](#_Toc71106279)

[4.1.1.3 Duration 20](#_Toc71106280)

[4.1.2 SEARCHGW 20](#_Toc71106281)

[4.1.2.1 Length & Packet Type 21](#_Toc71106282)

[4.1.2.2 Radius 21](#_Toc71106284)

[4.1.3 GWINFO 21](#_Toc71106285)

[4.1.3.1 Length & Packet Type 21](#_Toc71106286)

[4.1.3.2 GwId 21](#_Toc71106288)

[4.1.3.3 GwAdd 21](#_Toc71106290)

[4.1.4 CONNECT 21](#_Toc71106291)

[4.1.4.1 Length & Packet Type 22](#_Toc71106292)

[4.1.4.2 Connect Flags 22](#_Toc71106318)

[4.1.4.3 Protocol Version 23](#_Toc71106319)

[4.1.4.4 Keep Alive Timer 23](#_Toc71106320)

[4.1.4.5 Session Expiry Interval 23](#_Toc71106329)

[4.1.4.6 Max Packet Size 24](#_Toc71106330)

[4.1.4.7 Client Identifier 24](#_Toc71106331)

[4.1.5 CONNACK 25](#_Toc71106332)

[4.1.5.1 Length & Packet Type 25](#_Toc71106333)

[4.1.5.2 Return Code 25](#_Toc71106335)

[4.1.5.3 Session Expiry Interval 25](#_Toc71106336)

[4.1.5.4 Client Identifier 25](#_Toc71106337)

[4.1.6 WILLTOPICREQ and WILLPACKETREQ 25](#_Toc71106338)

[4.1.7 WILLTOPIC 26](#_Toc71106339)

[4.1.7.1 Length & Packet Type 26](#_Toc71106340)

[4.1.7.2 Flags 26](#_Toc71106347)

[4.1.7.3 Will Topic 26](#_Toc71106348)

[4.1.8 WILLPACKETREQ 26](#_Toc71106349)

[4.1.9 WILLPACKET 26](#_Toc71106350)

[4.1.9.1 Length & Packet Type 27](#_Toc71106351)

[4.1.9.2 Will Packet 27](#_Toc71106352)

[4.1.10 AUTH 27](#_Toc71106353)

[4.1.10.1 Length & Packet Type 27](#_Toc71106354)

[4.1.10.2 Auth Reason Code 27](#_Toc71106355)

[4.1.10.3 Auth Method Length 28](#_Toc71106356)

[4.1.10.4 Auth Method 28](#_Toc71106357)

[4.1.10.5 Auth Data 28](#_Toc71106358)

[4.1.11 REGISTER 28](#_Toc71106372)

[4.1.11.1 Length & Packet Type 28](#_Toc71106373)

[4.1.11.2 Topic Alias 28](#_Toc71106374)

[4.1.11.3 Packet Id 28](#_Toc71106375)

[4.1.11.4 Topic Name 29](#_Toc71106376)

[4.1.12 REGACK 29](#_Toc71106377)

[4.1.12.1 Length & Packet Type 29](#_Toc71106378)

[4.1.12.2 Regack Flags 29](#_Toc71106379)

[4.1.12.3 Topic Alias 29](#_Toc71106380)

[4.1.12.4 Packet Id 29](#_Toc71106381)

[4.1.12.5 Return Code 29](#_Toc71106382)

[4.1.13 PUBLISH 29](#_Toc71106386)

[4.1.13.1 Length & Packet Type 30](#_Toc71106387)

[4.1.13.2 Flags 30](#_Toc71106388)

[4.1.13.3 Topic Length 30](#_Toc71106396)

[4.1.13.4 Packet Id 30](#_Toc71106397)

[4.1.13.5 Topic Alias or Topic Name 30](#_Toc71106398)

[4.1.13.6 Data 30](#_Toc71106399)

[4.1.14 PUBACK 31](#_Toc71106401)

[4.1.14.1 Length & Packet Type 31](#_Toc71106402)

[4.1.14.2 Packet Id 31](#_Toc71106403)

[4.1.14.3 Return Code 31](#_Toc71106404)

[4.1.15 PUBREC, PUBREL, and PUBCOMP 31](#_Toc71106406)

[4.1.15.1 Length & Packet Type 31](#_Toc71106407)

[4.1.15.2 Packet Id 31](#_Toc71106409)

[4.1.16 SUBSCRIBE 32](#_Toc71106423)

[4.1.16.1 Length & Packet Type 32](#_Toc71106424)

[4.1.16.2 Flags 32](#_Toc71106425)

[4.1.16.3 Packet Id 32](#_Toc71106432)

[4.1.16.4 Topic Alias or Topic Name 32](#_Toc71106433)

[4.1.17 SUBACK 32](#_Toc71106434)

[4.1.17.1 Length & Packet Type 33](#_Toc71106435)

[4.1.17.2 Flags 33](#_Toc71106436)

[4.1.17.3 Topic Alias 33](#_Toc71106440)

[4.1.17.4 Packet Id 33](#_Toc71106441)

[4.1.17.5 Return Code 33](#_Toc71106442)

[4.1.18 UNSUBSCRIBE 33](#_Toc71106443)

[4.1.18.1 Length & Packet Type 34](#_Toc71106444)

[4.1.18.2 Flags 34](#_Toc71106445)

[4.1.18.3 Packet Id 34](#_Toc71106453)

[4.1.18.4 Topic Alias or Topic Name 34](#_Toc71106454)

[4.1.19 UNSUBACK 34](#_Toc71106455)

[4.1.19.1 Length & Packet Type 34](#_Toc71106456)

[4.1.19.2 Packet Id 34](#_Toc71106457)

[4.1.19.3 Return Code 34](#_Toc71106458)

[4.1.20 PINGREQ 34](#_Toc71106459)

[4.1.20.1 Length & Packet Type 35](#_Toc71106460)

[4.1.20.2 Max Messages 35](#_Toc71106461)

[4.1.20.3 Client Identifier 35](#_Toc71106462)

[4.1.21 PINGRESP 35](#_Toc71106463)

[4.1.21.1 Length & Packet Type 35](#_Toc71106464)

[4.1.21.2 Messages Remaining 35](#_Toc71106465)

[4.1.22 DISCONNECT 36](#_Toc71106466)

[4.1.22.1 Length & Packet Type 36](#_Toc71106468)

[4.1.22.2 Return Code 36](#_Toc71106469)

[4.1.22.3 Session Expiry Interval 36](#_Toc71106470)

[4.1.22.4 Reason String 37](#_Toc71106471)

[4.1.23 WILLTOPICUPD 37](#_Toc71106472)

[4.1.23.1 Length & Packet Type 37](#_Toc71106473)

[4.1.23.2 Flags 37](#_Toc71106474)

[4.1.23.3 Will Topic 37](#_Toc71106475)

[4.1.24 WILLPACKETUPD 37](#_Toc71106476)

[4.1.24.1 Length & Packet Type 38](#_Toc71106477)

[4.1.24.2 Will Packet 38](#_Toc71106478)

[4.1.25 WILLTOPICRESP 38](#_Toc71106479)

[4.1.25.1 Length & Packet Type 38](#_Toc71106480)

[4.1.25.2 Return Code 38](#_Toc71106481)

[4.1.26 WILLPACKETRESP 38](#_Toc71106482)

[4.1.26.1 Length & Packet Type 38](#_Toc71106483)

[4.1.26.2 Return Code 38](#_Toc71106485)

[4.2 Forwarder Encapsulation 38](#_Toc71106487)

[4.2.1.1 Length 39](#_Toc71106488)

[4.2.1.2 Packet Type 39](#_Toc71106489)

[4.2.1.3 Ctrl 39](#_Toc71106490)

[4.2.1.4 Wireless Node Id 39](#_Toc71106491)

[4.2.1.5 MQTT SN Packet 39](#_Toc71106492)

[5 Operational behavior 40](#_Toc71106493)

[5.1 Gateway Advertisement and Discovery 40](#_Toc71106494)

[5.2 Client’s Connection Setup 40](#_Toc71106495)

[5.3 Clean start 41](#_Toc71106496)

[5.4 Procedure for updating the Will data 42](#_Toc71106497)

[5.5 Topic Name Registration Procedure 42](#_Toc71106498)

[5.6 Topic Name Mapping and Aliasing 42](#_Toc71106499)

[5.7 Pre-defined topic alias’ and short topic names 43](#_Toc71106507)

[5.8 Client’s Topic Subscribe/Unsubscribe Procedure 43](#_Toc71106514)

[5.9 Client’s Publish Procedure 43](#_Toc71106515)

[5.10 PUBLISH with QoS Level -1 44](#_Toc71106516)

[5.11 Gateway’s Publish Procedure 44](#_Toc71106517)

[5.12 Keep Alive and PING Procedure 45](#_Toc71106518)

[5.13 Client’s Disconnect Procedure 45](#_Toc71106519)

[5.14 Client’s Retransmission Procedure 45](#_Toc71106520)

[5.15 Support of sleeping clients 45](#_Toc71106521)

[6 Authentication 48](#_Toc71106522)

[7 Retained Packets 50](#_Toc71106523)

[8 Safety, Security, and Data Protection Considerations 51](#_Toc71106525)

[9 Conformance 52](#_Toc71106526)

[Appendix A. Acknowledgments 53](#_Toc71106527)

[Appendix B. Implementation Notes 54](#_Toc71106528)

[B.1 Support of QoS Level -1 54](#_Toc71106529)

[B.2 “Best practice” values for timers and counters 54](#_Toc71106530)

[B.3 Mapping of Topic Alias to Topic Names 54](#_Toc71106531)

[Appendix C. Revision History 55](#_Toc71106532)

**Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.**

# Introduction

[All text is normative unless otherwise labeled]

## IPR Policy

This specification is provided under the [Non-Assertion](https://www.oasis-open.org/policies-guidelines/ipr#Non-Assertion-Mode) Mode of the [OASIS IPR Policy](https://www.oasis-open.org/policies-guidelines/ipr), the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (<https://www.oasis-open.org/committees/mqtt/ipr.php>).

## Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#RFC2119)] and [[RFC8174](#RFC8174)] when, and only when, they appear in all capitals, as shown here.

## Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

**[****RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

[Reference] [Full reference citation]

## Non-Normative References

**[****RFC3552]** Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.

[Reference] [Full reference citation]

(**Note**: Any work cited in the body of the text as needed to implement the specification must be listed here. Each reference to a separate document or artifact in this work must be listed here and must be identified as either a Normative or a Non-Normative Reference.

*For all References – Normative and Non-Normative:*

Recommended approach: Set up [Reference] label elements as "Bookmarks", then create hyperlinks to them within the document. (**Here's how:** Insert hyperlinkàPlace in this documentàscroll down to Bookmarks, select appropriate one.)

Use the "Ref term" and "Ref" paragraph styles to format references.

*The proper format for citation of technical work produced by an OASIS TC (whether Standards Track or Non-Standards Track) is:*

[Citation Label] Work Product [title](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html#workProductName) (italicized). Edited by Albert Alston, Bob Ballston, and Calvin Carlson. Approval date (DD Month YYYY). OASIS [Stage](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html#stage) Identifier and [Revision](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html#revision) Number (*e.g.*, OASIS Committee Specification Draft 01). Principal URI ([stage-specific URI](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html#this-version), *e.g*., with stage component: somespec-v1.0-csd01.html). Latest stage: ([latest stage URI](https://docs.oasis-open.org/specGuidelines/ndr/namingDirectives.html#latest-version), without stage identifiers).

For example:

**[OpenDoc-1.2]** *Open Document Format for Office Applications (OpenDocument) Version 1.2*. Edited by Patrick Durusau and Michael Brauer. 19 January 2011. OASIS Committee Specification Draft 07. <https://docs.oasis-open.org/office/v1.2/csd07/OpenDocument-v1.2-csd07.html>. Latest stage: <https://docs.oasis-open.org/office/v1.2/OpenDocument-v1.2.html>.

*Reference sources:*

For references to **IETF RFCs**, use the approved citation formats at:  
<https://docs.oasis-open.org/templates/ietf-rfc-list/ietf-rfc-list.html>.

For references to **W3C Recommendations**, use the approved citation formats at:  
<https://docs.oasis-open.org/templates/w3c-recommendations-list/w3c-recommendations-list.html>.

Remove this note before submitting for publication.)

## Background

There is a recent increase of interest in Wireless Sensor Networks (WSNs), both from commercial and technical point of view, due to their simplicity, low cost and easy deployment. Those networks can serve different purposes, from measurement and detection, to automation and process control. A typical WSN consists of a large number of battery-operated sensors and actuators (SAs), which are usually equipped with a limited amount of storage and processing capabilities. It is important that those devices communicate wirelessly, since the number of SA-nodes is typically very large, and the cost of deployment of a wired infrastructure is prohibitively expensive. Such a network is by nature very dynamic: the wireless links may temporarily break at any time, and nodes may fail and be replaced very often. In such situations the conventional approach of using addresses for communicating with the individual nodes may become a nightmare. Applications residing on the fixed network and requiring interactions with the wireless SA devices would need to manage and maintain means to communicate with a large number of nodes. In most cases they do not need to know the address or identity of the devices which deliver the information but are more interested in the content of the data. For example, an asset tracking application is more interested in the current location of a certain asset than in the network address of the GPS receivers that deliver that information. In addition, several applications may have interest in the same sensor data but for different purposes. In this case the SA nodes would need to manage and maintain communication means with multiple applications in parallel. This might exceed the limited capabilities of the simple and low-cost SA devices.

The problem described above may be overcome by using a data-centric communication approach, in which information is delivered to the receivers not based on their network addresses but rather as a function of their contents and interests. One well-known example of data-centric communication is the “Publish/Subscribe” (pub/sub) messaging system which is already being widely used in enterprise networks, mainly due to their scalability and support of dynamic network topology. Extending the enterprise pub/sub system into the WSNs also enables a seamless integration of the WSNs into the enterprise network, thus making the field data collected by the SAs available to all applications as any other enterprise information and enabling the control of the SAs from any enterprise application. This can be for example achieved by using the MQTT protocol, which is an open and lightweight publish/subscribe protocol designed specifically for machine-to-machine and mobile applications. It is optimized for communications over networks where bandwidth is at a premium or where the network connection could be intermittent. However MQTT requires an underlying network, such as TCP/IP, that provides an ordered lossless connection capability and this is too complex for very simple, small footprint, and low-cost devices such as wireless SAs.

The purpose of this document is to specify MQTT-SN, a pub/sub protocol for wireless sensor networks. MQTT-SN can be considered as a version of MQTT which is adapted to the peculiarities of a wireless communication environment. Wireless radio links have in general a higher failure rates than wired ones due to their susceptibility to fading and interference disturbances. They have also a lower transmission rate. For example, WSNs based on the IEEE 802.15.4 standard provide a maximum bandwidth of 250 kbit/s in the 2.4 GHz band. Moreover, to be resistant against transmission errors, their packets have a very short length. In the case of IEEE 802.15.4, the packet length at the physical layer is limited to 128 bytes. Half of these 128 bytes could be taken away by the overhead information required by supporting functions such as MAC layer, networking, security, etc.

MQTT-SN is also optimized for implementation on low-cost, battery-operated devices with limited processing and storage resources.

MQTT-SN is designed in such a way that it is agnostic of the underlying networking services. Any network which provides a bi-directional data transfer service between any node and a particular one (a gateway) should be able to support MQTT-SN. For example a simple datagram service which allows a source endpoint to send a data packet to a specific destination endpoint should be sufficient. A broadcast data transfer service is only required if the gateway discovery procedure is employed. To reduce the broadcast traffic created by the discovery procedure, it is desirable that MQTT-SN could indicate the required broadcast radius to the underlying layer.

## MQTT-SN comparison with MQTT

MQTT-SN is designed to be as close as possible to MQTT, but is adapted to the peculiarities of a wireless communication environment such as low bandwidth, high link failures, short packet length, etc. It is also optimized for the implementation on low-cost, battery-operated devices with limited processing and storage resources. Compared to MQTT, MQTT-SN is characterized by the following differences:

1. The CONNECT packet is split into three packets. The two additional ones are optional and used to transfer the Will topic and the Will packet to the server.
2. To cope with the short packet length and the limited transmission bandwidth in wireless networks, the topic name in the PUBLISH packets is replaced by a short, two-byte long “topic alias”. A registration procedure is defined to allow clients to register their topic names with the server/gateway and obtain the corresponding topic alias’. It is also used in the opposite direction to inform the client about the topic name and the corresponding topic alias that will be included in a following PUBLISH packet .
3. “Pre-defined” topic alias’ and “short” topic names are introduced, for which no registration is required. Predefined alias’ are also a two-byte long replacement of the topic name, their mapping to the topic names is however known in advance by both the client’s application and the gateway/server. Therefore both sides can start using pre-defined topic alias’; there is no need for a registration as in the case of “normal” topic alias’ mentioned above.  
     
   Short topic names are topic names that have a fixed length of two bytes. They are short enough for being carried together with the data within PUBLISH packets. As for pre-defined topic alias’, there is also no need for a registration for short topic names.
4. A discovery procedure helps clients without a pre-configured server/gateway’s address to discover the actual network address of an operating server/gateway. Multiple gateways may be present at the same time within a single wireless network and can co-operate in a load-sharing or stand-by mode.
5. The semantic of a “clean session” is extended to the Will feature, i.e. not only client’s subscriptions are persistent, but also Will topic and Will packet. A client can also modify its Will topic and Will packet during a session.
6. A new offline keep-alive procedure is defined for the support of *sleeping* clients. With this procedure, battery-operated devices can go to a sleeping state during which all packets destined to them are buffered at the server/gateway and delivered later to them when they wake up.

## MQTT-SN Architecture

A picture containing text, map

Description automatically generated

Figure 1: MQTT-SN Architecture

The architecture of MQTT-SN is shown in Fig. 1. There are three kinds of MQTT-SN components, MQTT-SN *clients*, MQTT-SN *gateways (GW)*, and MQTT-SN *forwarders*. MQTT-SN clients connect themselves to a MQTT server via a MQTT-SN GW using the MQTT-SN protocol. A MQTT-SN GW may or may not be integrated with a MQTT server. In case of a stand-alone GW the MQTT protocol is used between the MQTT server and the MQTT-SN GW. Its main function is the translation between MQTT and MQTT-SN.

MQTT-SN clients can also access a GW via a forwarder in case the GW is not directly attached to their network. The forwarder simply encapsulates the MQTT-SN frames it receives on the wireless side and forwards them unchanged to the GW; in the opposite direction, it decapsulates the frames it receives from the gateway and sends them to the clients, unchanged too.

Depending on how a GW performs the protocol translation between MQTT and MQTT-SN, we can differentiate between two types of GWs, namely *transparent* and *aggregating* GWs, see Fig. 2. They are explained in the following sections.

### Transparent Gateway

For each connected MQTT-SN client a transparent GW will setup and maintain a MQTT connection to the MQTT server. This MQTT connection is reserved exclusively for the end-to-end and almost transparent packet exchange between the client and the server. There will be as many MQTT connections between the GW and the server as MQTT-SN clients connected to the GW. The transparent GW will perform a “syntax” translation between the two protocols. Since all packet exchanges are end-to-end between the MQTT-SN client and the MQTT server, all functions and features that are implemented by the server can be offered to the client.

Although the implementation of the transparent GW is simpler when compared to the one of an aggregating GW, it requires the MQTT server to support a separate connection for each active client. Some MQTT server implementations might impose a limitation on the number of concurrent connections that they support.

A close up of a map

Description automatically generated

Figure 2: Transparent and Aggregating Gateways

### Aggregating Gateway

Instead of having a MQTT connection for each connected client, an aggregating GW will have only one MQTT connection to the server. All packet exchanges between a MQTT-SN client and an aggregating GW end at the GW. The GW then decides which information will be given further to the server. Although its implementation is more complex than the one of a transparent GW, an aggregating GW may be helpful in case of WSNs with very large number of SAs because it reduces the number of MQTT connections that the server has to support concurrently.

# Data representation

## Bits (Byte)

Bits in a byte are labelled 7 to 0. Bit number 7 is the most significant bit, the least significant bit is assigned bit number 0.

## Two Byte Integer

Two Byte Integer data values are 16-bit unsigned integers in big-endian order: the high order byte precedes the lower order byte. This means that a 16-bit word is presented on the network as Most Significant Byte (MSB), followed by Least Significant Byte (LSB).

## Four Byte Integer

Four Byte Integer data values are 32-bit unsigned integers in big-endian order: the high order byte precedes the successively lower order bytes. This means that a 32-bit word is presented on the network as Most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by Least Significant Byte (LSB).

## UTF-8 Encoded String

Text fields within the MQTT-SN Control Packets described later are encoded as UTF-8 strings. UTF-8 [[RFC3629]](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#RFC3629) is an efficient encoding of Unicode [[Unicode]](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#Unicode) characters that optimizes the encoding of ASCII characters in support of text-based communications.

Each of these strings is prefixed with a Two Byte Integer length field that gives the number of bytes in a UTF-8 encoded string itself, as illustrated in Figure 1.1 Structure of UTF-8 Encoded Strings below. Consequently, the maximum size of a UTF-8 Encoded String is 65,535 bytes.

Unless stated otherwise all UTF-8 encoded strings can have any length in the range 0 to 65,535 bytes.

Figure 1‑1 Structure of UTF-8 Encoded Strings

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | String length MSB | | | | | | | |
| byte 2 | String length LSB | | | | | | | |
| byte 3 …. | UTF-8 encoded character data, if length > 0. | | | | | | | |

The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode specification [[Unicode]](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#Unicode) and restated in RFC 3629 [[RFC3629]](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#RFC3629). In particular, the character data MUST NOT include encodings of code points between U+D800 and U+DFFF [MQTT-1.5.4-1]. If the Client or Server receives an MQTT Control Packet containing ill-formed UTF-8 it is a Malformed Packet. Refer to [section 4.13](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#S4_13_Errors) for information about handling errors.

A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000. [MQTT-1.5.4-2]. If a receiver (Server or Client) receives an MQTT Control Packet containing U+0000 it is a Malformed Packet. Refer to[section 4.13](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#S4_13_Errors) for information about handling errors.

The data SHOULD NOT include encodings of the Unicode [Unicode] code points listed below. If a receiver (Server or Client) receives an MQTT Control Packet containing any of them it MAY treat it as a Malformed Packet. These are the Disallowed Unicode code points. Refer to [section](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#S4_13_Errors)5.4.9 for more information about handling Disallowed Unicode code points.

         U+0001..U+001F control characters

         U+007F..U+009F control characters

         Code points defined in the Unicode specification [[Unicode]](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#Unicode) to be non-characters (for example U+0FFFF)

A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always interpreted as U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver [MQTT-1.5.4-3].

# MQTT-SN Control Packet format

## Structure of an MQTT-SN Control Packet

The MQTT-SN protocol operates by exchanging a series of MQTT-SN Control Packets in a defined way. This section describes the format of these packets.

An MQTT-SN Control Packet consists of up to two parts, always in the following order as shown below.

Figure ‑ Structure of an MQTT-SN Control Packet

|  |
| --- |
| Control Packet Header, present in all MQTT-SN Control Packets |
| Control Packet Variable Part, present in some MQTT-SN Control Packets |

### Packet Header

Each MQTT-SN Control Packet contains a Header of format1 or format2 as shown below.

Figure ‑ Header format1

|  |  |
| --- | --- |
| **Byte** | **Use** |
| 1 | Length |
| 2 | MQTT-SN Control Packet Type |

Figure ‑ Header format2

|  |  |
| --- | --- |
| **Byte** | **Use** |
| 1 | Length 0x01 |
| 2 | Length MSB |
| 3 | Length LSB |
| 4 | MQTT-SN Control Packet Type |

### Length

The *Length* field is either 1-byte or 3-byte integer and specifies the total number of bytes contained in the packet (including the *Length* field itself).

If the first byte of the *Length* field is coded “0x01” then the *Length* field is 3-bytes long; in this case, the two following bytes specify the total number of bytes of the packet (most-significant byte first). Otherwise, the *Length* field is only 1-byte long and specifies itself the total number of bytes contained in the packet.

The 3-byte format allows the encoding of packet lengths up to 65535 bytes. Packets with lengths up to and including 255 bytes MUST use the shorter byte format.

**Non-normative comment**

MQTT-SN does not support packet fragmentation and reassembly, the maximum packet length that could be used in a network is governed by the maximum packet size that is supported by that network, and not by the maximum length that could be encoded by MQTT-SN.

### MQTT-SN Control Packet Type

The MQTT-SN Control Packet Typefield is 1-byte long and specifies the MQTT-SN Control Packet type which is one of the values shown below.

Table 2‑1 MQTT Control Packet types

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Value** | **Direction of flow** | **Description** |
| ADVERTISE | 0x00 | Gateway broadcast | Advertise the gateway presence |
| SEARCHGW | 0x01 | Client broadcast | Client GWINFO request |
| GWINFO | 0x02 | Gateway to Client | Response to a SEARCHGW |
| AUTH | 0x03 | Client to Gateway or Gateway to Client | Authentication handshake |
| CONNECT | 0x04 | Client to Gateway | Connection request |
| CONNACK | 0x05 | Gateway to Client | Connection acknowledgement |
| WILLTOPICREQ | 0x06 | Gateway to Client | Request the will topic name |
| WILLTOPIC | 0x07 | Client to Gateway | Supply the will topic name |
| WILLPACKETREQ | 0x08 | Gateway to Client | Request the will packet |
| WILLPACKET | 0x09 | Client to Gateway | Supply the will packet |
| REGISTER | 0x0A | Client to Gateway | Request topic alias |
| REGACK | 0x0B | Gateway to Client | Supply topic alias |
| PUBLISH | 0x0C | Client to Gateway or  Gateway to Client | Publish packet |
| PUBACK | 0x0D | Client to Gateway or  Gateway to Client | Publish acknowledgment (QoS 1) |
| PUBCOMP | 0x0E | Client to Gateway or  Gateway to Client | Publish complete (QoS 2 delivery part 3) |
| PUBREC | 0x0F | Client to Gateway or  Gateway to Client | Publish received (QoS 2 delivery part 1) |
| PUBREL | 0x01 | Client to Gateway or  Gateway to Client | Publish release (QoS 2 delivery part 2) |
| reserved | 0x11 |  | Forbidden |
| SUBSCRIBE | 0x12 | Client to Gateway | Subscribe request |
| SUBACK | 0x13 | Gateway to Client | Subscribe acknowledgment |
| UNSUBSCRIBE | 0x14 | Client to Gateway | Unsubscribe request |
| UNSUBACK | 0x15 | Gateway to Client | Unsubscribe acknowledgment |
| PINGREQ | 0x16 | Client to Gateway | PING request |
| PINGRESP | 0x17 | Gateway to Client | PING response |
| DISCONNECT | 0x18 | Client to Gateway or  Gateway to Client | Disconnect notification |
| reserved | 0x19 |  | Forbidden |
| WILLTOPICUPD | 0x1A | Client to Gateway | Modify the will topic name |
| WILLTOPICRESP | 0x1B | Gateway to Client | Acknowledge the will topic name modification |
| WILLPACKETUPD | 0x1C | Client to Gateway | Modify the will packet |
| WILLPACKETRESP | 0x1D | Gateway to Client | Acknowledge the will packet modification |
| reserved | 0x1E-0xFD |  | Forbidden |
| Encapsulated packet | 0xFE | Forwarder to Client or  Forwarder to Gateway | Encapsulated MQTT-SN packet |
| reserved | 0xFF |  | Forbidden |



## MQTT-SN Control Packet Variable Part

The content of the MQTT-SN Control Packet variable part depends on the type of the Control Packet. The following fields are defined for the packet variable part.

### Publish Flags

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | DUP | QoS | | Retain | *Reserved* | *Reserved* | Topic Alias Type | |

Figure 3.2.1: Publish Flags



The Publish Flags field is 1-byte and contains the following flags (see Table 4):

* **DUP**: same meaning as with MQTT, i.e. set to “0” if packet is sent for the first time; set to “1” if retransmitted (only relevant within PUBLISH packets);
* **QoS**: meaning as with MQTT for QoS level 0, 1, and 2; set to “0b00” for QoS level 0, “0b01” for QoS level 1, “0b10” for QoS level 2, and “0b11” for new QoS level -1 (only relevant within PUBLISH packets sent by a client);
* **Retain**: same meaning as with MQTT
* **Topic Alias Type**: Refer to sections 3.3 for the definition of the various types of topic.

### Connect Flags

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | *Reserved* | *Reserved* | *Reserved* | *Reserved* | Capabilities | Authentication | Will | Clean Start |

Figure 3.2.2: Connect Flags

The Connect *Flags* field is 1-byte and contains the following flags (see Table 5):

* **Capabilities**: indicates capability exchange to follow
* **Authentication**: indicates authentication exchange to follow
* **Will**: if set, indicates that client is asking for Will topic and Will message prompting
* **Clean Start**: same meaning as with MQTT, however extended for Will topic and Will packet

### Subscribe Flags

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | No Local | QoS | | Retain as Published | Retain Handling | | Topic Type | |

Figure 3.2.3: Subscribe Flags

The Subscribe Flags field is 1-byte and contains the following flags (see Table 6):

* **QoS**: maximum QoS. This gives the maximum QoS level at which the Server can send Application Messages to the Client. It is a Protocol Error if the Maximum QoS field has the value 3.
* **No Local**: if the value is 1, Application Messages MUST NOT be forwarded to a connection with a ClientID equal to the ClientID of the publishing connection
* **Retain as published**: If 1, Application Messages forwarded using this subscription keep the RETAIN flag they were published with. If 0, Application Messages forwarded using this subscription have the RETAIN flag set to 0. Retained messages sent when the subscription is established have the RETAIN flag set to 1.
* **Retain handling**: This option specifies whether retained messages are sent when the subscription is established. This does not affect the sending of retained messages at any point after the subscribe. If there are no retained messages matching the Topic Filter, all of these values act the same. The values are:
  + 0: Send retained messages at the time of the subscribe
  + 1: Send retained messages at subscribe only if the subscription does not currently exist
  + 2: Do not send retained messages at the time of the subscribe.

It is a Protocol Error to send a Retain Handling value of 3.

* **Topic Type**: indicates the type of Topic Alias or Topic Name included in this. Refer to section 3.3 for the definition of the various types.

### Will Flags

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|  | *Reserved* | Will QoS | | Retain | *Reserved* | *Reserved* | *Reserved* | | *Reserved* |

Figure 3.2.4: Will Flags

* **Will QoS**: same as MQTT, contains the Will QoS
* **Retain**: same as MQTT, contains the Will Retain flag

### Packet Id

The *Packet Id* field is 2-byte long and corresponds to the MQTT ‘Packet ID’ parameter. It allows the sender to match a packet with its corresponding acknowledgment.

### Protocol Id

The *Protocol Id* is 1-byte long. It is only present in a CONNECT packet and corresponds to the MQTT ‘protocol name’ and ‘protocol version’.

It is coded 0x02. 0x01 was used for MQTT-SN 1.2. All other values are reserved.

### Radius

The *Radius* field is 1-byte long and indicates the value of the broadcast radius. The value 0x00 means “broadcast to all nodes in the network”.

### Return Code

The value and meaning of the 1-byte long *Return Code* field is shown in Table 5.

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | | Description | Packet |
| Dec | Hex |
| 0 | 0x00 | Accepted | CONNACK, SUBACK, UNSUBACK, REGACK, PUBACK, WILLTOPICRESP, WILLPACKETRESP |
| 1 | 0x01 | Rejected: congestion | CONNACK, SUBACK, UNSUBACK, REGACK, PUBACK, WILLTOPICRESP, WILLPACKETRESP |
| 2 | 0x02 | Rejected: invalid topic alias | SUBACK, UNSUBACK, REGACK, PUBACK, WILLTOPICRESP, WILLPACKETRESP |
| 3 | 0x03 | Rejected: not supported | CONNACK, SUBACK, UNSUBACK, REGACK, PUBACK, WILLTOPICRESP, WILLPACKETRESP |
| 4 | 0x04 | Rejected: packet size too large | DISCONNECT |
| 140 | Ox8C | Bad authentication method | AUTH |
| 135 | Ox87 | Not authorized | AUTH |

Table 5: Return Code Values



### Topic Alias

The *Topic Alias* field is 2-byte long and contains the value of the topic alias. The values “0x0000” and “0xFFFF” are reserved and therefore should not be used.

### Topic Name

The *Topic Name* field has a variable length and contains an UTF8-encoded string that specifies the topic name.

### Will Packet

The *Will Packet* field has a variable length and contains the Will packet .

### Will Topic

The *Will Topic* field has a variable length and contains the Will topic name.

## MQTT-SN Topic Types

A number of packets will refer to a topic id types in their flags. This is a 2-bit field which determines the format of the topicId value. The allowable values are as follows:

|  |  |  |
| --- | --- | --- |
| **Topic Alias Type Value** | **Name** | **Description** |
| 0b00 | Normal Topic Alias | A normal topic alias is negotiated between the gateway and client within the scope of a gateway session. |
| 0b01 | Predefined Topic Alias | A predefined alias is known statically by both the gateway and the client outside the scope of a gateway session. No negotiation is required since both entities have knowledge of the topic alias mapping. |
| 0b10 | Short Topic Name | A 2-byte topic name which requires no negotiation. |
| 0b11 | Long Topic Name | A full topic encode, which requires no session negotiation. |

Please refer to section 5.6 and 5.7 for detailed descriptions of topic types and aliases.

# MQTT-SN Control Packets

## Format of Individual Packets

This section specifies the format of the individual MQTT-SN packets. All packets are described with the 1-byte *Length* field. The packet formats in case of the 3-byte *Length* field could be derived straightforwardly and are therefore not mentioned.

### ADVERTISE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | GwId | | | | | | | |
| Byte 4 | Duration MSB | | | | | | | |
| Byte 5 | Duration LSB | | | | | | | |



Table 6: ADVERTISE Packet

The ADVERTISE packet is broadcast periodically by the gateway to advertise its presence. The time interval until the next broadcast is indicated by the *Duration* field . Its format is illustrated in Table 6:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### GwId

The *GwId* field is 1-byte long and uniquely identifies a gateway which is advertising itself in the network

#### Duration

The *Duration* field is a 2-byte long. It specifies the time interval in seconds until the next ADVERTISE packet is broadcasted by this gateway period. The maximum value that can be encoded is approximately 18 hours

### SEARCHGW

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Radius | | | | | | | |



Table 7: SEARCHGW packet

The SEARCHGW packet is broadcasted by a client when it searches for a GW. The broadcast radius of the SEARCHGW is limited and depends on the density of the clients deployment, e.g. only 1-hop broadcast in case of a very dense network in which every MQTT-SN client is reachable from each other within 1-hop transmission. The format of a SEARCHGW packet is illustrated in Table 7:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Radius

The broadcast radius is also indicated to the underlying network layer when MQTT-SN gives this packet for transmission.

### GWINFO

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | GwId | | | | | | | |
| Byte 4 … N | GwAddress *(optional)* | | | | | | | |



Table 8: GWINFO packet

The GWINFO packet is sent as response to a SEARCHGW packet using the broadcast service of the underlying layer, with the radius as indicated in the SEARCHGW packet. If sent by a GW, it contains only the id of the sending GW; otherwise, if sent by a client, it also includes the address of the GW, see Table 8:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 1.2.1 for a detailed description.

#### GwId

#### GwAdd

The *GwAdd* field has a variable length and contains the address of a Gateway. Its length depends on the type of network over which MQTT-SN operates and is specified by the Length byte. Optional, only included if packet is sent by a client.

### CONNECT



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | **2** | **1** | | **0** | |
| Byte 1 | Length | | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | | |
|  | *Reserved* | | | | | | | Reserved | Authentication | | Will | | Clean Start |
| Byte 3 | *0* | | *0* | *0* | *0* | | | *0* | *X* | | *X* | | *X* |
| Byte 4 | Protocol Version | | | | | | | | | | | | |
| Byte 5 | Keep Alive MSB | | | | | | | | | | | | |
| Byte 6 | Keep Alive LSB | | | | | | | | | | | | |
| Byte 7 | Session Expiry Interval MSB | | | | | | | | | | | | |
| Byte 8 | Session Expiry Interval | | | | | | | | | | | | |
| Byte 9 | Session Expiry Interval | | | | | | | | | | | | |
| Byte 10 | Session Expiry Interval LSB | | | | | | | | | | | | |
| Byte 11 | Max Packet Size MSB | | | | | | | | | | | | |
| Byte 12 | Max Packet Size LSB | | | | | | | | | | | | |
| Byte 13 . N | Client Identifier | | | | | | | | | | | | |

After a Network Connection is established by a Client to a Server, the first packet sent from the Client to the Server MUST be a CONNECT packet [MQTT-3.1.0-1].

A Client can only send the CONNECT packet once over a Network Connection. The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection [MQTT-3.1.0-2]. Refer to [section 4.13](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html" \l "S4_13_Errors) for information about handling errors. The CONNECT packet is sent from the Client to the server to setup a connection.

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.



#### Connect Flags

The Connect Flags byte contains several parameters specifying the behavior of the MQTT-SN connection. It also indicates the presence or absence of fields in the Payload. For a detailed breakdown of the connect flags please refer to figure 3.2.2.



The Server MUST validate that the reserved flags in the CONNECT packet are set to 0 [MQTT-3.1.2-3]. If any of the reserved flags is not 0 it is a Malformed Packet. Refer to [section 4.13](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html" \l "S4_13_Errors) for information about handling errors.

#### Protocol Version

The one-byte unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Version field for MQTT-SN version 2.0 is 2 (0x02).

A Server which supports multiple versions of the MQTT-SN protocol uses the Protocol Version to determine which version of MQTT-SN the Client is using. If the Protocol Version is not 2 and the Server does not want to accept the CONNECT packet, the Server MAY send a CONNACK packet with Reason Code 0x84 (Unsupported Protocol Version) and then MUST close the Network Connection.

#### Keep Alive Timer

The Keep Alive is a Two Byte Integer which is a time interval measured in seconds. It is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one MQTT-SN Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between MQTT Control Packets being sent does not exceed the Keep Alive value. If Keep Alive is non-zero and in the absence of sending any other MQTT-SN Control Packets, the Client MUST send a PINGREQ packet [MQTT-3.1.2-20].

If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value instead of the value it sent as the Keep Alive [MQTT-3.1.2-21].

The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and check for a corresponding PINGRESP to determine that the network and the Server are available.

If the Keep Alive value is non-zero and the Server does not receive an MQTT-SN Control Packet from the Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to the Client as if the network had failed [MQTT-3.1.2-22].

If a Client does not receive a PINGRESP packet within a reasonable amount of time after it has sent a PINGREQ, it SHOULD close the Network Connection to the Server.

A Keep Alive value of 0 has the effect of turning off the Keep Alive mechanism. If Keep Alive is 0 the Client is not obliged to send MQTT-SN Control Packets on any particular schedule.

**Non-normative comment**  
The Server may have other reasons to disconnect the Client, for instance because it is shutting down. Setting Keep Alive does not guarantee that the Client will remain connected.

**Non-normative comment**

The actual value of the Keep Alive is application specific; typically, this is a few minutes. The maximum value of 65,535 is 18 hours 12 minutes and 15 seconds.

#### Session Expiry Interval

If the Session Expiry Interval is absent the value 0 is used. If it is set to 0, the Session ends when the Network Connection is closed.

If the Session Expiry Interval is 0xFFFFFFFF (UINT\_MAX), the Session does not expire.

The Client and Server MUST store the Session State after the Network Connection is closed if the Session Expiry Interval is greater than 0.

**Non-normative comment**

The clock in the Client or Server may not be running for part of the time interval, for instance because the Client or Server are not running. This might cause the deletion of the state to be delayed.

When the Session expires the Client and Server need not process the deletion of state atomically.

#### Max Packet Size

A Two Byte (16-bit) Integer representing the Maximum Packet Size the Client is willing to accept. If the Maximum Packet Size is set to 0, no limit on the packet size is imposed beyond the limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

**Non-normative comment**

It is the responsibility of the application to select a suitable Maximum Packet Size value if it chooses to restrict the Maximum Packet Size.

The packet size is the total number of bytes in an MQTT Control Packet, as defined in section 3.1. The Client uses the Maximum Packet Size to inform the Server that it will not process packets exceeding this limit.

The Server MUST NOT send packets exceeding Maximum Packet Size to the Client [MQTT-3.1.2-24]. If a Client receives a packet whose size exceeds this limit, this is a Protocol Error, the Client uses DISCONNECT with Reason Code 0x95 (Packet too large), as described in section 3.2.8.

Where a Packet is too large to send, the Server MUST discard it without sending it and then behave as if it had completed sending that Application Message [MQTT-3.1.2-25].

**Non-normative comment**

Where a packet is discarded without being sent, the Server could place the discarded packet on a ‘dead letter queue’ or perform other diagnostic action. Such actions are outside the scope of this specification.

#### Client Identifier

The Client Identifier (ClientID) identifies the Client to the Server. Each Client connecting to the Server has a unique ClientID. The ClientID MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT-SN Session between the Client and the Server [MQTT-3.1.3-2]. Refer to [section xxx4.1](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html" \l "_Session_State) for more information about Session State.

The ClientID MUST be a UTF-8 Encoded String as defined in [section 1.5.4](https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html#_UTF-8_Encoded_String) [MQTT-3.1.3-4]. between 1 and 23 UTF-8 encoded bytes in length.

### CONNACK

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Return Code | | | | | | | |
| Byte 4 | Session Expiry Interval MSB | | | | | | | |
| Byte 5 | Session Expiry Interval | | | | | | | |
| Byte 6 | Session Expiry Interval | | | | | | | |
| Byte 7 | Session Expiry Interval LSB | | | | | | | |
| Byte 8 … N | Client Identifier (optional) | | | | | | | |

Table 10: CONNACK packet

The CONNACK packet is sent by the server in response to a connection request from a client. Its format is shown in Table 10:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Return Code

“Accepted” or rejected reason.

#### Session Expiry Interval

If the Session Expiry Interval is 0, the value in the CONNECT Packet used. The server uses this property to inform the Client that it is using a value other than that sent by the Client in the CONNACK. Refer to section 4.1.4.5 for a description of the use of Session Expiry Interval.

#### Client Identifier

The Client Identifier assigned by the gateway when the associated CONNECT packet contained no Client Identifier.

For more details of the connection procedure, refer to section 6.2.

### WILLTOPICREQ and WILLPACKETREQ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |



Table 11: WILLTOPICREQ and WILLPACKETREQ packet

The WILLTOPICREQ packet is sent by the GW to request a client for sending the Will topic name. Its format is shown in Table 11: it has only a header and no variable part.

### WILLTOPIC

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | **2** | **1** | | **0** |
| Byte 1 | Length | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | |
|  | *Reserved* | | *Will QoS* | | *Retain* | | | *Reserved* | *Reserved* | | *Reserved* | *Reserved* |
| Byte 3 | *0* | | *X* | *X* | *X* | | | *0* | *0* | | *0* | *0* |
| Byte 4.. N | Will Topic | | | | | | | | | | | |

Table 12: WILLTOPIC packet

The WILLTOPIC packet is sent by a client as response to the WILLTOPICREQ packet for transferring its Will topic name to the GW. Its format is shown in Table 12:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the will flags field please refer to figure 3.2.4.

#### Will Topic

Contains the Will topic name.

An empty WILLTOPIC packet is a WILLTOPIC packet without Flags and WillTopic field (i.e. it is exactly 2 bytes long). It is used by a client to delete the Will topic and the Will packet stored in the server, see Section 6.4.

### WILLPACKETREQ

The WILLPACKETREQ packet is sent by the GW to request a client for sending the Will packet. Its format is shown in Table 11: it has only a header and no variable part.

### WILLPACKET

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 .. N | Will Packet | | | | | | | |

Table 13: WILLPACKET packet

The WILLPACKET packet is sent by a client as response to a WILLPACKETREQ for transferring its Will packet to the GW. Its format is shown in Table 13:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Will Packet

Contains the will packet

### AUTH

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Auth Reason Code | | | | | | | |
| Byte 4 | Auth Method Length (K) | | | | | | | |
| Byte 5:5+K | Auth Method | | | | | | | |
| Byte 6+K:N | Auth Data (N) | | | | | | | |

Table 14: AUTH packet

The AUTH message is first sent by the client as part of an authentication exchange.  The server responds with another AUTH message and so on until the authentication is complete.  The server then responds with a CONNACK message.

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Auth Reason Code

The sender of the AUTH Packet MUST use one of the Authenticate Reason Codes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | | **Reason Code Name** | **Sent by** | **Description** |
| **Dec** | **Hex** |
| 0 | 0x00 | Success | Gateway | Authentication is successful |
| 24 | 0x18 | Continue authentication | Client or Server | Continue the authentication with another step |
| 25 | 0x19 | Re-authenticate | Client | Initiate another authentication |

#### Auth Method Length

The length of the auth method string.

#### Auth Method

A UTF-8 Encoded String containing the name of the authentication method.

#### Auth Data

Binary Data containing authentication data. The contents of this data are defined by the authentication method.

**Non-normative comment**

For a simple cleartext password analogous to MQTT user name and password, the SASL PLAIN method can be used.



### REGISTER

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Topic Alias MSB | | | | | | | |
| Byte 4 | Topic Alias LSB | | | | | | | |
| Byte 5 | Packet IdMSB | | | | | | | |
| Byte 6 | Packet Id LSB | | | | | | | |
| Byte 7 … N | Topic Name | | | | | | | |

Table 14: REGISTER packet

The REGISTER packet is sent by a client to a GW for requesting a topic alias value for the included topic name. It is also sent by a GW to inform a client about the topic alias value it has assigned to the included topic name. Its format is illustrated in Table 14:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Topic Alias

If sent by a client, it is coded 0x0000 and is not relevant; if sent by a GW, it contains the topic alias value assigned to the topic name included in the Topic Name field;

#### Packet Id

Should be coded such that it can be used to identify the corresponding REGACK packet.

#### Topic Name

Contains the fully qualified topic name.

### REGACK

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | | **6** | **5** | **4** | **3** | **2** | | **1** | | **0** |
| Byte 1 | Length | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | |
|  | Reserved | | Reserved | Reserved | Reserved | Reserved | | Reserved | Topic Type | | |
| Byte 3 | 0 | 0 | | 0 | 0 | 0 | | 0 | X | X | |
| Byte 4 | Topic Alias MSB | | | | | | | | | | |
| Byte 5 | Topic Alias LSB | | | | | | | | | | |
| Byte 6 | Packet Id MSB | | | | | | | | | | |
| Byte 7 | Packet Id LSB | | | | | | | | | | |
| Byte 8 | Return Code | | | | | | | | | | |

Table 15: REGACK packet

The REGACK packet is sent by a client or by a GW as an acknowledgment to the receipt and processing of a REGISTER packet. Its format is illustrated in Table 15:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Regack Flags

* **Topic Type**: indicates the type of Topic Alias or Topic Name included in this. Refer to section 3.3 for the definition of the various types.

#### Topic Alias

The value that shall be used as the topic alias in the PUBLISH packets.

#### Packet Id

The same value as the one contained in the corresponding REGISTER packet.

#### Return Code

“Accepted” or rejected reason.

### PUBLISH

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | | **2** | **1** | | **0** | |
| Byte 1 | Length | | | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | | | |
|  | *DUP* | | *QoS* | | *Retain* | | | *Reserved* | | *Reserved* | | *Topic Type* | | |
| Byte 3 | *X* | | *X* | *X* | *X* | | | *0* | | *0* | | *X* | | *X* |
| Byte 4 | Topic Length MSB (TL) | | | | | | | | | | | | | |
| Byte 5 | Topic Length LSB (TL) | | | | | | | | | | | | | |
| Byte 6 | Packet Id MSB | | | | | | | | | | | | | |
| Byte 7 | Packet Id LSB | | | | | | | | | | | | | |
| Byte 8 | Topic Alias MSB | | | | | | | | Topic Name  Byte 8 … (8 + TL) | | | | | |
| Byte 9 | Topic Alias LSB | | | | | | | |
| Byte (10 + TL - 2) .. N | Data | | | | | | | | | | | | | |

Table 16: PUBLISH packet

This packet is used by both clients and gateways to publish data for a certain topic. Its format is illustrated in Table 16:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the publish flags please refer to figure 3.2.1.

#### Topic Length

Contains the length of the topic value. This will be either 2 bytes when using a standard alias (0b00, 0b01 or 0b10) or the length of the full topic name when the topic type is 0b11.

#### Packet Id

Same meaning as the MQTT “Packet ID”; only relevant in case of QoS levels 1 and 2, otherwise coded 0x0000.

#### Topic Alias or Topic Name

Contains topic name, topic alias, or short topic name as indicated in the *Topic Type* field in flags.

#### Data

The *Data* field corresponds to payload of an MQTT PUBLISH packet. It has a variable length and contains the application data that is being published.

### PUBACK

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Packet Id MSB | | | | | | | |
| Byte 4 | Packet Id LSB | | | | | | | |
| Byte 5 | Return Code | | | | | | | |

Table 17: PUBACK packet

The PUBACK packet is sent by a gateway or a client as an acknowledgment to the receipt and processing of a PUBLISH packet in case of QoS levels 1 or 2. It can also be sent as response to a PUBLISH packet in case of an error; the error reason is then indicated in the *ReturnCode* field. Its format is illustrated in Table 17:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Packet Id

Same value as the one contained in the corresponding PUBLISH packet.

#### Return Code

“Accepted” or rejected reason.

### PUBREC, PUBREL, and PUBCOMP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Packet Id MSB | | | | | | | |
| Byte 4 | Packet Id LSB | | | | | | | |

Table 18: PUBREC, PUBREL, and PUBCOMP packets

As with MQTT, the PUBREC, PUBREL, and PUBCOMP packets are used in conjunction with a PUBLISH packet with QoS level 2. Their format is illustrated in Table 18:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 1.2.1 for a detailed description.

#### Packet Id

Same value as the one contained in the corresponding PUBLISH packet.



### SUBSCRIBE

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | | **3** | **2** | **1** | **0** | |
| Byte 1 | Length | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | |
|  | *No Local* | | *QoS* | | *Retain as published* | | | *Retain Handling* | | *Topic Type* | | |
| Byte 3 | *X* | | *X* | *X* | *X* | | | *X* | *X* | *X* | | *X* |
| Byte 4 | Packet Id MSB | | | | | | | | | | | |
| Byte 5 | Packet Id LSB | | | | | | | | | | | |
| Byte 6 | Topic Alias MSB | | | | | | Topic Name  Byte 6 … N | | | | | |
| Byte 7 | Topic Alias LSB | | | | | |

Table 19: SUBSCRIBE packet

The SUBSCRIBE packet is used by a client to subscribe to a certain topic name. Its format is illustrated in Table 19:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the publish flags please refer to figure 3.2.3.

#### Packet Id

Should be coded such that it can be used to identify the corresponding SUBACK packet.

#### Topic Alias or Topic Name

Contains topic name, topic alias, or short topic name as indicated in the *Topic Alias Type* field in flags.

### SUBACK

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | **2** | **1** | | **0** | |
| Byte 1 | Length | | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | | |
|  | *Reserved* | | *Granted QoS* | | *Reserved* | | | *Reserved* | *Reserved* | | *Topic Type* | | |
| Byte 3 | *0* | | *X* | *X* | *0* | | | *0* | *0* | | *X* | | *X* |
| Byte 4 | Topic Alias MSB | | | | | | | | | | | | |
| Byte 5 | Topic Alias LSB | | | | | | | | | | | | |
| Byte 6 | Packet Id MSB | | | | | | | | | | | | |
| Byte 7 | Packet Id LSB | | | | | | | | | | | | |
| Byte 8 | Return Code | | | | | | | | | | | | |



Table 20: SUBACK packet

The SUBACK packet is sent by a gateway to a client as an acknowledgment to the receipt and processing of a SUBSCRIBE packet. Its format is illustrated in Table 20:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the subscribe flags please refer to figure 3.2.3.

#### Topic Alias

In case of “accepted” the value that will be used as topic alias by the gateway when sending PUBLISH packets to the client (not relevant in case of subscriptions to a short topic name or to a topic name which contains wildcard characters)

#### Packet Id

Same value as the one contained in the corresponding SUBSCRIBE packet.

#### Return Code

Accepted”, or rejection reason.

### UNSUBSCRIBE

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | | **2** | **1** | | **0** | |
| Byte 1 | Length | | | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | | | |
|  | *Reserved* | | *Reserved* | | *Reserved* | | | *Reserved* | | *Reserved* | | *Topic Type* | | |
| Byte 3 | *0* | | *0* | *0* | *0* | | | *0* | | *0* | | *X* | | *X* |
| Byte 4 | Packet Id MSB | | | | | | | | | | | | | |
| Byte 5 | Packet Id LSB | | | | | | | | | | | | | |
| Byte 6 | Topic Alias MSB | | | | | | | | Topic Name  Byte 6 … N | | | | | |
| Byte 7 | Topic Alias LSB | | | | | | | |

Table 21: UNSUBSCRIBE packet

An UNSUBSCRIBE packet is sent by the client to the GW to unsubscribe from named topics. Its format is illustrated in Table 21:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the subscribe flags please refer to figure 3.2.3.

#### Packet Id

Should be coded such that it can be used to identify the corresponding SUBACK packet.

#### Topic Alias or Topic Name

Contains topic name, pre-defined topic alias, or short topic name as indicated in the *Topic Alias Type* field.

### UNSUBACK

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Packet Id MSB | | | | | | | |
| Byte 4 | Packet Id LSB | | | | | | | |
| Byte 5 | Return Code | | | | | | | |



Table 22: UNSUBACK packet

An UNSUBACK packet is sent by a GW to acknowledge the receipt and processing of an UNSUBSCRIBE packet. Its format is illustrated in Table 21:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Packet Id

Same value as the one contained in the corresponding UNSUBSCRIBE packet.

#### Return Code

Accepted, or rejection reason. See section 5.3.10.

### PINGREQ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Max Messages (optional) | | | | | | | |
| Byte 4 … N | Client Identifier (optional) | | | | | | | |



Table 23: PINGREQ packet

As with MQTT, the PINGREQ packet is an ”are you alive” packet that is sent from or received by a connected client. Its format is illustrated in Table 22:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Max Messages

The maximum number of messages that can be received by a client during its awake state. 0 means no limit. This field is optional, only included when a sleeping client goes to the awake state.

#### Client Identifier

Contains the client identifier (client id); this field is optional and is included by a “sleeping” client when it goes to the “awake” state and is waiting for packets sent by the server/gateway, see Section 6.14 for further details.

### PINGRESP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Messages Remaining (optional) | | | | | | | |



Table 24: PINGRESP packet

As with MQTT, a PINGRESP packet is the response to a PINGREQ packet and means ”yes I am alive”. Keep Alive packets flow in either direction, sent either by a connected client or the gateway. Its format is illustrated in Table 23: it has only a header and no variable part.

Moreover, a PINGRESP packet is sent by a gateway to inform a sleeping client that it has no more buffered packets for that client, see Section 6.14 for further details.

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Messages Remaining

The number of messages left when a client is sent back to sleep. Optional - only used at the end of a client awake period. Values can be:

|  |  |
| --- | --- |
| **Allowed Values** | **Description** |
| 0 | No messages remain in the queue |
| 1 – 65534 (incl.) | The number of messages remaining in the queue |
| 65535 (0xFFFF) | An unspecified positive number of messages remain in the queue greater than 0. |

### DISCONNECT

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Return Code (optional) | | | | | | | |
| Byte 4 | Session Expiry Interval MSB (optional) | | | | | | | |
| Byte 5 | Session Expiry Interval (optional) | | | | | | | |
| Byte 6 | Session Expiry Interval (optional) | | | | | | | |
| Byte 7 | Session Expiry Interval LSB (optional) | | | | | | | |
| Byte 8 … N | Reason String (optional) | | | | | | | |



Table 25: DISCONNECT packet

As with MQTT, the DISCONNECT packet is sent by a client to indicate that it wants to close the connection. The gateway will acknowledge the receipt of that packet by returning a DISCONNECT to the client. A server or gateway may also sends a DISCONNECT to a client, e.g. in case a gateway, due to an error, cannot map a received packet to a client. Upon receiving such a DISCONNECT packet, a client should try to setup the connection again by sending a CONNECT packet to the gateway or server. In all these cases the DISCONNECT packet does not contain the *Duration* field.

A DISCONNECT packet with a *Session Expiry Interval* field is sent by a client when it wants to go to the “asleep” state. The receipt of this packet is also acknowledged by the gateway by means of a DISCONNECT packet (without a duration field).

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Return Code

Reason for disconnection. See section 5.3.10.

#### Session Expiry Interval

Contains the value of the Session Expiry Interval timer; when the value of this field is greater than zero, it is deemed to be sent by a client that wants to transition to the “asleep” state, see Section 6.14 for further details.

#### Reason String

A string representing a clear text description of disconnection.

### WILLTOPICUPD

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | | **5** | | **4** | **3** | | **2** | **1** | | **0** |
| Byte 1 | Length | | | | | | | | | | | |
| Byte 2 | Packet Type | | | | | | | | | | | |
|  | *Reserved* | | *Will QoS* | | *Retain* | | | *Reserved* | *Reserved* | | *Reserved* | *Reserved* |
| Byte 3 | *0* | | *X* | *X* | *X* | | | *0* | *0* | | *0* | *0* |
| Byte 4.. N | Will Topic | | | | | | | | | | | |



Table 26: WILLTOPICUPD packet

The WILLTOPICUPD packet is sent by a client to update its Will topic name stored in the GW/server. Its format is shown in Table 26:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Flags

For a detailed description of the will flags field please refer to figure 3.2.4.

#### Will Topic

Contains the Will topic name.

An empty WILLTOPICUPD packet is a WILLTOPICUPD packet without Flags and WillTopic field (i.e. it is exactly 2 bytes long). It is used by a client to delete its Will topic and Will packet stored in the GW/server.

### WILLPACKETUPD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 .. N | Will Packet | | | | | | | |

Table 27: WILLPACKETUPD packet

The WILLPACKETUPD packet is sent by a client to update its Will packet stored in the GW/server.

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Will Packet

Contains the Will packet.

### WILLTOPICRESP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Return Code | | | | | | | |



Table 28: WILLTOPICRESP and WILLPACKETRESP packets

The WILLTOPICRESP packet is sent by a GW to acknowledge the receipt and processing of an WILLTOPICUPD packet. Its format is illustrated in Table 28:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Return Code

Accepted, or rejection reason

### WILLPACKETRESP

The WILLPACKETRESP packet is sent by a GW to acknowledge the receipt and processing of an WILLPACKETUPD packet. Its format is illustrated in Table 28:

#### Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Please refer to figure 3.1.2 for a detailed description.

#### Return Code

Accepted, or rejection reason

## Forwarder Encapsulation

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Byte 1 | Length | | | | | | | |
| Byte 2 | Packet Type | | | | | | | |
| Byte 3 | Ctrl | | | | | | | |
| Byte 4 .. N | Wireless Node Id | | | | | | | |
| Byte (N + 1 ,,, M) | MQTT SN packet | | | | | | | |

Table 29: Format of an encapsulated MQTT-SN frame

#### Length

1-byte long, specifies the number of bytes up to the end of the “Wireless Node Id” field (incl. the Length byte itself)

#### Packet Type

Coded “0xFE”, see Table 3

#### Ctrl

The Ctrl byte contains control information exchanged between the GW and the forwarder. Its format is shown in Table 30:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|  | Reserved | | | | | | *Radius* | |
|  | 0 | 0 | 0 | 0 | 0 | 0 | X | X |

Table 29: Format of the ctrl byte

* Radius: broadcast radius (only relevant in direction GW to forwarder)

#### Wireless Node Id

Identifies the wireless node which has sent or should receive the encapsulated MQTT-SN packet. The mapping between this Id and the address of the wireless node is implemented by the forwarder, if needed.

#### MQTT SN Packet

The MQTT-SN packet, encoded according to Table 1.

# Operational behavior

An important design point of MQTT-SN is to be as close as possible to MQTT. Therefore, all protocol semantics should remain, as far as possible, the same as those defined by MQTT. In the following we will focus on those points that either are new to or deviate from MQTT.

## Gateway Advertisement and Discovery

This procedure is new and does not exist in MQTT.

A gateway may announce its presence by broadcasting periodically an ADVERTISE packet to all devices that are currently parts of the network. A gateway should only advertise its presence if it is connected to a server (or is itself a server).

Multiple gateways may be active at the same time in the same network. In this case they will have different ids. It is up to the client to decide to which gateway it wants to connect. However at any point in time a client is allowed to be connected to only one gateway.

A client should maintain a list of active gateways together with their network addresses. This list is populated by means of the ADVERTISE and GWINFO packets received.

The time duration *TADV* until the gateway sends the next ADVERTISE packet is indicated in the *Duration* field of the ADVERTISE packets. A client may use this information to monitor the availability of a gateway. For example, if it does not receive ADVERTISE packets from a gateway for *NADV* consecutive times, it may assume that the gateway is down and removes it from its list of active gateways. Similarly, gateways in stand-by mode will become active (i.e. start sending ADVERTISE packets) if they miss successively a couple of times advertisements from a certain gateway.

Since the ADVERTISE packets are broadcasted into the whole wireless network, the time interval *TADV* between two ADVERTISE packets sent by a gateway should be large enough (e.g. greater than 15 min) to avoid bandwidth congestion in the network.

The large value of *TADV* will lead to a long waiting time for new clients which are looking for a gateway. To shorten this waiting time a client may broadcast a SEARCHGW packet. To prevent broadcast storms when multiple clients start searching for GW almost at the same time, the sending of the SEARCHGW packet is delayed by a random time between 0 and *TSEARCHGW*. A client will cancel its transmission of the SEARCHGW packet if it receives during this delay time a SEARCHGW packet sent by another client and identical to the one it wants to send, and behaves as if the SEARCHGW packet was sent by itself.

The broadcast radius *Rb* of the SEARCHGW packet is limited, e.g. to a single hop in case of a dense deployment of MQTT-SN clients.

Upon receiving a SEARCHGW packet a gateway replies with a GWINFO packet containing its id. Similarly, a client answers with a GWINFO packet if it has at least one active gateway in its list of active gateways. If the client has multiple GWs in its list, it selects one GW out of its list and includes that information into the GWINFO packet.

Like the SEARCHGW packet, the GWINFO packet is broadcast with the same radius *Rb*, which is indicated in the SEARCHGW packet. The radius *Rb* is also given to the underlying layer when these two packets are passed down for transmission.

To give priority to the gateways a client will delay its sending of the GWINFO packet for a random time

*TGWINFO*. If during this delay time the client receives a GWINFO packet it will cancel the transmission of its GWINFO packet.

In case of no response the SEARCHGW packet may be retransmitted. In this case the time intervals between two consecutive SEARCHGW packets should be increased exponentially.

## Client’s Connection Setup

As with MQTT, a MQTT-SN client needs to setup a connection to a GW before it can exchange information with that GW. The procedure for setting up a connection with a GW is illustrated in Fig. 3, in which it is assumed that the client requests the gateway to prompt for the transfer of Will topic and Will packet. This request is indicated by setting the Will flag of the CONNECT packet. The client then sends these two pieces of information to the GW upon receiving the corresponding request packets WILLTOPICREQ and WILLPACKETREQ. The procedure is terminated with the CONNACK packet sent by the GW.

If Will flag is not set then the GW answers directly with a CONNACK packet.

A screenshot of a cell phone

Description automatically generated

Figure 3: Connect procedure

In case the GW could not accept the connection request (e.g. because of congestion or it does not support a feature indicated in the CONNECT packet), the GW returns a CONNACK packet with the rejection reason.

In the case where the client provides a zero length client identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server.

## Clean start

With MQTT, when a client disconnects, its subscriptions are not deleted. They are persistent and valid for new connections, until either they are explicitly un-subscribed by the client, or the client establishes a new connection with the “clean start” flag set.

In MQTT-SN the meaning of a “clean start” is extended to the Will feature, i.e. not only the subscriptions are persistent, but also the Will topic and the Will packet. The two flags “CleanStart” and “Will” in the CONNECT have then the following meanings:

* CleanStart=true, Will=true: The GW will delete all subscriptions and Will data related to the client, and starts prompting for new Will topic and Will packet.
* CleanStart=true, Will=false: The GW will delete all subscriptions and Will data related to the client, and returns CONNACK (no prompting for Will topic and Will packet).
* CleanStart=false, Will=true: The GW keeps all stored client’s data, but prompts for new Will topic and Will packet. The newly received Will data will overwrite the stored Will data.
* CleanStart=false, Will=false: The GW keeps all stored client’s data and returns CONNACK (no prompting for Will topic and Will packet).

Note that if a client wants to delete only its Will data at connection setup, it could send a CONNECT packet with “CleanStart=false” and “Will=true”, and sends an empty WILLTOPIC packet to the GW when prompted to do so. It could also send a CONNECT packet with “CleanStart=false” and “Will=false”, and use the procedure of Section 6.4 to delete or modify the Will data.

## Procedure for updating the Will data

At any time during a connection a client could update its Will data stored in the gateway by sending a WILLTOPICUPD or a WILLPACKETUPD packet. The information contained in these two packets will overwrite the corresponding ones stored in the gateway. Both packets are acknowledged by the gateway. Both packets can be used independently from each other.

Note that an empty WILLTOPICUPD packet will delete both the Will topic and Will packet stored at the gateway.

## Topic Name Registration Procedure

Because of the limited bandwidth and the small packet payload in wireless sensor networks, data is not published together with its topic name as in MQTT. A registration procedure is introduced which allows both a client and a GW to inform its peer about the short topic alias and its corresponding topic name before it can start sending PUBLISH packets using the short topic alias.

To register a topic name a client sends a REGISTER packet to the GW. If the registration could be accepted, the gateway assigns a *topic alias* to the received topic name and returns it with a REGACK packet to the client. If the client initiates a REGISTER against a topic which is known by the gateway to have a predefined topic alias associated with it, the gateway will specify its topic alias type to be predefined and set the topic alias value to match this in the REGACK. The client can then choose to update its registry of predefined topic alias’ if it so wishes. If there are no predefined topic alias’, the gateway will pass back a NORMAL topic alias type. If the registration could not be accepted, a REGACK is also returned to the client with the failure reason encoded in the *ReturnCode* field.

After having received the REGACK packet with *ReturnCode=“accepted”*, the client shall use the assigned *topicId* to publish data of the corresponding topic name. If however the REGACK contains a rejection code, the client may try to register later again. If the return code was *“rejected: congestion”*, the client should wait for a time *TWAIT* before restarting the registration procedure.

At any point in time a client may have only one REGISTER packet outstanding, i.e. it has to wait for a REGACK packet before it can register another topic name.

A GW sends a REGISTER packet to a client if it wants to inform that client about the topic name and the assigned topic alias that it will use later on when sending PUBLISH packets of the corresponding topic name. This happens for example when the client re-connects without having set the “CleanSession” flag or the client has subscribed to topic names that contain wildcard characters such as # or +.

Topic Alias mappings exist only while a client is active and last for the entire duration of the active state. A receiver MUST NOT carry forward any Topic Alias mappings from one active state to another.

## Topic Name Mapping and Aliasing

On the gateway the mapping table between registered topic ids and topic names is implemented per client (and not by a single shared pool between all clients), to reduce the risk of an incorrect topic id from a client matching another client’s valid topic. For performance and efficiency reasons the broker may choose to align topic alias’ for registered topic aliases between multiple clients. The mapping table of predefined topic aliases is separate from registered aliases. It is global and shared between all clients and gateways and may overlap with registered aliases, since it is considered to be in a different pool.

## Pre-defined topic alias’ and short topic names

A topic alias is a two-byte long replacement of the string-based topic name. A client needs to use the REGISTER procedure to inform the gateway about the topic name it wants to employ and gets from the gateway the corresponding topic alias. It then will use this topic alias in the PUBLISH packets it sends to the gateway. In the opposite direction, the PUBLISH packets also contain a 2-byte topic alias (instead of the string-based topic name). The client is informed about the relation between topic alias and topic name by means of either a former SUBSCRIBE procedure or a REGISTER procedure started by the gateway.

A “pre-defined” topic alias is a topic alias whose mapping to a topic name is known in advance by both the client’s application and the gateway. This is indicated in the *Flags* field of the packet. When using pre-defined topic alias’, both sides can start immediately with the sending of PUBLISH packets; there is no need for the REGISTER procedure as in the case of ”normal” topic alias’. When receiving a PUBLISH packet with a pre-defined topic alias, of which the mapping to a topic name is unknown, the receiver should return a PUBACK with the *ReturnCode= “Rejection: invalid topic* alias*”*. Note that this error situation cannot be resolved by means of re-registering as in the case of normal topic alias.

A client is still required to subscribe to a pre-defined topic alias, if it wants to receive PUBLISH packets relating to that topic alias. To avoid confusion between a pre-defined topic alias and a two-byte short topic name, the SUBSCRIBE packet contains a flag indicating whether it is subscribing for a short topic name or a pre-defined topic alias.

A “short” topic name is a topic name that has a fixed length of two bytes. It could be carried together with the data within a PUBLISH packet, thus no REGISTER procedure is needed for a short topic name. Otherwise, all rules that apply to normal topic names also apply to short topic names. Note however that it does not make sense to do wildcarding in subscriptions to short topic names, because it is not possible to define a meaningful name hierarchy with only two characters.

## Client’s Topic Subscribe/Unsubscribe Procedure

To subscribe to a topic name, a client sends a SUBSCRIBE packet to the gateway with the topic name included in that packet. If the gateway is able accept the subscription, it assigns a topic alias to the received topic name and returns it within a SUBACK packet to the client. If the subscription cannot be accepted, then a SUBACK packet is also returned to the client with the rejection cause encoded in the *ReturnCode* field. If the rejection cause is *“rejected: congestion”*, the client should wait for the time *TWAIT* before resending the SUBSCRIBE packet to the gateway.

If the client subscribes to a topic name which contains a wildcard character, the returning SUBACK packet will contain the topic alias value 0x0000. The GW will the use the registration procedure to inform the client about the to-be-used topic alias value when it has the first PUBLISH packet with a matching topic name to be sent to the client, see also Section 6.10.

Similar to the client’s PUBLISH procedure, topic alias’ may also be pre-defined for certain topic names. Short topic names may be used as well. In those two cases the client still needs to subscribe to those pre-defined topic alias’ or short topic names.

To unsubscribe, a client sends an UNSUBSCRIBE packet to the gateway, which will then be answered by means of an UNSUBACK packet.

As for the REGISTER procedure, a client may have only one SUBSCRIBE or one UNSUBCRIBE transaction open at a time.

## Client’s Publish Procedure

After having registered successfully a topic name with the gateway, the client can start publishing data relating to the registered topic name by sending PUBLISH packets to the gateway. The PUBLISH packets contain the assigned topic alias.

All three QoS levels and their corresponding packet flows are supported as defined in MQTT. The only difference is the use of topic alias’ instead of topic names in the PUBLISH packets.

Regardless of the requested QoS level the client may receive in response to its PUBLISH a PUBACK packet which contains either

* the *ReturnCode= “Rejection: invalid topic* alias*”*: in this case the client needs to register the topic name again before it can publish data related to that topic name; or
* the *ReturnCode= “Rejection: congestion”*: in this the client shall stop publishing toward the gateway for at least the time *TWAIT*.

At any point in time a client may have only one QoS level 1 or 2 PUBLISH packet outstanding in each direction; i.e. it has to wait for the termination of this PUBLISH packet exchange before it could start a new level 1 or 2 transaction

## PUBLISH with QoS Level -1

This feature is defined for very simple client implementations which do not support any other features except this one. There is no connection setup nor tear down, no registration nor subscription. The client just sends its PUBLISH packets to a GW (whose address is known a-priori by the client) and forgets them. It does not care whether the GW address is correct, whether the GW is alive, or whether the packets arrive at the GW. Only the following parameter values are allowed for a PUBLISH packet with QoS level -1:

* QoS flag: set to “0b11”;
* Topic Alias Type flag: either “0b01” for pre-defined topic alias or “0b10” for short topic name;
* Topic Alias field: value of the pre-defined topic alias or of the short topic name;
* Data field: the data to be published.

A client may wish to PUBLISH at Level -1 at any point in their lifecycle, this includes either DISCONNECTED or SLEEPING states.

## Gateway’s Publish Procedure

Similar to the client’s PUBLISH procedure described in Section 6.6, the gateway sends PUBLISH packets with the topic alias value that was returned in the SUBACK packet to the client.

Preceding the PUBLISH packet the GW may send a REGISTER packet to inform the client about the topic name and its assigned topic alias value. This will happen for example when the client re-connects without clean start or has subscribed to topic names with wildcard characters. Upon receiving a REGISTER packet the client replies with a REGACK packet. The GW will wait for the REGACK packet before it sends the PUBLISH packet to the client.

The client could reject the REGISTER packet with a REGACK packet indicating the rejection reason; this corresponds to an unsubscribe to the topic name indicated in the REGISTER packet. Note that unsubscribe to a topic name with wildcard characters can only be done with the unsubscribe procedure described in Section 6.9 and not with the rejection of a REGISTER packet, since a REGISTER packet never contains a topic name with wildcard characters.

If the client receives a PUBLISH packet with an unknown topic alias value, it shall respond with a PUBACK packet with the *ReturnCode=“Rejected: invalid topic alias”*. This will trigger the gateway to delete or correct the wrong topic alias assignment.

Note that in case either the topic name or the data is too long to fit into a REGISTER or a PUBLISH packet, the gateway silently aborts the publish procedure, i.e. no warning is sent to the affected subscribers.

## Keep Alive and PING Procedure

As with MQTT, the value of the Keep Alive timer is indicated in the CONNECT packet. The client should send a PINGREQ packet within each Keep Alive time period, which the GW acknowledges with a PINGRESP packet.

Similarly, a client shall answer with a PINGRESP packet when it receives a PINGREQ packet from the GW to which it is connected. Otherwise the received PINGREQ packet is ignored.

Clients should use this procedure to supervise the liveliness of the gateway to which they are connected. If a client does not receive a PINGRESP from the gateway even after multiple retransmissions of the PINGREQ packet, it should first try to connect to another gateway before trying to re-connect to this gateway (see also section 6.13). Note that because the clients’ keep alive timers are not synchronized with each other, in case of a gateway failure there is practically no danger for a storm of CONNECT packets sent almost at the same time by all affected clients towards a new gateway.

## Client’s Disconnect Procedure

A client sends a DISCONNECT packet to the GW to indicate that it is about to close its connection. After this point, the client is then required to establish a new connection with the GW before it can exchange information with that GW again. Similar to MQTT, sending the DISCONNECT packet does not affect existing subscriptions and Will data if the CleanSession flag is set. They are persistent until they are either explicitly un-subscribed, or deleted, or modified by the client, or if the client establishes a new connection with the CleanSession flag set. The gateway acknowledges the receipt of the DISCONNECT packet by returning a DISCONNECT to the client.

A client may also receive an unsolicited DISCONNECT sent by the gateway. This may happen for example when the gateway, due to an error, cannot identify the client to which a received packet belongs. Upon receiving such a DISCONNECT packet a client should retry to setup the connection again by sending a CONNECT packet to the gateway.

## Client’s Retransmission Procedure

All packets that are “unicasted” to the GW (i.e. sent using the GW’s unicast address and not broadcasted) and for which a GW’s reply is expected are supervised by a retry timer *Tretry* and a retry counter *Nretry*. The retry timer *Tretry* is started by the client when the packet is sent and stopped when the expected GW’s reply is received. If *Tretry* times out and the expected GW’s reply is not received, the client retransmits the packet. After *Nretry* retransmissions, the client aborts the procedure and assumes that its MQTT-SN connection to the gateway is disconnected. It should then try to connect to another gateway, and only if it fails to re-connect again to the former gateway.

## Support of sleeping clients

*Sleeping* clients are clients residing on (battery-operated) devices that want to save as much energy as possible. These devices need to enter a sleep mode whenever they are not active, and will wake up whenever they have data to send or to receive. The server/gateway needs to be aware of the sleeping state of these clients and will buffer messages destined to them for later delivery when they wake up. The gateway must guarantee to buffer quality-of-service messages 1 and 2 and *may* choose to buffer messages of all quality-of-service 0, whilst the client is sleeping and is within it’s expiry interval.

A picture containing text, map

Description automatically generated

Figure 4: Client’s state transition diagram

As illustrated in Fig 4, from the perspective of the server/gateway, a client may be in one of the following states: *active, asleep, awake, disconnected,* or *lost*. A client is in the *active* state when the server/gateway receives a CONNECT packet from that client, as described in section 6.2. This state is supervised by the server/gateway with the “keep alive” timer as described in section 6.11. If the server/gateway does not receive any packet from the client for a period longer than the keep alive duration (indicated in the CONNECT packet), the gateway will consider that client as *lost* and activates for example the Will feature for that client.

A client goes to the *disconnected* state when the server/gateway receives a DISCONNECT without a duration field. This state is not time-supervised by the server/gateway.

If a client wants to sleep, it sends a DISCONNECT packet which contains a sleep duration. The server/gateway acknowledges that packet with a DISCONNECT packet and considers the client for being in *asleep* state, see also Fig. 5. The *asleep* state is supervised by the server/gateway with the indicated sleep duration. If the server/gateway does not receive any packet from the client for a period longer than the sleep duration, the server/gateway will consider that client as *lost* and - as with the keep alive procedure - activates for example the Will feature. During the *asleep* state, all packets that need to be sent to the client are buffered at the server/gateway.

The sleep timer is stopped when the server/gateway receives a PINGREQ from the client. Like the CONNECT packet, this PINGREQ packet contains the *Client Id*. The identified client is then in the *awake* state. If the server/gateway has buffered packets for the client, it will send these packets to the client, acknowledging the max-receive value sent in the PINGREQ packet. If the number of messages buffered on the gateway queue exceeds the value specified by the client in the max-receive field, the gateway shall send only the max-receive value number of messages, and cut short the AWAKE cycle, responding with a PINGRESP with a messages-left value of either the number of messages remaining in the gateway buffer or 0xFFFF (meaning undetermined number of messages greater than 0 remaining).

For each packet the gateway sends to the client, the packets' quality of service shall be honoured, and a full packet interaction shall take place, including any associated retransmission logic.

The transfer of packets to the client is closed by the server/gateway by means of a PINGRESP packet, i.e. the server/gateway will consider the client as *asleep* and restart the sleep timer again after having sent the PINGRESP packet.

If the server/gateway does not have any packets buffered for the client, it answers immediately with a PINGRESP packet, returns the client back to the *asleep* state, and restarts the sleep timer for that client.

After having sent the PINGREQ to the server/gateway, the client uses the “retransmission procedure” of section 6.13 to supervise the arrival of packets sent by the server/gateway, i.e. it restarts timer Tretry when it receives a packet other than a PINGRESP, and stops it when it receives a PINGRESP. The PINGREQ packet is retransmitted, and timer Tretry restarted when timer Tretry times out. To avoid a flattening of its battery due to excessive retransmission of the PINGREQ packet (e.g. if it loses the gateway), the client should limit the retransmission of the PINGREQ packet (e.g. by a retry counter) and go back to sleep when the limit is reached and it still does not receive a PINGRESP packet.

From the *asleep* or *awake* state, a client can return either to the *active* state by sending a CONNECT packet or to the *disconnected* state by sending a normal DISCONNECT packet (i.e. without duration field). The client can also modify its sleep duration by sending a DISCONNECT packet with a new value of the sleep duration.

Note that a sleeping client should go the *awake* state only if it just wants to check whether the server/gateway has any messages buffered for it and return as soon as possible to the *asleep* state without sending any packets to the server/gateway. Otherwise, it should return to the *active* state by sending a CONNECT packet to the server/gateway.

Per section 5.5, Topic Alias mappings exist only while a client is active and last for the entire duration of the active state. Therefore, when the gateway must re-register any topic alias’s during the AWAKE state, which will last until the last PINGRESP is issued. The gateway should attempt to make the best effort to reuse the same topic alias’ mappings that existed during any initial associated ACTIVE states.

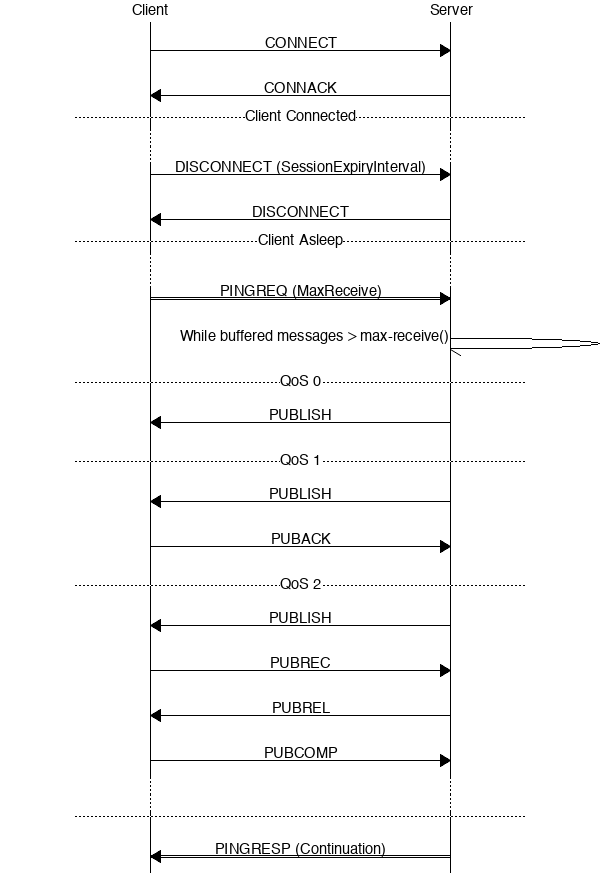


Figure 5: Awake ping packet flush

# Authentication

Authentication involves the exchange of AUTH packets between the Client and the Server after the CONNECT and before the CONNACK packets.

To begin an authentication exchange, the Client sets the AUTH flag in the CONNECT packet. It then sends an AUTH packet with an Authentication Method. This specifies the authentication method to use. If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) and MUST close the Connection.

The Authentication Method is an agreement between the Client and Server about the meaning of the data sent in the Authentication Data and any of the other fields in CONNECT, and the exchanges and processing needed by the Client and Server to complete the authentication.

**Non-normative comment**

The Authentication Method is commonly a SASL mechanism, using such a registered name aids interchange. However, the Authentication Method is not constrained to using registered SASL mechanisms.

If the Authentication Method selected by the Client specifies that the Client sends data first, the Client SHOULD include an Authentication Data property in the AUTH packet. This property can be used to provide data as specified by the Authentication Method. The contents of the Authentication Data are defined by the authentication method.

If the Server requires additional information to complete the authentication, it can send an AUTH packet to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication). If the authentication method requires the Server to send authentication data to the Client, it is sent in the Authentication Data.

The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet MUST contain a Reason Code of 0x18 (Continue authentication). If the authentication method requires the Client to send authentication data for the Server, it is sent in the Authentication Data.

The Client and Server exchange AUTH packets as needed until the Server accepts the authentication by sending a CONNACK with a Reason Code of 0. If the acceptance of the authentication requires data to be sent to the Client, it is sent in the Authentication Data.

The Client can close the connection at any point in this process by sending a DISCONNECT packet. The Server can reject the authentication at any point in this process by sending a CONNACK with a Reason Code of 0x80 or above as described in section 4.13.

The implementation of authentication is OPTIONAL for both Clients and Servers. If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet. If the Client does not set the Authentication Flag in the CONNECT, the Client MUST NOT send an AUTH packet to the Server.

If the Client does not set the Authentication Flag in the CONNECT packet, the Server SHOULD authenticate using some or all of the information in the CONNECT packet and Network Connection.

**Non-normative example showing a user name and password authentication**:

· Client to Server: CONNECT Authentication Flag=1 Authentication Data=client-first-data

· Client to Server: AUTH rc=0x01 Authentication Method="PLAIN" Authentication Data=client-first-data

· Server to Client CONNACK rc=0

Where client-first data is the content of the SASL PLAIN message as described in RFC 4616:

*The mechanism consists of a single message, a string of [UTF-8]* *encoded [Unicode] characters, from the client to the server. The[UTF-8]* *client presents the authorization identity (identity to act as),* *followed by a NUL (U+0000) character, followed by the authentication* *identity (identity whose password will be used), followed by a NUL* *(U+0000) character, followed by the clear-text password. As with* *other SASL mechanisms, the client does not provide an authorization identity when it wishes the server to derive an identity from the* *credentials and use that as the authorization identity.*

# Retained Packets

If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace any existing retained packet for this topic and store the Publish Data, so that it can be delivered to future subscribers whose subscriptions match its Topic Name. If the Publish Data contains zero bytes it is processed normally by the Server but any retained packet with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained packet. A retained packet with Publish Data containing zero bytes MUST NOT be stored as a retained packet on the Server.

# Safety, Security, and Data Protection Considerations

(**Note:** OASIS strongly recommends that Technical Committees consider issues that might affect safety, security, privacy, and/or data protection in implementations of their specification and document these for implementers and adopters. For some purposes, you may find it required, e.g. if you apply for IANA registration.

While it may not be immediately obvious how your specification might make systems vulnerable to attack, most specifications, because they involve communications between systems, packet formats, or system settings, open potential channels for exploit. For example, IETF [[RFC3552](#RFC3552)] lists “eavesdropping, replay, packet insertion, deletion, modification, and man-in-the-middle” as well as potential denial of service attacks as threats that must be considered and, if appropriate, addressed in IETF RFCs.

In addition to considering and describing foreseeable risks, this section should include guidance on how implementers and adopters can protect against these risks.

We encourage editors and TC members concerned with this subject to read *Guidelines for Writing RFC Text on Security Considerations,* IETF [[RFC3552](#RFC3552)], for more information.)

# Conformance

(**Note**: The [OASIS TC Process](https://www.oasis-open.org/policies-guidelines/tc-process#wpComponentsConfClause) requires that a specification approved by the TC at the Committee Specification Public Review Draft, Committee Specification or OASIS Standard level must include a separate section, listing a set of numbered conformance clauses, to which any implementation of the specification must adhere in order to claim conformance to the specification (or any optional portion thereof). This is done by listing the conformance clauses here.

For the definition of "conformance clause," see [OASIS Defined Terms](https://www.oasis-open.org/policies-guidelines/oasis-defined-terms-2017-05-26#dConformanceClause).

See "Guidelines to Writing Conformance Clauses":   
<https://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html>.

Remove this note before submitting for publication.)

1. Acknowledgments

(**Note:** A Work Product approved by the TC must include a list of people who participated in the development of the Work Product. This is generally done by collecting the list of names in this appendix. This list shall be initially compiled by the Chair, and any Member of the TC may add or remove their names from the list by request.

Remove this note before submitting for publication.)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

1. Implementation Notes
   1. Support of QoS Level -1

Because PUBLISH packets with QoS level -1 could be sent at any time by clients (even with no connection setup) a transparent GW needs to maintain for those packets a dedicated MQTT connection with the server. An aggregating or hybrid GW may use any aggregating MQTT connection to forward those packets to the server.

* 1. “Best practice” values for timers and counters

Table 30 shows the “best practice” values for the timers and counters defined in this specification.

|  |  |
| --- | --- |
| Timer/Counter | Recommended value |
| *TADV* | greater than 15 minutes |
| *NADV* | 2 -3 |
| *TSEARCHGW* | 5 seconds |
| *TGWINFO* | 5 seconds |
| *TWAIT* | greater than 5 minutes |
| *Tretry* | 10 - 15 seconds |
| *Nretry* | 3 - 5 |

Table 30: “Best practice” values for timers and counters

The “tolerance” of the sleep and keep-alive timers at the server/gateway depends on the duration indicated by the clients. For example, the timer values should be 10% higher than the indicated values for durations larger than 1 minute, and 50% higher if less.

* 1. Mapping of Topic Alias to Topic Names

It is strongly recommended that in the gateway the mapping table between topic alias and topic names is implemented per client (and not by a single shared pool between all clients), to reduce the risk of an incorrect topic alias from a client matching another client’s valid topic, and thus causing a publication to the wrong topic, which could potentially have disastrous consequences.

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| WD-01 | [27th February 2020] | [Andrew Banks] | [Merge Initial Document and Input Specification] |
| WD-02 | [4th April 2020] | [Andrew Banks]  [Rahul Gupta] | [Terminology, DataTypes, CONNECT packet]  [Specification Diagrams] |
| WD-05 | [21st February 2021] | [Simon Johnson] | [Packet Diagrams, Bit Tables, Field Definitions] |
| WD-06 | [10th March 2021] | [Simon Johnson] | [Sleeping client operational behavior, Terminology changes, 13 JIRA resolutions added to specification, Section numbering changes] |
| WD-07 | [15th March 2021] | [Simon Johnson] | [Added 4 byte (32 bit) integer description] |
| WD-08 | [26th March 2021] | [Simon Johnson] | [Added max packet size to CONNECT, Added Session Expiry Interval to CONNACK, Removed ZigBee references, Removed capabilities flag from CONNECT, AUTH packet added along with Authentication operational behavior. Standardized page margins] |
| WD-09 | [05th May 2021] | [Simon Johnson] | [Added long topic type to topicIdTypes, updated PUBLISH to accommodate new topic type, added topic type matrix] |