MQTT for Sensor Networks (MQTT-SN) Version 2.0

Committee Specification Draft 01

01 May 2025

This stage:

…

Previous stage:

N/A

Latest stage:

…

Technical Committee:

OASIS Message Queuing Telemetry Transport (MQTT) TC

Chairs:

Ian Craggs (icraggs@gmail.com), Individual
Simon Johnson (simon.johnson@hivemq.com), HiveMQ GmbH

Editors:

Andrew Banks (andrewdjbanks@gmail.com), Individual
Andy Stanford-Clark (andysc@uk.ibm.com), IBM
Davide Lenzarini (davide.lenzarini@u-blox.com), u-blox AG
Ian Craggs (icraggs@gmail.com), Individual
Rahul Gupta (rahul.gupta@us.ibm.com), IBM
Simon Johnson (simon.johnson@hivemq.com), HiveMQ GmbH
Stefan Hagen (stefan@hagen.link), Individual
Tara E. Walker (tara.walker@microsoft.com), Microsoft Corporation

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.
- *MQTT-SN Version 1.2* by Andy Stanford-Clark and Hong Linh Truong. Link: https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf.

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. MQTT-SN is optimized for implementation on low-cost, battery-operated devices with limited processing and storage resources. It is designed so that it will work over a variety of networking technologies and bridge to an MQTT network.

Status:

This document was last revised or approved by the OASIS Message Queuing Telemetry Transport (MQTT) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt#technical .

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/mqtt/.

This specification is provided under the Non-Assertion Mode of the OASIS IPR Policy, 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, 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) 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.

Citation format:

When referencing this document, the following citation format should be used:

[MQTT-SN-v2.0]

MQTT for Sensor Networks Version 2.0. Edited by Andrew Banks, Davide Lenzarini, Ian Craggs, Rahul Gupta, Simon Johnson, Stefan Hagen, and Tara E. Walker. 01 May 2025. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/mqtt/mqtt-sn/v2.0/csd01/mqtt-sn-v2.0-csd01.docx. Latest stage: https://docs.oasis-open.org/mqtt/mqtt-sn/v2.0/mqtt-sn-v2.0.docx

(**Note:** Publication URIs are managed by OASIS TC Administration; please don't modify. The OASIS TC Process requires that Work Products at any level of approval must use the OASIS file naming scheme, 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.)

Notices

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 may be found at the OASIS website: [https://www.oasis-open.org/policies-guidelines/ipr/].

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. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specifications, OASIS Standards, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or

deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of OASIS, the owner and developer of this document, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, documents, while reserving the right to enforce its marks against misleading uses. See https://www.oasis-open.org/policies-guidelines/trademark/ for above guidance.

Table of Contents

1 Introduction 13
1.0 Intellectual property rights policy 13
1.1 Changes from earlier Versions 13
1.1.1 MQTT-SN 1.2 13
1.2 Organization of the MQTT-SN specification 13
1.3 Terminology 13
1.4 Normative references 18
1.5 Informative References 20
1.6 MQTT For Sensor Networks (MQTT-SN) 20
1.6.1 Differences Between MQTT-SN and MQTT 21
1.7 Data representation 21
1.7.1 Bits (Byte) 21
1.7.2 Two Byte Integer 21
1.7.3 Four Byte Integer 21
1.7.4 UTF-8 Encoded String 21
2 MQTT-SN Control Packet format 24
2.1 Structure of an MQTT-SN Control Packet 24
2.1.1 Packet Header 24
2.1.2 Length 24
2.1.3 MQTT-SN Control Packet Type 25
2.2 Packet Identifier 27
2.3 Reason Code 29
2.4 Topic Types 35
3 MQTT-SN Control Packets 37
3.1 CONNECT - Connection Request 37
3.1.1 CONNECT Header 38
3.1.2 Connect Flags 38
3.1.2.1 Clean Start 38
3.1.2.2 Will Flag 38
3.1.2.3 Authentication Flag 39
3.1.2.4 Session Expiry Flag 39
3.1.2.5 Default Number of Awake Messages 39
3.1.3 Will Flags 39
3.1.3.1 Will Topic Type 40
3.1.3.2 Will QoS 40
3.1.3.3 Will Retain 40
3.1.4 Packet Identifier 40
3.1.5 Protocol Version 40
3.1.6 Keep Alive 41
3.1.7 Maximum Packet Size 42
3.1.8 Session Expiry Interval 42
3.1.9 Will Topic Alias or Will Topic Name Length 43

3.1.10 Will Topic Name 43
3.1.11 Will Payload Length 43
3.1.12 Will Payload 43
3.1.13 Authentication Method Length 43
3.1.14 Authentication Method 43
3.1.15 Authentication Data Length 44
3.1.16 Authentication Data 44
3.1.17 Client Identifier 44
3.1.18 CONNECT Actions 44
3.2 CONNACK - Connect Acknowledgement 46
3.2.1 CONNACK Header 46
3.2.2 CONNACK Flags 47
3.2.2.1 Session Present 47
3.2.2.2 Session Expiry Interval Flag (Sess Exp) 47
3.2.2.3 Server Keep Alive Flag (Server KA) 47
3.2.2.4 Authentication Flag (Auth) 48
3.2.3 Packet Identifier 48
3.2.4 Reason Code 48
3.2.5 Session Expiry Interval 48
3.2.6 Server Keep Alive 48
3.2.7 Authentication Method Length 48
3.2.8 Authentication Method 49
3.2.9 Authentication Data Length 49
3.2.10 Authentication Data 49
3.2.11 Assigned Client Identifier 49
3.3 AUTH - Authentication Exchange 49
3.3.1 AUTH Header 50
3.3.2 Packet Identifier 50
3.3.2 Reason Code 50
3.3.3 Authentication Method Length 50
3.3.4 Authentication Method 50
3.3.5 Authentication Data 51
3.3.6 AUTH Actions 51
3.4 REGISTER - Register Topic Alias Request 51
3.4.1 REGISTER Header 52
3.4.2 REGISTER Flags 52
3.4.2.1 Topic Alias Flag 52
3.4.2 Packet Identifier 52
3.4.3 Topic Alias 52
3.4.4 Topic Name 52
3.4.5 REGISTER Actions 52
3.5 REGACK - Register Topic Alias Acknowledgement 53
3.5.1 REGACK Header 53
3.5.2 REGACK Flags 53

3.5.2.1 Topic Type 53
3.5.2.2 Topic Alias Flag 53
3.5.3 Packet Identifier 54
3.5.4 Topic Alias 54
3.5.5 Reason Code 54
3.6 Publish Requests and Responses 54
3.6.1 PUBWOS - Publish Without Session 54
3.6.1.1 PUBWOS Header 55
3.6.1.2 PUBWOS Flags 55
3.6.1.2.1 Topic Type 55
3.6.1.2.2 Retain 56
3.6.1.3 Topic Alias or Topic Name Length 56
3.6.1.4 Topic Name 56
3.6.1.5 Payload 56
3.6.1.6 PUBWOS Actions 56
3.6.2 PUBLISH with QoS 0 57
3.6.2.1 PUBLISH Header 57
3.6.2.2 PUBLISH Flags 57
3.6.2.2.1 Topic Type 57
3.6.2.2.2 QoS 57
3.6.2.2.3 Retain 58
3.6.2.3 Topic Alias or Topic Name Length 58
3.6.2.4 Topic Name 58
3.6.2.5 Payload 58
3.6.2.6 PUBLISH - QoS 0 Actions 58
3.6.3 PUBLISH with QoS 1 and 2 59
3.6.3.1 PUBLISH Header 59
3.6.3.2 PUBLISH Flags 59
3.6.3.2.1 Topic Type 59
3.6.3.2.2 QoS 60
3.6.3.2.3 DUP 60
3.6.3.2.4 Retain 60
3.6.3.3 Packet Identifier 60
3.6.3.4 Topic Alias or Topic Name Length 60
3.6.3.5 Topic Name 60
3.6.3.6 Payload 61
3.6.3.7 PUBLISH Actions 61
3.6.4 PUBACK – Publish Acknowledgement (QoS 1 delivery) 62
3.6.4.1 PUBACK Header 62
3.6.4.2 Packet Identifier 62
3.6.4.3 Reason Code 62
3.6.4.4 PUBACK Actions 62
3.6.5 PUBREC - Publish Received (QoS 2 delivery part 1) 62

3.6.5.1 PUBREC Header 62
3.6.5.2 Packet Identifier 63
3.6.5.3 Reason Code 63
3.6.5.4 PUBREC Actions 63
3.6.6 PUBREL - Publish Release (QoS 2 delivery part 2) 63
3.6.6.1 PUBREL Header 63
3.6.6.2 Packet Identifier 63
3.6.6.3 Reason Code 63
3.6.6.4 PUBREL Actions 63
3.6.7 PUBCOMP - Publish Complete (QoS 2 delivery part 3) 63
3.6.7.1 PUBCOMP Header 64
3.6.7.2 Packet Identifier 64
3.6.7.3 Reason Code 64
3.6.7.4 PUBCOMP Actions 64
3.7 SUBSCRIBE - Subscribe Request 65
3.7.1 SUBSCRIBE Header 65
3.7.2 SUBSCRIBE Flags 65
3.7.2.1 Topic Type 65
3.7.2.2 Retain handling 65
3.7.2.3 Retain as Published 66
3.7.2.4 QoS 66
3.7.2.5 No Local 66
3.7.3 Packet Identifier 66
3.7.4 Topic Alias 66
3.7.5 Topic Filter 67
3.7.6 SUBSCRIBE Actions 67
3.8 SUBACK - Subscribe Acknowledgement 68
3.8.1 SUBACK Header 68
3.8.2 SUBACK Flags 69
3.11.2.1 Topic Type 69
3.8.2.1 Topic Alias Flag 69
3.8.3 Packet Identifier 69
3.8.4 Topic Alias 69
3.8.5 Reason Code 69
3.9 UNSUBSCRIBE - Unsubscribe Request 69
3.9.1 UNSUBSCRIBE Header 70
3.9.2 UNSUBSCRIBE Flags 70
3.9.2.1 Topic Type 70
3.9.3 Packet Identifier 70
3.9.4 Topic Alias 70
3.9.5 Topic Filter 71
3.9.6 UNSUBSCRIBE Actions 71
3.10 UNSUBACK - Unsubscribe Acknowledgement 71
3.10.1 UNSUBACK Header 72

3.10.2 Packet Identifier 72
3.10.3 Reason Code 72
3.11 PINGREQ - Ping Request 72
3.11.1 PINGREQ Header 72
3.11.2 Packet Identifier 72
3.11.3 Client Identifier 73
3.11.4 PINGREQ Actions 73
3.12 PINGRESP - Ping Response 73
3.12.1 PINGRESP Header 74
3.12.2 Packet Identifier 74
3.12.3 Application Messages Remaining 74
3.13 DISCONNECT - Disconnect Notification 75
3.13.1 DISCONNECT Header 75
3.13.2 DISCONNECT Flags 76
3.13.2.1 Packet Identifier Flag 76
3.13.2.2 Session Expiry Interval Flag 76
3.13.2.3 Reason Code Flag 76
3.13.3 Packet Identifier 76
3.13.4 Reason Code 76
3.13.5 Session Expiry Interval 76
3.13.6 Reason String 77
3.13.7 DISCONNECT Actions 77
3.14 WAKEUP - Wake up request 77
3.14.1 WAKEUP Header 78
3.14.2 WAKEUP Actions 78
3.15 SLEEPREQ - Sleep request 78
3.15.1 SLEEPREQ Header 78
3.15.2 SLEEPREQ Flags 78
3.15.2.1 Retain Topic Aliases 78
3.15.3 Packet Identifier 79
3.15.4 Sleep Duration 79
3.15.5 SLEEPREQ Actions 79
3.16 SLEEPRESP - Sleep response 79
3.16.1 SLEEPRESP Header 80
3.16.2 Packet Identifier 80
3.17 Protection Encapsulation 81
3.17.1 Protection Encapsulation Header 82
3.17.2 Protection Flags 82
3.17.2.1 Monotonic Counter Length 82
3.17.2.2 Cryptographic Material Length 82
3.17.2.3 Authentication Tag Length 83
3.17.3 Protection Scheme 83
3.17.4 Sender Identifier 85

3.17.5 Random 85
3.17.6 Cryptographic Material 86
3.17.7 Monotonic Counter 86
3.17.8 Protected MQTT-SN Packet 86
3.17.9 Authentication Tag 86
3.18 Forwarder Encapsulation 87
3.18.1 Forwarder Encapsulation Header 87
3.18.2 Client Addressing Information 87
3.18.3 MQTT-SN Packet 87
3.19 Gateway Discovery Packets 88
3.19.1 ADVERTISE - Gateway Advertisement 88
3.19.1.1 ADVERTISE Header 88
3.19.1.2 Gateway Identifier 88
3.19.1.3 Duration 88
3.19.2 SEARCHGW - Search for A Gateway 88
3.19.2.1 SEARCHGW Header 89
3.19.2.2 Additional Network Information 89
3.19.3 GWINFO - Gateway Information 89
3.19.3.1 GWINFO Header 90
3.19.3.2 Gateway Identifier 90
3.19.3.3 Gateway Address 90
4 Operational behavior 91
4.1 Session state 91
4.1.1 Storing Session State 91
4.1.2 Session Establishment 92
4.2 Networks and Virtual Connections 93
4.3 Quality of Service levels and protocol flows 95
4.3.1 Publish without session 95
4.3.2 QoS 0: At most once delivery 95
4.3.3 QoS 1: At least once delivery 96
4.3.4 QoS 2: Exactly once delivery 97
4.4 Packet delivery retry 98
4.4.1 Virtual Connection End 99
4.4.2 Unacknowledged Packets 99
4.5 Application Message receipt 100
4.6 Application Message ordering 100
4.7 Topics 101
4.7.1 Topic Names and Topic Filters 101
4.7.1.1 Topic wildcards 101
4.7.1.1.1 Topic level separator 101
4.7.1.1.2 Multi-level wildcard 101
4.7.1.1.3 Single-level wildcard 102
4.7.1.2 Topics beginning with $ 102
4.7.1.3 Topic semantic and usage 103

4.7.2 Topic Aliases 104
4.7.2.1 Predefined Topic Aliases 104
4.7.2.2 Session Topic Aliases 104
4.8 Subscriptions 105
4.9 Flow Control 105
4.10 Server redirection 106
4.11 Authentication 107
4.11.1 CONNECT and AUTH packets 107
4.11.1.1 Re-authentication 109
4.11.1.2 MQTT User Name and Password Support 109
4.12 Handling errors 112
4.12.1 Malformed Packet and Protocol Errors 112
4.12.2 Other errors 113
4.13 Retained Messages 113
4.14 Client states 115
4.14.1 Session Timers 116
4.14.2 Sleeping Clients 117
4.15 Optional Features 119
5 Security (Informative) 121
5.1 Introduction 121
5.2 MQTT-SN solutions: security and certification 121
5.3 Lightweight cryptography and constrained devices 122
5.4 Implementation notes 122
5.4.1 Authentication of Clients by the Server 123
5.4.2 Authorization of Clients by the Server 123
5.4.3 Authentication of the Server by the Client 124
5.4.4 Integrity of Application Messages and MQTT-SN Control Packets 124
5.4.5 Privacy of Application Messages and MQTT-SN Control Packets 124
5.4.6 Non-repudiation of message transmission 124
5.4.7 Detecting compromise of Clients and Servers 125
5.4.8 Detecting abnormal behaviors 125
5.4.9 Handling of Disallowed Unicode code points 125
5.4.9.1 Considerations for the use of Disallowed Unicode code points 126
5.4.9.2 Interactions between Publishers and Subscribers 126
5.4.9.3 Remedies 127
5.4.10 Other security considerations 127
5.4.11 Use of SOCKS 127
5.4.12 Security profiles 128
5.4.12.1 Clear communication profile 128
5.4.12.2 Secured network communication profile 128
5.4.12.3 Secured transport profile 128
5.4.12.4 Industry specific security profiles 128
6 Conformance 129

Appendix A. Acknowledgments **130**
A.1 Special Thanks 130
A.2 Participants 130
Appendix B. Mandatory normative statements (informative) **131**
Appendix C. Implementation Guidance (Informative) **132**
C.1 Example MQTT-SN Architectures 132
C.1.1 Transparent Gateway 132
C.1.2 Aggregating Gateway 133
C.1.3 Forwarder 134
C.1.4 MQTT-SN Broker 135
C.2 Server Congestion 136
C.3 Example Timer and Counter Values 137
C.4 Exponential Backoff 138
C.5 Client State Diagrams 139
C.6 PUBLISH with QoS -1 141
C.6.1 PUBLISH Header 141
C.6.2 PUBLISH Flags 141
C.6.2.1 Topic Type 142
C.6.2.2 QoS 142
C.6.2.3 DUP 142
C.6.2.4 Retain 142
C.6.3 Topic Alias 142
C.6.4 Topic Short Name 142
C.6.5 Topic Name Length 142
C.6.6 Topic Name 142
C.6.7 Payload 143
C.6.8 PUBLISH with QoS -1 Actions 143
C.7 Gateway Advertisement and Discovery 144
Appendix D. Revision History (informative) **146**

1 Introduction

[All text is normative unless otherwise labeled]

1.0 Intellectual property rights policy

This specification is provided under the Non-Assertion Mode of the OASIS IPR Policy, 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, refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/mqtt/ipr.php).

1.1 Changes from earlier Versions

[Optional section.]
This section provides a description of significant differences from previously published, differently numbered Versions of this specification, if any. (Detailed revision history of this numbered Version should be tracked in an Appendix.)

1.1.1 MQTT-SN 1.2

Text describing the changes/differences

1.2 Organization of the MQTT-SN specification

The specification is split into six chapters:

- Chapter 1 – Introduction
- Chapter 2 – MQTT-SN Control Packet format
- Chapter 3 – MQTT-SN Control Packets
- Chapter 4 – Operational Behavior
- Chapter 5 - Security
- Chapter 6 – Conformance

1.3 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119], except where they appear in text that is marked as non-normative.

Datagram:

An independent, self-contained sequence of bytes. If received, the contents of a datagram must be correct.

Underlying Network:

The underlying network which provides the means to send datagrams from one network endpoint to another.

Network Address:

A unique label provided by the Underlying Network to identify a network endpoint.

To receive datagrams, an MQTT-SN Client or Server listens to the network for packets addressed to a specific Network Address.

Unicast Address:

A Network Address which represents one device on a network. For packets intended to reach one network endpoint.

Multicast Address:

A Network Address which represents all or groups of devices on a network. For packets intended for more than one network endpoint.

> **Informative:**
>
> Multicast Address as used in this specification also includes the concept of broadcast addresses, for brevity.

Network Identity:

The identity used to establish that a sequence of datagrams originates from the same network source. This could be, for example:

- A Network Address
- A DTLS connection ID
- An MQTT-SN Protection Packet Sender Identifier

Virtual Connection:

An MQTT-SN construct corresponding to the network connection in MQTT. It associates a Network Identity with a Session, by means of the Client Identifier.

Application Message:

The data carried by the MQTT-SN protocol across the network for the application. When an Application Message is transported by MQTT-SN it contains payload data, a Quality of Service (QoS), and a Topic Name.

Client:

A program or device that uses MQTT-SN. A Client:

- creates a Virtual Connection to a Server.
- publishes Application Messages that other Clients might be interested in.
- subscribes to request Application Messages that it is interested in receiving.
- unsubscribes to remove a request for Application Messages.
- deletes the Virtual Connection to the Server.

and/or:

- publishes Application Messages to a Multicast Address.

and/or:

- accepts Application Messages from a Multicast Address.

Server:

A program or device that acts as an intermediary between Clients which publish Application Messages and Clients which have made Subscriptions.

Also known as a **Gateway** - sometimes abbreviated to **GW**.

A Server:

- accepts CONNECT requests from Clients.
- accepts Application Messages published by Clients.
- processes Subscribe and Unsubscribe requests from Clients.
- forwards Application Messages that match Client Subscriptions.
- accepts DISCONNECT requests from connected Clients.

and/or

- accepts packets from a Multicast Address.

and/or

- opens an MQTT Network Connection to an MQTT Server.
- accepts Application Messages from the MQTT Server and forwards some or all to MQTT-SN Clients.
- accepts Application Messages from MQTT-SN Clients and forwards some or all to the MQTT Server.

and/or

- opens an MQTT Network Connection to an MQTT Server for every MQTT-SN CONNECT request

- forwards equivalent MQTT packets to the MQTT Server for each MQTT-SN packet received
- forwards equivalent MQTT-SN packets to the MQTT-SN Client for each MQTT packet received
- closes the MQTT Network Connection when the Virtual Connection is deleted

MQTT Client:

A program or device that uses MQTT. An MQTT Client:

- opens the Network Connection to the MQTT Server.
- publishes Application Messages that other MQTT (or MQTT-SN) Clients might be interested in.
- subscribes to request Application Messages that it is interested in receiving.
- unsubscribes to remove a request for Application Messages.
- closes the Network Connection to the Server.

MQTT Server:

A program or device that acts as an intermediary between MQTT Clients which publish Application Messages and MQTT Clients which have made Subscriptions.

Also known informally as an MQTT **Broker**.

An MQTT Server:

- accepts Network Connections from MQTT Clients.
- accepts Application Messages published by MQTT Clients.
- processes Subscribe and Unsubscribe requests from MQTT Clients.
- forwards Application Messages that match MQTT Client Subscriptions.
- closes the Network Connection from the MQTT Client.

Client Identifier:

A UTF-8 encoded character string which uniquely identifies every Client connecting to a Server.

Session:

A stateful interaction between a Client and a Server which is associated with a Client Identifier. Some Sessions last only as long as the Virtual Connection, others can span multiple consecutive Virtual Connections between a Client and a Server.

Session State:

The set of data that describes a Session. The Session State held by a Client is different to that held by a Server. See 4.1 Session state for details.

Subscription:

A Subscription comprises a Topic Filter and a maximum QoS. A Subscription is associated with a single Session. A Session can contain more than one Subscription. Each Subscription within a Session has a different Topic Filter.

Wildcard Subscription:

A Wildcard Subscription is a Subscription with a Topic Filter containing one or more wildcard characters. This allows the subscription to match more than one Topic Name. Refer to 4.7.1.1 Topic wildcards for a description of wildcard characters in a Topic Filter.

Topic Name:

A label attached to an Application Message which is matched against the Subscriptions known to the Server.

Topic Alias:

A Topic Alias is an integer value that is used to identify the Topic instead of using the Topic Name. This reduces Packet sizes, and is useful when the Topic Names are long and the same Topic Names are used repetitively within a Virtual Connection.

Topic Filter:

An expression contained in a Subscription to indicate an interest in one or more topics. A Topic Filter can include wildcard characters and can match more than one Topic Name.

MQTT-SN Control Packet:

A packet of information that is sent to a Network Address.

Malformed Packet:

A Control Packet that cannot be parsed according to this specification. Refer to 4.12 Handling errors for information about error handling.

Protocol Error:

An error that is detected after the packet has been parsed and found to contain data that is not allowed by the protocol or is inconsistent with the state of the Client or Server. Refer to 4.12 Handling errors for information about error handling.

Will Message:

An Application Message which is published by the Server after the Virtual Connection is deleted in cases where the Virtual Connection is not deleted normally. Refer to 3.1.3 Will Flags for information about Will Messages.

Retained Message:

An Application Message which is stored by the Server for a topic on the receipt of a Publish Packet with the retained flag set. When a Client subscribes to a topic with a Retained Message set, the Server sends the Retained Message to the Client, depending on the setting of the Retain Handling Subscribe Flags. Refer to 3.7.2 SUBSCRIBE Flags and 4.13 Retained Messages for more information about Retained Messages.

Disallowed Unicode code point:

The set of Unicode Control Codes and Unicode Noncharacters which should not be included in a UTF-8 Encoded String. Refer to 1.7.4 UTF-8 Encoded String for more information about the Disallowed Unicode code points.

1.4 Normative references

[Required section.]

This appendix contains the normative and informative references that are used in this document.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

Note: Any normative work cited in the body of the text as needed to implement the work product must be listed in the Normative References section below. Each reference to a separate document or artifact in this work must be listed here and must be identified as either a Normative or an Informative Reference.

For all References – Normative and Informative:

Recommended approach: Set up **[Reference]** label elements as "Bookmarks", then create hyperlinks to them within the document at locations from which the references are cited. Citations in the body of the text should be hyperlinked to the appropriate Reference entry, not directly to targets which are not a part of this Work Product.

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 (italicized). Edited by Albert Alston, Bob Ballston, and Calvin Carlson. Approval date (DD Month YYYY). OASIS Stage Identifier and Revision Number (e.g., OASIS Committee Specification Draft 01). Principal URI (stage-specific URI, e.g., with stage component: somespec-v1.0-csd01.html). Latest stage: (static URI, without stage identifiers, used as a symbolic link to most recently published stage of this Version).

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.
The most recent IETF RFC references are listed by the IETF at
https://www.rfc-editor.org/in-notes/rfc-ref.txt.
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.

[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, <https://www.rfc-editor.org/info/rfc8174>.

[RFC3629]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003,

http://www.rfc-editor.org/info/rfc3629

[RFC6455]

Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011,

http://www.rfc-editor.org/info/rfc6455

[Unicode]

The Unicode Consortium. The Unicode Standard,

http://www.unicode.org/versions/latest/

1.5 Informative 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]

1.6 MQTT For Sensor Networks (MQTT-SN)

Sensor Networks are simple, low cost and easy to deploy. They are typically used to provide event detection, monitoring, automation, process control and more. Sensor Networks often comprise many battery-powered sensors and actuators, each containing a limited amount of storage and processing capability. They usually communicate wirelessly.

Sensor Networks are typically self-forming, continually changing, and do not have any central control. The wireless network connections and processing nodes will fail, and the batteries will run out. The nodes will be replaced, added or removed in an unplanned way. The identities of the devices are usually created when they are manufactured, this avoids the need for specialist configuration when they are deployed. Applications running outside the Sensor Network do not need to know the details of the devices in it. The applications consume information from the sensors and send instructions to actuators based only on labels created by the application designers. The labels are called Topic Names in the MQTT and MQTT-SN protocols. The MQTT-SN implementation carries information between a set of applications and the correct set of devices based on its knowledge of the network and the applications designer’s choice of Topic Names.

Consider an example of a medicine tracking application. The application needs to know the location and temperature of the medicine, but it does not want to concern itself with the network details of the devices providing the data. It may be that the number and types of the devices changes over time. There may also be other applications using the same sensor data for other purposes. The model is that the devices and applications produce and consume data addressed by the Topics rather than the other devices and applications.

This MQTT-SN specification is a variant of the MQTT version 5 specification. It is adapted to exploit low power and low bandwidth wireless networks. Low power wireless radio links typically have higher numbers of transmission errors compared to more powerful networks because they are more susceptible to interference and fading of the radio signals. They also have lower transmission rates.

For example, wireless networks based on the IEEE 802.15.4 standard used by Zigbee have a maximum bandwidth of 250 kbit/s in the 2.4 GHz band. To reduce transmission errors the packets are kept short. The maximum packet length at the physical layer is 128 bytes and half of these may be used for Media Access Control and security.

The MQTT-SN protocol is optimized for implementation on low-cost, battery-operated devices with limited processing and storage resources. The capabilities are kept simple and the specification allows partial implementations.

1.6.1 Differences Between MQTT-SN and MQTT

To facilitate interoperation MQTT-SN is similar in many ways to MQTT, but the two are independent of each other.

MQTT-SN can work isolated from other networks or in conjunction with MQTT. The main differences between MQTT-SN and MQTT are:

1. In addition to Topic Alias and long Topic Names MQTT-SN allows Predefined Topic Aliases.
2. Support for sleeping clients allows battery operated devices to enter a low power mode. In this state, Application Messages for the Client are buffered by the Server and delivered when the client wakes.
3. A new Quality of Service level (WITHOUT SESSION) is introduced in MQTT-SN, allowing devices to publish without a session having been established.
4. MQTT-SN has fewer requirements on the underlying transport and it can use connectionless network transports such as User Datagram Protocol (UDP).
5. MQTT-SN introduces the PROTECTION packet for packet-based security based on symmetric-key cryptography.
6. If the network supports sending messages to more than one recipient at once, Gateway Advertisement and Discovery can be implemented.

1.7 Data representation

1.7.1 Bits (Byte)

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

1.7.2 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).

1.7.3 Four Byte Integer

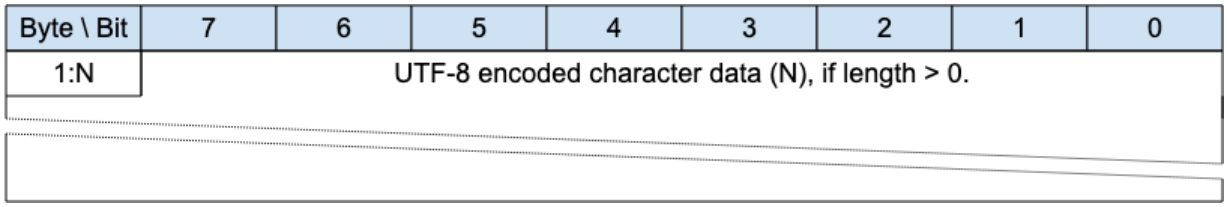
Four Byte Integer data values are 32-bit unsigned integers in big-endian order: the high order byte precedes the successively lower order bytes. This means that a 32-bit word is presented on the network as Most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by the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

1.7.4 UTF-8 Encoded String

Text fields within the MQTT-SN Control Packets are encoded as fixed length UTF-8 strings. UTF-8 [RFC3629] is an efficient encoding of Unicode [Unicode] characters that optimizes the encoding of ASCII characters in support of text-based communications.

Unless stated otherwise all variable length 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



The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular, the character data MUST NOT include encodings of code points between U+D800 and U+DFFF [MQTT-SN-1.7.4-1].

If the Client or Server receives an MQTT-SN Control Packet containing ill-formed UTF-8 it is a Malformed Packet. Refer to 4.12 Handling errors for information about handling errors.

A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000 [MQTT-SN-1.7.4-2]. If a receiver (Server or Client) receives an Control Packet containing U+0000 in a UTF-8 Encoded String it is a Malformed Packet.

UTF-8 Encoded Strings SHOULD NOT include the Unicode [Unicode] code points listed below. If a receiver (Server or Client) receives an MQTT-SN Control Packet with UTF-8 Encoded Strings containing any of them it MAY treat it as a Malformed Packet. These are the Disallowed Unicode code points.

- U+0001..U+001F control characters
- U+007F..U+009F control characters
- Code points defined in the Unicode specification [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-SN-1.7.4-3].

Informative example

For example, the string A鼸 which is LATIN CAPITAL Letter A followed by the code point U+2A6D4 (which represents a CJK IDEOGRAPH EXTENSION B character) is encoded as follows:

Figure 1-2 – Fixed Length UTF-8 Encoded String informative example

Byte \ Bit	7	6	5	4	3	2	1	0
	‘A’ (0x41)							
1	0	1	0	0	0	0	0	1
	'ð' (0xF0)							
2	1	1	1	1	0	0	0	0
	'ª' (0xAA)							
3	1	0	1	0	1	0	1	0
	'›' (0x9B)							
4	1	0	0	1	1	0	1	1
	'”' (0x94)							
5	1	0	0	1	0	1	0	0

2 MQTT-SN Control Packet format

2.1 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 2-1 — 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

2.1.1 Packet Header

Each MQTT-SN Control Packet contains a Header of format 1 or format 2 as shown below.

Figure 2-2 – Packet Header Format 1

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	MQTT-SN Control Packet Type							

Figure 2-3 – Packet Header Format 2

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length 0x01							
2	Length MSB							
3	Length LSB							
4	MQTT-SN Control Packet Type							

2.1.2 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 65,535 bytes. It is more efficient to use the shorter 1-byte format for packets with lengths up to and including 255 bytes.

A Client or Server receiving MQTT-SN control packets MUST be able to process both 1-byte and 3-byte length formats. [MQTT-SN-2.1.2-1]

Informative 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.

2.1.3 MQTT-SN Control Packet Type

The MQTT-SN Control Packet Type field is a 1-byte unsigned value, the values are shown below.

Figure 2-4 – MQTT-SN Control Packet Types

Name	Value	Direction of flow	Description
Reserved	0x00	Forbidden	Reserved
CONNECT	0x01	Client to Server	Virtual Connection request
CONNACK	0x02	Server to Client	Virtual Connection acknowledgement
PUBLISH	0x03	Client to Server or Server to Client	Publish message
PUBACK	0x04	Client to Server or Server to Client	Publish acknowledgment (QoS 1) or Publish error (Any QoS).
PUBREC	0x05	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	0x06	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	0x07	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	0x08	Client to Server	Subscribe request
SUBACK	0x09	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	0x0A	Client to Server	Unsubscribe request
UNSUBACK	0x0B	Server to Client	Unsubscribe acknowledgment

PINGREQ	0x0C	Client to Server	PING request
PINGRESP	0x0D	Server to Client	PING response
DISCONNECT	0x0E	Client to Server or Server to Client	Disconnect notification
AUTH	0x0F	Client to Server or Server to Client	Authentication handshake
REGISTER	0x10	Client to Server	Request topic alias
REGACK	0x11	Server to Client	Supply topic alias
PUBWOS	0x12	Client to Server or Server to Client	Publish packet for out of session messages which have no session on the receiver
SLEEPREQ	0x13	Client to Server	Sleep request
SLEEPRESP	0x14	Server to Client	Sleep response
WAKEUP	0x15	Server to Client	Wake up request
ADVERTISE	0x16	Server to Clients	Advertise the Server presence
SEARCHGW	0x17	Client to Servers	Client GWINFO request
GWINFO	0x18	Server to Client	Response to a SEARCHGW
Reserved	0x19-0xFD	Forbidden	Reserved
FORWARDER ENCAPSULATION	0xFE	Forwarder to Client or Forwarder to Server	Encapsulated MQTT-SN packet
PROTECTION ENCAPSULATION	0xFF	Client to Server or Server to Client	A protection envelope that can encapsulate any MQTT-SN packet with the exception of Forwarder-Encapsulation packet (0xFE)

2.2 Packet Identifier

The Variable Header component of many of the MQTT-SN Control Packet types includes a Two Byte Integer Packet Identifier field. MQTT-SN Control Packets that require a Packet Identifier are shown in Figure 2-5.

Figure 2-5 – Packets with Packet Identifier

MQTT-SN Control Packet	Packet Identifier field
ADVERTISE	NO
AUTH	YES
CONNACK	YES
CONNECT	YES
DISCONNECT	OPTIONAL
FORWARDER ENCAPSULATION	NO
GWINFO	NO
PINGREQ	YES
PINGRESP	YES
PROTECTION ENCAPSULATION	NO
PUBACK	YES
PUBCOMP	YES
PUBLISH	YES (If QoS > 0)
PUBREC	YES
PUBREL	YES
PUBWOS	NO
REGACK	YES
REGISTER	YES
SEARCHGW	NO
SLEEPREQ	YES

SLEEPRESP	YES
SUBACK	YES
SUBSCRIBE	YES
UNSUBACK	YES
UNSUBSCRIBE	YES
WAKEUP	NO

Each time a Client sends a new MQTT-SN Control Packet which is identified in the above table as requiring a Packet Identifier, it MUST assign it a non-zero Packet Identifier that is currently unused.

A PUBLISH packet MUST NOT contain a Packet Identifier if its QoS value is set to 0.

Each time a Server sends a new PUBLISH (with QoS > 0) MQTT-SN Control Packet it MUST assign it a non zero Packet Identifier that is currently unused.

Packet Identifiers used with PUBLISH, SUBSCRIBE and UNSUBSCRIBE packets form a single, unified set of identifiers separately for the Client and the Server in a Session. A Packet Identifier cannot be used by more than one Packet at any time.

The Packet Identifier becomes available for reuse after the sender has processed the corresponding acknowledgement packet, defined as follows. In the case of a QoS 1 PUBLISH, this is the corresponding PUBACK; in the case of QoS 2 PUBLISH it is PUBCOMP or a PUBREC with a Reason Code of 0x80 or greater. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK.

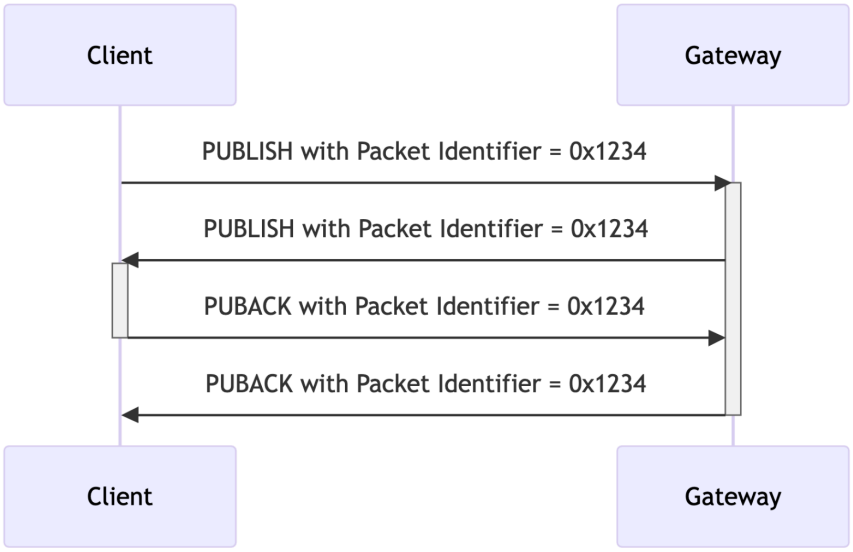
A PUBACK, PUBREC , PUBREL, or PUBCOMP packet MUST contain the same Packet Identifier as the PUBLISH packet that was originally sent. A SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE packet respectively.

The Client and Server assign Packet Identifiers independently of each other. As a result, Client-Server pairs can participate in concurrent Packet exchanges using the same Packet Identifiers.

Informative comment

It is possible for a Client to send a PUBLISH packet with Packet Identifier 0x1234 and then receive a different PUBLISH packet with Packet Identifier 0x1234 from its Server before it receives a PUBACK for the PUBLISH packet that it sent.

Figure 2-6 - Publishes with the same Packet Identifier



2.3 Reason Code

A Reason Code is a one byte unsigned value that indicates the result of an operation. Reason Codes less than 0x80 indicate successful completion of an operation. The normal Reason Code for success is 0x00. Reason Code values of 0x80 or greater indicate failure.

The Reason Codes share a common set of values as shown below.

Figure 2-7 – Reason Codes

Identifier		Name	Packets	Description
Dec	Hex			
0	0x00	Success	CONNACK, SUBACK, UNSUBACK, REGACK, PUBACK, PUBREC, PUBREL, PUBCOMP, AUTH (server only)	The operation was successful.
0	0x00	Normal disconnection	DISCONNECT	Delete the Virtual Connection normally. Do not send the Will Message.
0	0x00	Granted QoS 0	SUBACK	The subscription is accepted and the maximum QoS sent will be QoS 0. This might be a lower QoS than was requested.

1	0x01	Granted QoS 1	SUBACK	The subscription is accepted and the maximum QoS sent will be QoS 1. This might be a lower QoS than was requested.
2	0x02	Granted QoS 2	SUBACK	The subscription is accepted and any received QoS will be sent to this subscription.
4	0x04	Disconnect with will message	DISCONNECT (client only)	The Client wishes to disconnect but requires that the Server also publishes its Will Message.
16	0x10	No matching subscribers	PUBACK, PUBREC	The Application Message is accepted but there are no subscribers. If the Server knows that there are no matching subscribers, it MAY use this Reason Code instead of 0x00 (Success).
17	0x11	No subscription existed	UNSUBACK	No matching Topic Filter is being used by the Client.
24	0x18	Continue authentication	AUTH	Continue the authentication with another step.
25	0x19	Re-authenticate	AUTH (client only)	Initiate a re-authentication.
26	0x1A	Topic Alias Exists	REGACK	A Session Topic Alias was requested, but a Session or Predefined Topic Alias already exists. (MQTT-SN only)
128	0x80	Unspecified error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT	The receiver does not accept the request but either does not want to reveal the reason, or it does not match one of the other values.

129	0x81	Malformed packet	CONNACK, DISCONNECT	The received packet does not conform to this specification.
130	0x82	Protocol error	CONNACK, DISCONNECT	An unexpected or out of order packet was received.
131	0x83	Implementation specific error	CONNACK, PUBACK, PUBREC, REGACK, SUBACK, UNSUBACK, DISCONNECT	The packet received is valid but cannot be processed by this implementation.
132	0x84	Unsupported Protocol Version	CONNACK	The Server does not support the version of the MQTT or MQTT-SN protocol requested by the Client.
133	0x85	Client identifier not valid	CONNACK	The Client Identifier is a valid string but is not allowed by the Server.
134	0x86	Bad user name or password	CONNACK	The Server does not accept the User Name or Password specified by the Client
135	0x87	Not authorized	CONNACK, PUBACK, PUBREC, REGACK, SUBACK, UNSUBACK, DISCONNECT (server only)	The request is not authorized.
136	0x88	Server unavailable	CONNACK	The MQTT-SN Server is not available or, in the case of a Transparent gateway, the MQTT server is not available.
137	0x89	Server busy	CONNACK, DISCONNECT (server only)	The Server is busy and cannot continue processing requests from this Client.
138	0x8A	Banned	CONNACK	This Client has been banned by administrative action. Contact the server administrator.

139	0x8B	Server shutting down	DISCONNECT (server only)	The Server is shutting down.
140	0x8C	Bad authentication method	CONNACK, DISCONNECT	The authentication method is not supported or does not match the authentication method currently in use.
141	0x8D	Keep alive timeout	DISCONNECT (server only)	The Connection is closed because no packet has been received for 1.5 times the Keepalive time.
142	0x8E	Session taken over	DISCONNECT (server only)	Another Connection using the same Client Identifier has connected causing this Connection to be closed.
143	0x8F	Topic filter invalid	SUBACK, UNSUBACK, DISCONNECT (server only)	The Topic Filter is correctly formed, but is not accepted by this Server.
144	0x90	Topic name invalid	CONNACK, PUBACK, PUBREC, DISCONNECT (server only)	The Topic Name is correctly formed, but is not accepted by this Client or Server.
145	0x91	Packet identifier in use	PUBACK, PUBREC, SUBACK, UNSUBACK, REGACK, PINGRESP, SLEEPRESP	The specified Packet Identifier is already in use.
146	0x92	Packet identifier not found	PUBREL, PUBCOMP	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session State on the Client and Server.
147	0x93	Receive maximum exceeded	DISCONNECT	The Client or Server has received more than Receive Maximum publication for which it has not sent PUBACK or PUBCOMP.

148	0x94	Topic alias invalid	DISCONNECT (server only)	The Client or Server has received a PUBLISH packet containing a Topic Alias which is greater than the Maximum Topic Alias it sent in the CONNECT or CONNACK packet. (Transparent gateway only)
149	0x95	Packet too large	CONNACK, DISCONNECT	The packet size is greater than Maximum Packet Size for this Client or Server.
150	0x96	Packet rate too high	DISCONNECT	The received data rate is too high.
151	0x97	Quota exceeded	REGACK, SUBACK, DISCONNECT	An implementation or administrative imposed limit has been exceeded.
152	0x98	Administrative action	DISCONNECT	The Virtual Connection is deleted due to an administrative action.
153	0x99	Payload format invalid	PUBACK, PUBREC, DISCONNECT (server only)	The MQTT payload format does not match the one specified by the Payload Format Indicator. (Transparent gateway only)
154	0x9A	Retain not supported	CONNACK, DISCONNECT (server only)	The MQTT Server does not support retained messages. (Transparent gateway only)
155	0x9B	QoS not supported	CONNACK, DISCONNECT (server only)	The Client specified a QoS greater than the QoS specified in a Maximum QoS in the MQTT CONNACK. (Transparent gateway only)
156	0x9C	Use another server	CONNACK, DISCONNECT (server only)	The Client should temporarily change its Server.
157	0x9D	Server moved	CONNACK, DISCONNECT (server only)	The Server is moved and the Client should

				permanently change its server location.
158	0x9E	Shared subscription not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Shared Subscriptions. (Transparent gateway only)
159	0x9F	Connection rate exceeded	CONNACK, DISCONNECT (server only)	This Virtual Connection is deleted because the connection rate is too high.
160	0xAD	Maximum connect time	DISCONNECT (server only)	The maximum connection time authorized for this Virtual Connection has been exceeded.
161	0xA1	Subscription identifiers not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Subscription Identifiers; the subscription is not accepted. (Transparent gateway only)
162	0xA2	Wildcard subscription not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Wildcard Subscriptions; the subscription is not accepted. (Transparent Gateway only)
230	0xE6	Only PROTECTION packet supported (Note 1)	Any packet except PROTECTION and Forwarder Encapsulation	The Receiver was expecting a packet to be Protection Encapsulated. (MQTT-SN only)
231	0xE7	Protection scheme invalid	DISCONNECT	Specific to MQTT-SN
232	0xE8	Unknown Sender Id	DISCONNECT	Specific to MQTT-SN
240	0xF0	Unknown Topic Alias	PUBACK, PUBREC, SUBACK, UNSUBACK, REGACK	Specific to MQTT-SN

241	0xF1	Congestion	SUBACK, REGACK, CONNACK, PUBACK, PUBREC	Try again later. See C.3 Server Congestion (MQTT-SN only)
242	0xF2	Protection packet not supported	DISCONNECT	Specific to MQTT-SN
243	0xF3	Forwarder Encapsulation not supported	DISCONNECT	Specific to MQTT-SN
244	0xF4	No Virtual Connection exists	DISCONNECT	Specific to MQTT-SN
245 - 255	0xF5 - 0xFF	Reserved for MQTT-SN		Specific to MQTT-SN

Note(s):

1. It is used by a receiver to indicate that it expected a packet to be protected and it wasn't.
2. The MQTT-SN dedicated range of reason codes is from 0xE6 (230) to 0xFF(255).

2.4 Topic Types

Several packets refer to a Topic Type in their flags. This is a 2-bit field which determines the format of the topic value. The allowable values are as follows:

Figure 2-8 – Topic Types

	Topic Type Value	Name	Description
0	0b00	Session Topic Alias	A session Topic Alias is negotiated between the Server and Client within the scope of a session.
1	0b01	Predefined Topic Alias	A predefined Topic Alias is known statically by both the Server and the Client outside the scope of a session. No negotiation is required since both entities have knowledge of the topic alias mapping.

2	0b10		Reserved
3	0b11	Topic Name or Filter	A Topic Name or Topic Filter, which requires no session negotiation.

Predefined and Session Topic Aliases are assigned from different pools so there is no danger of collision.

Refer to 4.7 Topics for detailed descriptions of Topic Names and Topic Aliases.

3 MQTT-SN Control Packets

3.1 CONNECT - Connection Request

Figure 3-4 – CONNECT Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Connect Flags								
	Default Awake Messages				Sess Exp	Auth	Will	Clean Start
3	X	X	X	X	X	X	X	X
Will Flags *(F=1 or 0) - only present when Will flag is set*								
	Reserved			Wil Retain	Will QoS		Will Topic Type	
4	0	0	0	X	X	X	X	X
4+F	Packet Identifier MSB							
5+F	Packet Identifier LSB							
6+F	Protocol Version							
7+F	Keep Alive MSB							
8+F	Keep Alive LSB							
9+F	Maximum Packet Size MSB							
10+F	Maximum Packet Size LSB							
Session Expiry Interval *(S=F+4 or F) - only present when Sess Exp flag is set*								
11+F	Session Expiry Interval MSB							
12+F	Session Expiry Interval							
13+F	Session Expiry Interval							
14+F	Session Expiry Interval LSB							
Will Fields *(W=4+N+P+S or S) - only present when Will flag is set*								
11+S	Will Topic Alias or Will Topic Name Length MSB							
12+S	Will Topic Alias or Will Topic Name Length LSB							
13+S	Will Topic Name *(N) - only present when Will Topic Type is Topic Name*							
13+S+N	Will Payload Length MSB							
14+S+N	Will Payload Length LSB							
15+S+N	Will Payload *(P)*							
Authentication Fields *(A=3+M+D+W or W) - only present when Auth flag is set*								
11+W	Authentication Method Length							
12+W	Authentication Method (M)							
12+W+M	Authentication Data Length MSB							
13+W+M	Authentication Data Length LSB							
14+W+M	Authentication Data (D)							
11+A	Client Identifier (I)							

The CONNECT packet is sent from the Client to the Server to request the creation of or continuation of a Session.

3.1.1 CONNECT Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format.Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.1.2 Connect Flags

The Connect Flags is 1 byte field which contains several parameters specifying the behavior of the MQTT-SN Virtual Connection. It also indicates the presence or absence of fields in the Packet.

The Server MUST validate that the reserved flags in the CONNECT packet are set to 0 [MQTT-SN-3.4.2-1]. If any of the reserved flags is not 0 it is a Malformed Packet. Refer to 4.12 Handling errors for information about handling errors.

3.1.2.1 Clean Start

Position: bit 0 of the Connect Flags byte.
This flag specifies whether the Virtual Connection starts a new Session or is a continuation of an existing Session. Refer to 4.1 Session state for a definition of the Session State.

If a CONNECT packet is received with Clean Start is set to 1, the Client and Server MUST discard any existing Session and start a new Session [MQTT-SN-3.4.2.1-1]. Consequently, the Session Present flag in CONNACK is always set to 0 if Clean Start is set to 1.

If a CONNECT packet is received with Clean Start set to 0 and there is a Session associated with the Client Identifier, the Server MUST resume communications with the Client based on state from the existing Session [MQTT-SN-3.4.2.1-2]. If a CONNECT packet is received with Clean Start set to 0 and there is no Session associated with the Client Identifier, the Server MUST create a new Session [MQTT-3.4.2.1-3].

3.1.2.2 Will Flag

Position: bit 1 of the Connect Flags byte.

If the Will Flag is set to 1, the Will Flags, Will Topic, and Will Payload fields MUST be present in the Packet [MQTT-SN-3.4.2.2-1].

If the Will Flag is set to 1 this indicates that a Will Message MUST be stored on the Server and associated with the Session [MQTT-SN-3.4.2.2-2]. The Will Message consists of the Will Topic, and Will Payload fields in the CONNECT Packet. The Will Message MUST be published after the Virtual Connection is deleted or the Session ends, unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) [MQTT-SN-3.4.2.2-3].

Situations in which the Will Message is published include, but are not limited to:

- The Server deletes the Virtual Connection as a result of an I/O error or network failure it has detected.
- The Client fails to communicate within the Keep Alive time.
- The Server deletes the Virtual Connection because of a protocol error.
- The Server deletes the Virtual Connection because of a Retry timeout.

The Will Message MUST be removed from the stored Session State in the Server once it has been published or the Server has received a DISCONNECT packet with a Reason Code of 0x00 (Normal disconnection) from the Client [MQTT-SN-3.4.2.2-4].

The Server SHOULD publish Will Messages promptly after the Virtual Connection is deleted or the Session ends, whichever occurs first. In the case of a Server shutdown or failure, the Server MAY defer publication of Will Messages until a subsequent restart. If this happens, there might be a delay between the time the Server experienced failure and when the Will Message is published.

3.1.2.3 Authentication Flag

Position: bit 2 of the Connect Flags byte. Labelled *Auth* in Figure 3-4.

If the Authentication Flag is set to 1, the Authentication Method and Authentication Data fields MUST be present in the Packet [MQTT-SN-3.4.2.3-1].

3.1.2.4 Session Expiry Flag

Position: bit 3 of the Connect Flags byte. Labelled *Sess Exp* in Figure 3-4.

If the Session Expiry Flag is set to 1, the Session Expiry Interval field MUST be present in the Packet [MQTT-SN-3.4.2.4-1].

If the Session Expiry Flag is set to 0, the Session Expiry Interval field MUST NOT be present in the Packet [MQTT-SN-3.4.2.4-2].

3.1.2.5 Default Number of Awake Messages

Position: bits 7 through 4 of the Connect Flags byte. Labelled *Default Awake Messages* in Figure 3-4.

A value between 0-15 to indicate the maximum number of messages a Client shall receive during an AWAKE session. Specifying 0 will mean it is up to the Server to determine how many messages it will send, which may be unbounded.

3.1.3 Will Flags

If the Will Flag is set to 0, the Will Flags MUST NOT be present in the Packet [MQTT-SN-3.4.3-1].

If the Will Flag is set to 1, the Will Flags MUST be present in the Packet [MQTT-SN-3.4.3-2].

The *Will Flags* is 1 byte field which contains several parameters specifying the handling of the Will Message.

3.1.3.1 Will Topic Type

Position: bits 1 and 0 of the Will Flags byte.

This is a 2-bit field which determines the format of the topic value. Refer to 2.4 Topic Types for the definition of the various topic types.

3.1.3.2 Will QoS

Position: bits 3 and 2 of the Will Flags byte.

These two bits specify the QoS level to be used. The value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). A value of 3 (0x03) is a Malformed Packet.

3.1.3.3 Will Retain

Position: bit 4 of the Will Flags byte.

This specifies if the Will Message is to be retained when it is published. See 4.13 Retained Messages for more information about Retained Messages.

If the Will Flag is set to 1 and Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message [MQTT-SN-3.4.3.3-1].

If the Will Flag is set to 1 and Will Retain is set to 1, the Server MUST publish the Will Message as a retained message [MQTT-SN-3.4.3.3-2].

3.1.4 Packet Identifier

Used to identify the corresponding CONNACK or AUTH packet. It should ideally be populated with a random integer value.

3.1.5 Protocol Version

The one-byte unsigned value that represents the revision level of the protocol used by the Client.

Figure 3-5 – Protocol Versions

Protocol Version	Value
Version 1.2	0x01
Version 2.0	**0x02**
Reserved for future use	0x03 – 0xFF

The value of the Protocol Version field for MQTT-SN version 2.0 MUST be 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).

3.1.6 Keep Alive

The Keep Alive is a Two Byte Integer greater than 0 (1 - 65,535), 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-SN Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other MQTT-SN Control Packets, the Client MUST send a PINGREQ packet [MQTT-SN-3.4.6-1].

Informative comment

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 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 delete the Virtual Connection and move the Client to the Disconnected state (see 4.14 Client states) [MQTT-SN-3.4.6-2].

If a Client does not receive a PINGRESP packet within a *Retry Interval* amount of time after it has sent a PINGREQ, it SHOULD retry the transmission according to 4.4.2 Unacknowledged Packets up to the maximum number of attempts. If a PINGRESP is still not received it MUST delete the Virtual Connection to the Server by way of a DISCONNECT, with the understanding that the Server may no longer be reachable.

Informative Comment

Unlike MQTT, the MQTT-SN Keep Alive timeout can not be turned off (by setting a value of 0). This is because there is no other indication in MQTT-SN of a connection failure, as there is in MQTT with the underlying TCP/IP connection.

The Keep Alive must have a value greater than 0. It is a protocol error if a Keep Alive value of 0 or below is set.

Informative 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.

Informative 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.

Informative Comment

Clients can use the Keep Alive procedure to supervise the liveliness of the Server to which they are connected. If a Client does not receive a PINGRESP from the Server even after multiple retransmissions of the PINGREQ packet and deletes the Virtual Connection, it might try to connect to another Server before trying to reconnect to this Server. Note that because the Keep Alive timers are not synchronized between Clients, in case of a Server failure there is a low risk of a storm of CONNECT packets sent almost at the same time by all affected clients towards a new Server.

3.1.7 Maximum 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.

Informative 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-SN Control Packet, as defined in 2.1 Structure of an MQTT-SN Control Packet. 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. 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).

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.

Informative comment

Where a packet is discarded without being sent, the Server could take some diagnostic action including alerting the Server administrator. Such actions are outside the scope of this specification.

3.1.8 Session Expiry Interval

The Session Expiry Interval is a four-byte integer time interval measured in seconds. If the Session Expiry Interval is set to 0, the Session ends when the Virtual Connection is deleted by the Client or Server.

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 Virtual Connection is deleted if the Session Expiry Interval is greater than 0.

Informative 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.

Informative comment

The client and Server between them should negotiate a reasonable and practical session expiry interval according to the network and infrastructure environment in which they are deployed. For example, it would not be practical to set a session expiry interval of many months on a Server whose hardware is only capable of storing a few client sessions.

Informative comment

A Client that only wants to process messages while connected will set Clean Start to 1 and set the Session Expiry Interval to 0. It will not receive Application Messages

published before it is connected and has to subscribe afresh to any topics that it is interested in each time it connects.

Informative comment
The Client should always use the Session Present flag in the CONNACK to determine whether the Server has a Session State for this Client.

3.1.9 Will Topic Alias or Will Topic Name Length

If the Will Flag is set to 1, the Will Topic Alias or Will Topic Name Length is the next field in the Packet. In both cases, this is two bytes.
In the case of Will Topic Type being Topic Name, this field will refer to the length of the Will Topic Name field. In the other cases, this will be the value used as the Will Topic Alias.

3.1.10 Will Topic Name

If the Will Flag is set to 1 and the Will Topic Type is set to Topic Name (0b11), the Will Topic Name is the next field in the Packet. The Will Topic Name MUST be a UTF-8 Encoded String as defined in 1.7.4 UTF-8 Encoded String [MQTT-SN-3.4.10-1].

3.1.11 Will Payload Length

If the Will Flag is set to 1, the Will Payload Length is the next field in the Packet. It contains the length of the Will Payload field.

3.1.12 Will Payload

If the Will Flag is set to 1, the Will Payload is the next field in the Packet. The Will Payload defines the Application Message Payload that is to be published to the Will Topic as described in 3.1.2.2 Will Flag. This field consists of Binary Data.

3.1.13 Authentication Method Length

If the Auth Flag is set to 1, the Authentication Method Length is the next field in the Packet. It is a single byte value (max 0-255 bytes), representing the length of the field used to specify the authentication method. Refer to 4.11 Authentication for more information about authentication.

3.1.14 Authentication Method

If the Auth Flag is set to 1, the Authentication Method is the next field in the Packet. It is a UTF-8 Encoded String containing the name of the Authentication Method.

To support the equivalent of the MQTT User Name and Password fields in the CONNECT packet, see 4.11.1.2 MQTT User Name and Password Support.

Refer to 4.11 Authentication for more information about authentication.

3.1.15 Authentication Data Length

If the Auth Flag is set to 1, the Authentication Data Length is the next field in the Packet. It is a two byte value (max 0-65535 bytes), representing the length of the field used to specify the authentication data. Refer to 4.11 Authentication for more information about authentication.

3.1.16 Authentication Data

If the Auth Flag is set to 1, the Authentication Data is the next field in the Packet.

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

To support the equivalent of the MQTT User Name and Password CONNECT packet fields, see 4.11.1.2 MQTT User Name and Password Support.

Refer to 4.11 Authentication for more information about authentication.

3.1.17 Client Identifier

The Client Identifier MUST be a UTF-8 Encoded String. This field is optional - its existence or absence is inferred from the Packet length.

The Client Identifier identifies the Client to the Server. Each Client connecting to the Server has a unique Client Identifier. The Client Identifier MUST be used by Clients and by Server to identify the state that they hold relating to this MQTT-SN Session between the Client and the Server.

> **Informative comment**
>
> A Client Identifier can be between 0 - 65,521 bytes. It is recommended for practicality, Client Identifiers are restricted to a reasonable size (less than 243 bytes to fit within a small CONNECT packet).

When the Client Identifier is present (greater than 0 bytes), the Server MUST allow values which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".

The Server MAY choose to allow more than 23 bytes.

3.1.18 CONNECT Actions

Note that a Server MAY support multiple protocols on the same network endpoint. If the Server determines that the protocol is MQTT-SN 2.0 then it validates the connection attempt as follows.

1. The Server MUST validate that the CONNECT packet matches the format described in 3.1 CONNECT and MUST NOT create a Virtual Connection for this CONNECT if it does not match [MQTT-SN-3.4.18-1]. The Server MAY send a CONNACK with a Reason Code of 0x80 or greater as described in 4.12 Handling errors.

2. The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST NOT create a Virtual Connection for this CONNECT [MQTT-SN-3.4.18-2]. It MAY send an appropriate CONNACK response with a Reason Code of 0x80 or greater as described in 3.5 CONNACK and 4.12 Handling errors.

If validation is successful, the Server performs the following steps.

1. If the Client Identifier represents a Client already connected to the Server, the Server sends a DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) as described in 4.12 Handling errors and MUST delete the Virtual Connection of the existing Client [MQTT-SN-3.4.18-3]. If the existing Client has a Will Message, that Will Message is published as described in 3.4.3 Will Flags.
2. The Server MUST perform the processing of Clean Start that is described in 4.14.2 Clean start [MQTT-SN-3.4.18-4].
3. The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Reason Code [MQTT-SN-3.4.18-5].
4. Start Application Message delivery and Keep Alive monitoring.

Informative comment

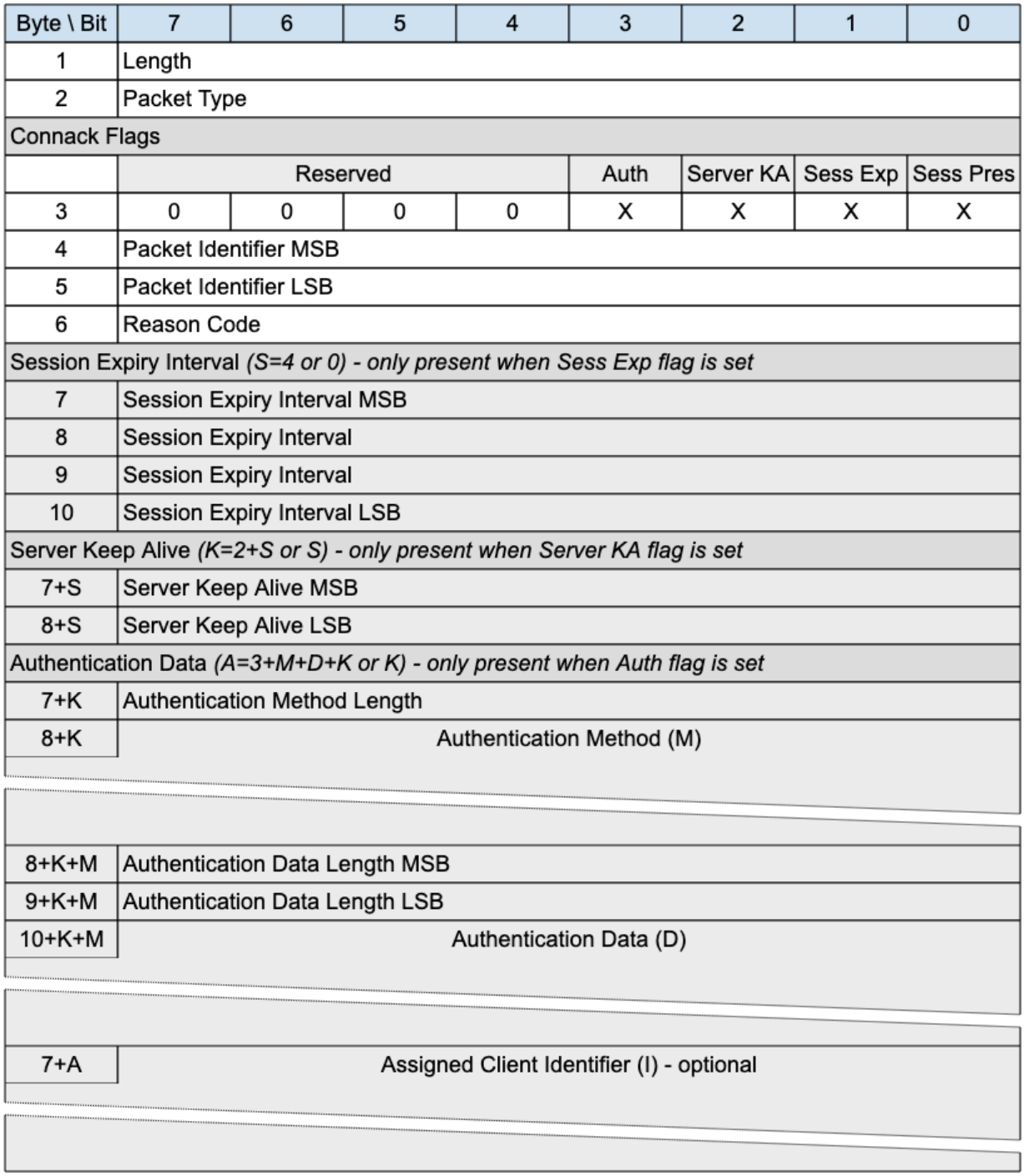
It is recommended that authentication and authorization checks be performed if the Server is being used to process any form of business critical data. If these checks succeed, the Server responds by sending CONNACK with a 0x00 (Success) Reason Code. If they fail, it is suggested that the Server does not send a CONNACK at all, as this could alert a potential attacker to the presence of the MQTT-SN Server and encourage such an attacker to launch a denial of service or password-guessing attack.

A Client MUST wait for a CONNACK packet with a 0x00 (Success) Reason Code before sending any packet that needs a Virtual Connection [MQTT-SN-3.4.18-6].

The Server MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets [MQTT-SN-3.4.18-7].

3.2 CONNACK - Connect Acknowledgement

Figure 3-6 – CONNACK Packet



The CONNACK packet is sent by the Server in response to a CONNECT request from a client.

3.2.1 CONNACK Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.2.2 CONNACK Flags

The CONNACK Flags is a 1 byte field which contains flags specifying the behavior of the MQTT-SN Virtual Connection on the Server. Bits 7-2 of the CONNACK Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the CONNACK packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.2.2.1 Session Present

Position: bit 0 of the CONNACK Flags. Labelled *Sess Pres* in Figure 3-6.

Specifies whether an existing session was present on the Server for the given client identifier. A value of 1 indicates a session was present, a value 0 indicates no session was present.

If the Server accepts a CONNECT with Clean Start set to 1, the Server MUST set Session Present to 0 in the CONNACK Packet in addition to setting a 0x00 (Success) Reason Code in the CONNACK packet.

If the Server accepts a CONNECT with Clean Start set to 0 and the Server has Session State for the client identifier it MUST set Session Present to 1 in the CONNACK packet, otherwise it MUST set Session Present to 0 in the CONNACK packet. In both cases it MUST set a 0x00 (Success) Reason Code in the CONNACK packet.

If the value of Session Present received by the Client from the Server is not as expected, the Client proceeds as follows:

If the Client does not have Session State and receives Session Present set to 1 it MUST delete the Virtual Connection. If it wishes to restart with a new Session the Client can reconnect using Clean Start set to 1.

If the Client does have Session State and receives Session Present set to 0 it MUST discard its Session State if it continues with the Virtual Connection.

If a Server sends a CONNACK packet containing a non-zero Reason Code it MUST set Session Present to 0.

3.2.2.2 Session Expiry Interval Flag (Sess Exp)

Position: bit 1 of the CONNACK Flags. Labelled *Sess Exp* in Figure 3-6.

If the Session Expiry Interval Flag is set to 0, a Session Expiry Interval MUST NOT be present in the Packet [MQTT-SN-3.5.2.2-1].

If the Session Expiry Interval Flag is set to 1, a Session Expiry Interval MUST be present in the Packet [MQTT-SN-3.5.2.2-2].

3.2.2.3 Server Keep Alive Flag (Server KA)

Position: bit 2 of the CONNACK Flags. Labelled *Server KA* in Figure 3-6.

Indicates whether the packet includes a Server Keep Alive or not.

If the Server Keep Alive Flag is set to 0, a Server Keep Alive field MUST NOT be present in the Packet [MQTT-SN-3.5.2.3-1].

If the Server Keep Alive Flag is set to 1, a Server Keep Alive field MUST be present in the Packet [MQTT-SN-3.5.2.3-2].

3.2.2.4 Authentication Flag (Auth)

Position: bit 3 of the CONNACK Flags. Labelled *Auth* in Figure 3-6.

Specifies whether the packet contains authentication material to be considered.

If the Authentication Flag is set to 0, Authentication Method and Data MUST NOT be present in the Packet [MQTT-SN-3.5.2.4-1].
If the Authentication Flag is set to 1, Authentication Method and Data MUST be present in the Packet [MQTT-N-3.5.2.4-2].

3.2.3 Packet Identifier

The same value as the Packet Identifier in the CONNECT or AUTH Packet being acknowledged.

3.2.4 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The Server sending the CONNACK Packet MUST use one of the Reason Codes applicable to CONNACK.

If a Server sends a CONNACK packet containing a Reason code of 0x80 or greater it MUST then delete the Virtual Connection.

3.2.5 Session Expiry Interval

If the Session Expiry Interval is absent the value of Session Expiry Interval in the CONNECT Packet is used. The Server uses this field to inform the Client that it is using a value other than that sent by the Client in the CONNECT.
Refer to 3.4.8 Session Expiry Interval for a description of the use of Session Expiry Interval.

3.2.6 Server Keep Alive

The Server uses this field to inform the Client that it is using a value other than that sent by the Client in the CONNECT.
If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT [MQTT-SN-3.5.6-1].
If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT [MQTT-SN-3.5.6-2].
Refer to 3.4.6 Keep Alive for a description of the use of Keep Alive Interval.

> **Informative comment**
> The primary use of the Server Keep Alive is for the Server to inform the Client that it will disconnect the Client for inactivity sooner than the Keep Alive specified by the Client.

3.2.7 Authentication Method Length

Single byte value (max 0-255 bytes), representing the length of field used to specify the authentication method. Refer to 4.11 Authentication for more information about authentication.

3.2.8 Authentication Method

A UTF-8 Encoded String containing the name of the authentication method. Refer to 4.11 Authentication for more information about authentication.

3.2.9 Authentication Data Length

Two byte value (max 0-65535 bytes), representing the length of field used to specify the authentication data. Refer to 4.11 Authentication for more information about authentication.

3.2.10 Authentication Data

Binary Data containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. Refer to 4.11 Authentication for more information about authentication.

3.2.11 Assigned Client Identifier

The Assigned Client Identifier MUST be a UTF-8 Encoded String [MQTT-SN-3.5.11-1]. This field is optional - its existence or absence is inferred from the Packet length.

The Assigned Client Identifier is the Client Identifier assigned by the Server when the associated CONNECT packet contained no Client Identifier. If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier [MQTT-SN-3.5.11-2].The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server [MQTT-SN-3.5.11-3].

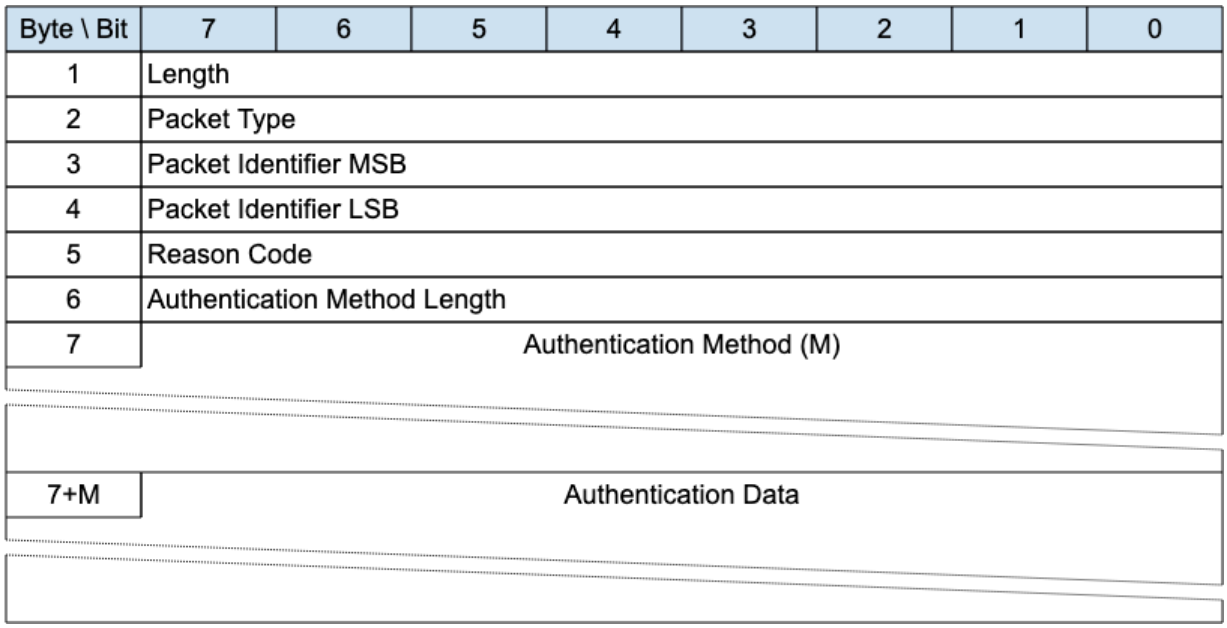
It is suggested that the 36 character Universally Unique IDentifier (UUID) format described in RFC9562 is used for MQTT-SN Assigned Client Identifiers. In any case they should be no longer than 36 characters.

(RFC9562 describes UUIDs that are 128 bits in size, 16 bytes or 32 hexadecimal digits. These UUIDs are commonly expressed in 36 characters, including 4 dashes as separators such as the following: *f81d4fae-7dec-11d0-a765-00a0c91e6bf6*.)

> **Informative comment**
>
> The length of Assigned Client Identifiers should take into account the maximum packet size allowed by the MQTT-SN implementation.
>
> **Informative comment**
>
> Where a Transparent Gateway receives an Assigned Client Identifier from an MQTT Server which is deemed too long for a device, it should map shorter Gateway generated Client Identifiers with their versions returned from the MQTT Server.

3.3 AUTH - Authentication Exchange

Figure 3-7 – AUTH Packet



The authentication method and data is first sent by the Client as part of a CONNECT exchange. If the Server requires additional information to complete the authentication, it responds with an AUTH packet to signal that the Client generates and sends another AUTH packet with the required information and so on until the authentication is complete. The server then responds with a CONNACK message.

3.3.1 AUTH Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.3.2 Packet Identifier

Used to identify the corresponding CONNECT or AUTH packet. It should ideally be populated with a random integer value when sent from Client to Server. When sent from Server to Client, it MUST contain the packet identifier of the CONNECT or AUTH packet being responded to.

3.3.2 Reason Code

The values for the Authentication Reason Code field are shown in Table 9: Reason Code Values. The sender of the AUTH Packet MUST use one of the Reason Codes shown as applicable to the AUTH packet in this table.

3.3.3 Authentication Method Length

The length of the Authentication Method string.

3.3.4 Authentication Method

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

3.3.5 Authentication Data

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

3.3.6 AUTH Actions

Refer to 4.11 Authentication for more information about authentication.

3.4 REGISTER - Register Topic Alias Request

Figure 3-8 – REGISTER Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
REGISTER Flags								
	Reserved							Topic Alias
3	0	0	0	0	0	0	0	X
3	Packet Identifier MSB							
4	Packet Identifier LSB							
Topic Alias (T=2 or 0) - *only present when Topic Alias flag is 1*								
5	Topic Alias MSB							
6	Topic Alias LSB							
5+T	Topic Name							

A REGISTER packet is sent by a Client or Server to create a Session Topic Alias, before sending a PUBLISH with that Session Topic Alias.

The REGISTER packet is sent by a Client to a Server to request a Session Topic Alias for the included Topic Name.

It is sent by a Server to inform a Client about the Session Topic Alias it has assigned to the included Topic Name.

Topic Aliases are always assigned and managed by the Server, not the Client. For more information see 4.7.2 Topic Aliases.

A REGISTER packet may be sent by the Server when the Client is in the Awake state if the Retain Topic Aliases flag on the SLEEPREQ was set to 0, to reinform the Client of a Session Topic Alias.

If the REGISTER packet is sent by a Client, it MUST NOT contain a Topic Alias.

If the REGISTER packet is sent by a Server, it MUST contain a Topic Alias.

3.4.1 REGISTER Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.4.2 REGISTER Flags

The REGISTER Flags is a 1 byte field which contains flags specifying the contents of the REGISTER packet. Bits 7-1 of the REGISTER Flags are reserved and MUST be set to 0.

The receiver MUST validate that the reserved flags in the REGISTER packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.4.2.1 Topic Alias Flag

Position: bit 0 of the REGISTER Flags.
Determines the presence of the Topic Alias field.

If the Topic Alias Flag is set to 0, a Topic Alias MUST NOT be present in the Packet [MQTT-SN-3.7.2.1-1].

If the Topic Alias Flag is set to 1, a Topic Alias MUST be present in the Packet [MQTT-SN-3.7.2.1-2].

3.4.2 Packet Identifier

Used to identify the corresponding REGACK packet. It should ideally be populated with a random integer value.

3.4.3 Topic Alias

Contains the Topic Alias value assigned to the Topic Name included in the Topic Name field.

3.4.4 Topic Name

Fixed Length UTF-8 Encoded String Contains the fully qualified topic name.

3.4.5 REGISTER Actions

As described in 4.7.2 Topic Aliases.

3.5 REGACK - Register Topic Alias Acknowledgement

Figure 3-9 – REGACK Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
REGACK Flags								
	Reserved					Topic Alias	Topic Type	
3	0	0	0	0	0	X	X	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
Topic Alias (T=2 or 0) - *only present when Topic Alias flag is 1*								
6	Topic Alias MSB							
7	Topic Alias LSB							
6+T	Reason Code							

The REGACK packet is sent by a Client or by a Server as an acknowledgment to the receipt and processing of a REGISTER packet.

3.5.1 REGACK Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.5.2 REGACK Flags

The REGACK Flags is a 1 byte field which contains flags specifying the contents of the REGACK packet. Bits 7-3 of the REGACK Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the REGACK packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.5.2.1 Topic Type

Position: bits 0 and 1 of the REGACK Flags.
Determines the format of the topic value. Refer to 2.4 Topic Types for the definition of the various topic types.

The Topic Type in the REGACK packet MUST be Predefined Topic Alias or Session Topic Alias. Any other value is a Protocol Error.

3.5.2.2 Topic Alias Flag

Position: bit 2 of the REGISTER Flags.
Determines the presence of the Topic Alias field.

If the Topic Alias Flag is set to 0, a Topic Alias MUST NOT be present in the Packet [MQTT-SN-3.7.2.1-1].

If the Topic Alias Flag is set to 1, a Topic Alias MUST be present in the Packet [MQTT-SN-3.7.2.1-2].

3.5.3 Packet Identifier

The same value as the Packet Identifier in the REGISTER packet being acknowledged.

3.5.4 Topic Alias

A Topic Alias is an integer value that is used to identify the Topic instead of the Topic Name. This numeric value is used as the Topic Alias.

If the REGACK is sent by a Server in response to a REGISTER request from a Client, the Topic Alias is that which has been assigned by the Server, and which the Client should use during the rest of the Session to refer to the Topic Name identified in the REGISTER packet.

If the REGACK is sent by a Client, it is in response to a REGISTER packet from a Server informing the Client which Topic Alias it should use. When sent by a Client the REGACK MUST NOT contain a Topic Alias.

3.5.5 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the REGACK Packet MUST use one of the Reason Codes applicable to REGACK.

3.6 Publish Requests and Responses

MQTT-SN is designed to be optimized for packet size. For this reason, publish requests have 3 variants:

1. PUBWOS, Publish Without Session, where no session is required
2. PUBLISH Quality of Service 0 where no response is required and thus no packet identifier
3. PUBLISH Quality of Service 1 and 2 where a response is expected.

The table below shows the two packet types.

Figure 3-10 – Publish Packet Types

Packet Name	Type	Description
Publish	0x0C	A PUBLISH packet corresponding to Quality of Service (QoS) 0, 1 or 2
Publish Without Session	0x11	A PUBWOS Packet sent by a Client and does not need not to have an active Session

3.6.1 PUBWOS - Publish Without Session

Figure 3-11 – PUBWOS Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
PUBWOS Flags								
	Reserved			Retain	Reserved		Topic Type	
3	0	0	0	X	0	0	X	X
4	Topic Alias or Topic Name Length MSB							
5	Topic Alias or Topic Name Length LSB							
6	Topic Name *(N) - only present when Topic Type is Topic Name*							
6+N	Payload							

This packet is used by both clients and Servers to publish data for a certain topic.

The PUBWOS packet does not have a corresponding feature in MQTT. If forwarded to an MQTT connection, PUBWOS packets MUST have their MQTT Quality of Service level set to 0. [MQTT-SN-3.10-1]

> **Informative comment**
>
> If the Transport Layer supports multicast, like UDP/IP, the PUBWOS packet is generally sent to a multicast address.

> **Informative comment**
>
> PUBWOS packets received by a Server are not associated with a MQTT-SN Client Session and can be optionally discarded by the Server without being processed for onward delivery.

3.6.1.1 PUBWOS Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.1.2 PUBWOS Flags

The PUBWOS Flags is a 1 byte field which contains flags specifying the content of the packet and the Server behavior on receipt. Bits 7-5 and 3-2 of the PUBWOS FLAGS are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the PUBWOS packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.6.1.2.1 Topic Type

Position: bits 0 and 1 of the PUBWOS Flags.

This determines the format of the topic data field. Refer to 2.4 Topic Types for the definition of the topic types. The Topic Type in the PUBWOS packet MUST be Predefined Topic Alias or Topic Name.

3.6.1.2.2 Retain

Position: bit 4 of the PUBWOS Flags.

This field signifies whether the existing Retained Message for this topic is replaced or kept. For a detailed description of Retained Messages see 4.13 Retained Messages.

3.6.1.3 Topic Alias or Topic Name Length

This field is 2 bytes. It contains the Topic Name length if the Topic Type is Topic Name, or a predefined Topic Alias if the Topic Type is Predefined Topic Alias. Determines the topic which this payload will be published to.

3.6.1.4 Topic Name

If the Topic Type is Topic Name, the Topic Name field MUST be present in the PUBWOS packet.

If the Topic Type is Predefined Topic Alias, the Topic Name field MUST NOT be present in the PUBWOS packet.

If the Topic Type is Topic Name this field will be a UTF-8 encoded string value of length determined by the Topic Name Length field.

3.6.1.5 Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. It is valid for a PUBWOS packet to contain a zero length Payload.

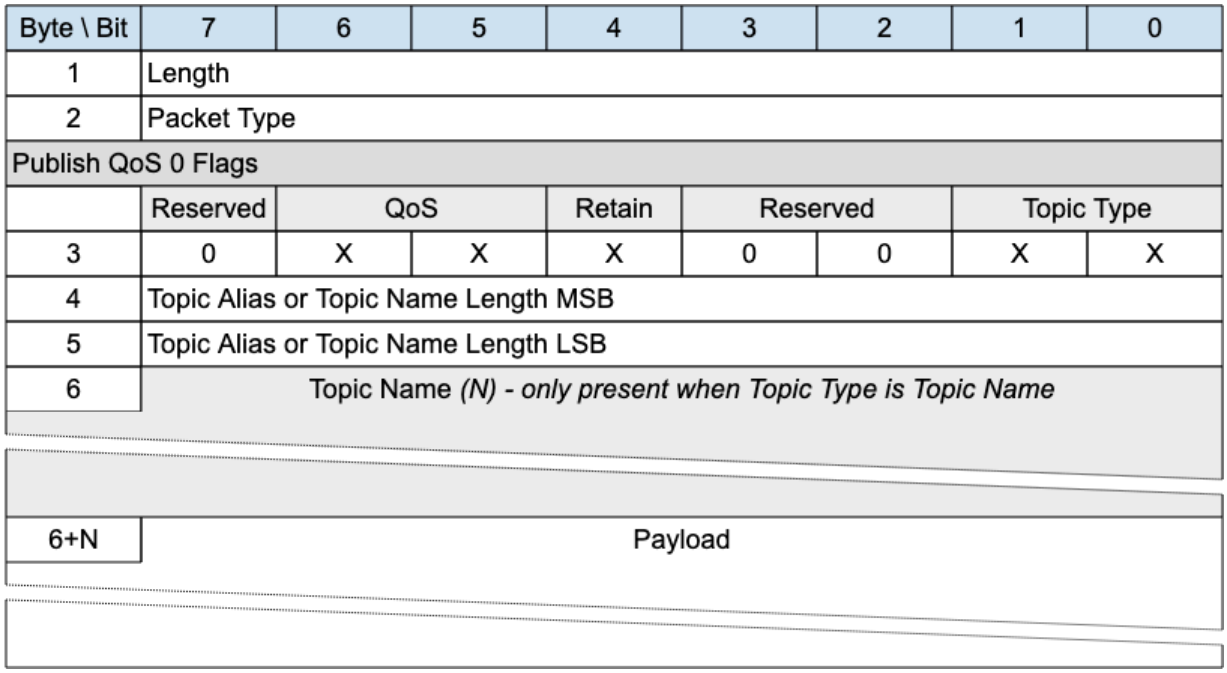
3.6.1.6 PUBWOS Actions

The Client or Server uses a PUBWOS packet to send an Application Message to a Network Address, for possible receipt by a Server or another Client.

If received by a Client or Server, the PUBWOS packet MUST be treated as if its QoS were 0 [MQTT-SN-3.10.5-1] as described in 3.6.3.7 PUBLISH Actions.

3.6.2 PUBLISH with QoS 0

Figure 3-11 – PUBLISH Packet for QoS 0



A PUBLISH packet is sent from a Client to a Server or from a Server to a Client to transport an Application Message.

PUBLISH packets with QoS equal to 0 received by a Client or Server MUST be associated with a Session [MQTT-SN-3.9.3-1].

3.6.2.1 PUBLISH Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.2.2 PUBLISH Flags

The PUBLISH Flags is a 1 byte field which contains flags specifying the content of the packet and the Server behavior. Bits 7 and 3-2 of the PUBLISH Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the PUBLISH packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.6.2.2.1 Topic Type

Position: bits 0 and 1 of the PUBLISH Flags.

This determines the content of the Topic Alias and Topic Name fields. Refer to 2.4 Topic Types for the definition of the various topic types.

The Topic Type may be Topic Name, Predefined Topic Alias or Session Topic Alias.

3.6.2.2.2 QoS

Position: bits 5 and 6 of the PUBLISH Flags.

This field is set to “0b00” for QoS 0. For a detailed description of the various Quality Of Service levels refer to 4.3 Quality of Service levels and protocol flows.

3.6.2.2.3 Retain

Position: bit 4 of the PUBLISH Flags.

This flag signifies whether the message is published as a retained message or not. See 4.13 Retained Messages for more information about Retained Messages.

3.6.2.3 Topic Alias or Topic Name Length

Contains 2 bytes of Topic Name Length if the Topic Type is Topic Name, or the Predefined or Session Topic Alias if the Topic Type is Predefined Topic Alias or Session Topic Alias respectively.

3.6.2.4 Topic Name

If the Topic Type is Topic Namer (0b11) the Topic Name field MUST be present in the PUBLISH packet.

If the Topic Type is Predefined Topic Alias or Session Topic Alias, then the Topic Name field MUST NOT be present in the PUBLISH packet.

Topic Name is a UTF-8 encoded string of length Topic Length.

3.6.2.5 Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. It is valid for a PUBLISH packet to contain a zero length Payload

3.6.2.6 PUBLISH - QoS 0 Actions

As described in 3.6.3.7 PUBLISH Actions.

3.6.3 PUBLISH with QoS 1 and 2

Figure 3-12 – PUBLISH Packet for QoS 1 and 2

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Publish QoS 1 and 2 Flags								
	DUP	QoS		Retain	Reserved		Topic Type	
3	X	X	X	X	0	0	X	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
6	Topic Alias or Topic Name Length MSB							
7	Topic Alias or Topic Name Length LSB							
8	Topic Name *(N) - only present when Topic Type is Topic Name*							
8+N	Payload							

A PUBLISH packet is sent from a Client to a Server or from a Server to a Client to transport an Application Message.

PUBLISH packets with QoS equals to 1 or 2 received by a Client or Server MUST be associated with a Session [MQTT-SN-3.9.4-1].

3.6.3.1 PUBLISH Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.3.2 PUBLISH Flags

The PUBLISH Flags is a 1 byte field which contains flags specifying the content of the packet and the Server behavior. Bits 7 and 3-2 of the PUBLISH Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the PUBLISH packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.6.3.2.1 Topic Type

Position: bits 0 and 1 of the PUBLISH Flags.

This determines the format of the Topic Data field. Refer to 2.4 Topic Types for the definition of Topic Types.

The Topic Type may be Topic Name, Predefined Topic Alias or Session Topic Alias.

3.6.3.2.2 QoS

Position: bits 5 and 6 of the PUBLISH Flags.

Quality of Service - as in MQTT. The QoS levels are:

Figure 3-13 – QoS Definitions

QoS value	Bit 6	bit 5	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

For a detailed description of the various Quality Of Service levels refer to 4.3 Quality of Service levels and protocol flows.

3.6.3.2.3 DUP

Position: bit 7 of the PUBLISH Flags.

The DUP flag indicates the duplicate delivery of QoS 2 PUBLISH packets. If the DUP flag is set to 0, it signifies that the packet is sent for the first time. If the DUP flag is set to 1, it signifies that the packet is retransmitted.

3.6.3.2.4 Retain

Position: bit 4 of the PUBLISH Flags.

This flag signifies whether the message is published as a retained message or not. See 4.13 Retained Messages for more information about Retained Messages.

3.6.3.3 Packet Identifier

Used to identify the corresponding PUBACK packet in the case of QoS 1. Used to identify the corresponding PUBREC, PUBREL and PUBCOMP packets in the case of QoS 2. It should ideally be populated with a random integer value.

3.6.3.4 Topic Alias or Topic Name Length

Contains 2 bytes of Topic Name Length if the Topic Type is Topic Name, or the Predefined or Session Topic Alias if the Topic Type is Predefined Topic Alias or Session Topic Alias respectively.

3.6.3.5 Topic Name

If the Topic Type is Topic Namer (0b11) the Topic Name field MUST be present in the PUBLISH packet.

If the Topic Type is Predefined Topic Alias or Session Topic Alias, then the Topic Name field MUST NOT be present in the PUBLISH packet.

Topic Name is a UTF-8 encoded string of length Topic Length.

3.6.3.6 Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. It is valid for a PUBLISH packet to contain a zero length Payload.

3.6.3.7 PUBLISH Actions

The receiver of a PUBLISH packet MUST respond with the packet as determined by the QoS in the PUBLISH Packet. [MQTT-SN-3.12.7-1].

Figure 3-14 – Expected PUBLISH packet responses

QoS Level	Expected Response
QoS 0	None
QoS 1	PUBACK packet
QoS 2	PUBREC packet

The Client uses a PUBLISH packet to send an Application Message to the Server, for distribution to Clients with matching subscriptions.

The Server uses a PUBLISH packet to send an Application Message to each Client which has a matching subscription.

When Clients make subscriptions with Topic Filters that include wildcards, it is possible for a Client's subscriptions to overlap so that a published Application Message might match multiple filters. In this case the Server MUST deliver the Application Message to the Client respecting the maximum QoS of all the matching subscriptions [MQTT-SN-3.12.7-2]. In addition, the Server MAY deliver further copies of the Application Message, one for each additional matching subscription and respecting the subscription's QoS in each case.

The action of the recipient when it receives a PUBLISH packet depends on the QoS level as described in 4.3 Quality of Service levels and protocol flows.

Informative Comment

If the Server distributes Application Messages to Clients to different protocols and levels (such as MQTT V3.1.1) which do not support features provided by this specification, some information in the Application Message can be lost, and applications which depend on this information might not work correctly.

No more than one QoS 1 or 2 PUBLISH requests MUST be outstanding for a Sender at any one time. Other packets are included in this constraint - refer to 4.9 Flow Control for more information about Flow Control.

Informative comment

The Sender might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets for Flow Control reasons.

3.6.4 PUBACK – Publish Acknowledgement (QoS 1 delivery)

Figure 3-15 – PUBACK Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Reason Code							

A PUBACK packet is the response to a PUBLISH packet with QoS 1.

3.6.4.1 PUBACK Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.4.2 Packet Identifier

The same value as the Packet Identifier in the PUBLISH Packet being acknowledged.

3.6.4.3 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the PUBACK Packet MUST use one of the Reason Codes applicable to PUBACK.

3.6.4.4 PUBACK Actions

As described in 4.3.3 QoS 1: At least once delivery.

3.6.5 PUBREC - Publish Received (QoS 2 delivery part 1)

Figure 3-16 – PUBREC Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Reason Code							

A PUBREC packet is the response to a PUBLISH packet with QoS 2. It is the second packet of the QoS 2 protocol exchange.

3.6.5.1 PUBREC Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.5.2 Packet Identifier

The same value as the Packet Identifier in the PUBLISH Packet being acknowledged.

3.6.5.3 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the PUBREC Packet MUST use one of the Reason Codes applicable to PUBREC.

3.6.5.4 PUBREC Actions

As described in 4.3.4 QoS 2: Exactly once delivery.

3.6.6 PUBREL - Publish Release (QoS 2 delivery part 2)

Figure 3-17 – PUBREL Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Reason Code							

A PUBREL packet is the response to a PUBREC packet. It is the third packet of the QoS 2 protocol exchange.

3.6.6.1 PUBREL Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.6.2 Packet Identifier

The same value as the Packet Identifier in the PUBLISH Packet being acknowledged.

3.6.6.3 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the PUBREL Packet MUST use one of the Reason Codes applicable to PUBREL.

3.6.6.4 PUBREL Actions

As described in 4.3.4 QoS 2: Exactly once delivery.

3.6.7 PUBCOMP - Publish Complete (QoS 2 delivery part 3)

Figure 3-18 – PUBCOMP Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Reason Code							

The PUBCOMP packet is the response to a PUBREL packet. It is the fourth and final packet of the QoS 2 protocol exchange.

3.6.7.1 PUBCOMP Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.6.7.2 Packet Identifier

The same value as the Packet Identifier in the PUBLISH Packet being acknowledged.

3.6.7.3 Reason Code

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the PUBCOMP Packet MUST use one of the Reason Codes applicable to PUBCOMP.

3.6.7.4 PUBCOMP Actions

As described in 4.3.4 QoS 2: Exactly once delivery.

3.7 SUBSCRIBE - Subscribe Request

Figure 3-19 – SUBSCRIBE Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Subscribe Flags								
	No Local	QoS		RaP	Retain Handling		Topic Type	
3	X	X	X	X	X	X	X	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
Topic Alias - *only present when Topic Type is Topic Alias*								
6	Topic Alias MSB							
7	Topic Alias LSB							
6	Topic Filter - *only present when Topic Type is Topic Name*							

The SUBSCRIBE packet is sent from the Client to the Server to create one or more Subscriptions. A Subscription registers a Client's interest in one or more Topics. The Server sends PUBLISH packets to the Client to forward Application Messages that were published to Topics that match the Subscription. The SUBSCRIBE packet also specifies the maximum QoS with which the Server can send Application Messages to the Client.

3.7.1 SUBSCRIBE Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.7.2 SUBSCRIBE Flags

The SUBSCRIBE Flags field is 1 byte and governs the behavior of subscriptions.

3.7.2.1 Topic Type

Position: bits 0 and 1 of the SUBSCRIBE Flags.

This field determines the content of the Topic Alias and Topic Filter fields. Refer to 2.4 Topic Types for the definition of the various topic types.

The Topic Type may be Topic Filter, Predefined Topic Alias or Session Topic Alias.

3.7.2.2 Retain handling

Position: bits 2 and 3 of the SUBSCRIBE Flags.

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 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.

3.7.2.3 Retain as Published

Position: bit 4 of the SUBSCRIBE Flags. Labelled *RaP* in Figure 3-19.

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.

3.7.2.4 QoS

Position: bits 5 and 6 of the SUBSCRIBE Flags.

The 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.

3.7.2.5 No Local

Position: bit 7 of the SUBSCRIBE Flags.

if the value is 1, Application Messages MUST NOT be forwarded to a Virtual Connection with a Client Identifier equal to the Client Identifier of the publishing Virtual Connection.

> **Informative Comment**
>
> A Session is associated with a Client Identifier. A Virtual Connection is a link between Network Identity and a Session by means of the Client Identifier. So a Virtual Connection can be matched to a Client Identifier.

3.7.3 Packet Identifier

Used to identify the corresponding SUBACK packet. It should ideally be populated with a random integer value.

3.7.4 Topic Alias

If the Topic Type is Predefined Topic Alias or Session Topic Alias, then the Topic Alias field MUST be present in the SUBSCRIBE packet.

If the Topic Type is Topic Filter the Topic Alias field MUST NOT be present in the SUBSCRIBE packet.

Contains Fixed Length UTF-8 Encoded String topic filter or Topic Alias (Predefined or Session) as indicated in the *Topic Type* field in flags. Determines the topic names which this subscription is interested in.

3.7.5 Topic Filter

If the Topic Type is Topic Filter the Topic Filter field MUST be present in the SUBSCRIBE packet.
If the Topic Type is Predefined Topic Alias or Session Topic Alias, then the Topic Filter field MUST NOT be present in the SUBSCRIBE packet.

The Topic Filter is a UTF-8 encoded string, which may contain wildcards. A SUBSCRIBE packet with a zero length Topic Filter is a Protocol Error. Refer to 4.12 Handling errors for information about handling errors.

This existence or absence of this field is inferred from the Packet length.

3.7.6 SUBSCRIBE Actions

When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a SUBACK packet [MQTT-SN-3.17.5-1]. The SUBACK packet MUST have the same Packet Identifier as the SUBSCRIBE packet that it is acknowledging [MQTT-SN-3.17.5-2].

If a Server receives a SUBSCRIBE packet containing a Topic Filter that is identical to a Subscription's Topic Filter for the current Session, then it MUST replace that existing Subscription with a new Subscription [MQTT-SN-3.17.5-3]. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its Subscription Options could be different. If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Application Messages MUST NOT be lost due to replacing the Subscription [MQTT-SN-3.17.5-4].

If a Server receives a Topic Filter that is not identical to any Topic Filter for the current Session, a new Subscription is created. If the Retain Handling option is not 2, all matching retained messages are sent to the Client.

The SUBACK packet sent by the Server to the Client MUST contain a Reason Code [MQTT-SN-3.17.5-5]. This Reason Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed [MQTT-SN-3.17.5-6]. The Server might grant a lower Maximum QoS than the subscriber requested. The QoS of Application Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published Application message and the Maximum QoS granted by the Server [MQTT-SN-3.17.5-7]. The server is permitted to send duplicate copies of a Application message to a subscriber in the case where the original Application message was published with QoS 1 and the maximum QoS granted was QoS 0.

> **Informative comment**
>
> If a subscribing Client has been granted maximum QoS 1 for a particular Topic Filter, then a QoS 0 Application Message matching the filter is delivered to the Client at QoS 0. This means that at most one copy of the Application Message is received by the Client. On the other hand, a QoS 2 Application Message published to the same topic is

downgraded by the Server to QoS 1 for delivery to the Client, so that Client might receive duplicate copies of the Application Message.

Informative comment

If the subscribing Client has been granted maximum QoS 0, then an Application Message originally published as QoS 2 might get lost on the hop to the Client, but the Server should never send a duplicate of that Application Message. A QoS 1 Application Message published to the same topic might either get lost or duplicated on its transmission to that Client.

Informative comment

Subscribing to a Topic Filter at QoS 2 is equivalent to saying "I would like to receive Application Messages matching this filter at the QoS with which they were published". This means a publisher is responsible for determining the maximum QoS an Application Message can be delivered at, but a subscriber is able to require that the Server downgrades the QoS to one more suitable for its usage.

3.8 SUBACK - Subscribe Acknowledgement

Figure 3-20 – SUBACK Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
SUBACK Flags								
	Reserved					Topic Alias	Topic Type	
3	0	0	0	0	0	X	X	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
Topic Alias *(A=2 or 0) - only present when Topic Alias flag is set*								
6	Topic Alias MSB							
7	Topic Alias LSB							
6+A	Reason Code							

The SUBACK packet is sent by a Server to a client as an acknowledgment to the receipt and processing of a SUBSCRIBE packet.

3.8.1 SUBACK Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.8.2 SUBACK Flags

The SUBACK Flags is a 1 byte field which contains flags specifying the contents of the SUBACK packet. Bits 7-3 of the SUBACK Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the SUBACK packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.11.2.1 Topic Type

Position: bits 0 and 1 of the SUBACK Flags.

Determines the format of the topic value. Refer to 2.4 Topic Types for the definition of the various topic types.

The Topic Type in the SUBACK packet MUST be either Predefined Topic Alias or Session Topic Alias.

If there is no Topic Alias returned the Topic Type MUST be Predefined Topic Alias.

3.8.2.1 Topic Alias Flag

Position: bit 2 of the SUBACK Flags.

If the Topic Alias Flag is set to 0, a Topic Alias MUST NOT be present in the Packet [MQTT-SN-3.11.2.1-1].

If the Topic Alias Flag is set to 1, a Topic Alias MUST be present in the Packet [MQTT-SN-3.11.2.1-2].

3.8.3 Packet Identifier

The same value as the Packet Identifier in the SUBSCRIBE Packet being acknowledged.

3.8.4 Topic Alias

If a Topic Alias is returned, it MUST be used instead of the Topic Name by the Server when sending PUBLISH packets to the client.

If no Topic Alias is returned, the Topic Alias Flag MUST be 0. This will be the case when subscribing to a Topic Filter containing wildcards, as Topic Aliases can only be applied to Topic Names.

If a Predefined Topic Alias was subscribed to, a Topic Alias MUST NOT be present in the SUBACK.

3.8.5 Reason Code

The values of Reason Codes are shown in 2.3 Reason Code. The sender of the SUBACK Packet MUST use one of the Reason Codes applicable to SUBACK.

3.9 UNSUBSCRIBE - Unsubscribe Request

Figure 3-21 – UNSUBSCRIBE Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Unsubscribe Flags								
	Reserved						Topic Type	
3	0	0	0	0	0	0	X	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
Topic Alias - *only present when Topic Type is Topic Alias*								
6	Topic Alias MSB							
7	Topic Alias LSB							
6	Topic Filter - *only present when Topic Type is Topic Name*							

An UNSUBSCRIBE packet is sent by the Client to the Server to remove subscriptions to topics.

3.9.1 UNSUBSCRIBE Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.9.2 UNSUBSCRIBE Flags

The UNSUBSCRIBE Flags is a 1 byte field which contains flags specifying the contents of the UNSUBSCRIBE packet. Bits 7-2 of the UNSUBSCRIBE Flags are reserved and MUST be set to 0.

The Client MUST validate that the reserved flags in the UNSUBSCRIBE packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.9.2.1 Topic Type

Position: bits 0 and 1 of the UNSUBSCRIBE Flags.

Determines the existence of the Topic Alias or Topic Filter. Refer to 2.4 Topic Types for the definition of the various topic types.

3.9.3 Packet Identifier

Used to identify the corresponding UNSUBACK packet. It should ideally be populated with a random integer value.

3.9.4 Topic Alias

A Topic Alias MUST be present in the UNSUBSCRIBE packet if the Topic Type is Predefined or Session Topic Alias.

A Topic Alias MUST NOT be present in the UNSUBSCRIBE packet if the Topic Type is Topic Name.

Predefined or Session Topic Alias as indicated by the *Topic Type*. Determines the topic names which this subscription is interested in.

3.9.5 Topic Filter

A Topic Filter MUST be present in the UNSUBSCRIBE packet if the Topic Type is Topic Name.

A Topic Filter MUST NOT be present in the UNSUBSCRIBE packet if the Topic Type is Predefined or Session Topic Alias.

The Topic Filter is an UTF-8 Encoded String. The existence or absence of this field is inferred from the Packet length.

3.9.6 UNSUBSCRIBE Actions

If a Topic Alias is used in an UNSUBSCRIBE request, it MUST be translated to its equivalent Topic Name before any other action takes place.

The Topic Filter (whether it contains wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription MUST be deleted [MQTT-SN-3.19.4-1], otherwise no additional processing occurs.

When a Server receives UNSUBSCRIBE :

- It MUST stop adding any new Application Messages which match the Topic Filters, for delivery to the Client [MQTT-SN-3.19.4-2].
- It MUST complete the delivery of any QoS 1 or QoS 2 Application Messages which match the Topic Filters and it has started to send to the Client [MQTT-SN-3.19.4-3].
- It MAY continue to deliver any existing Application Messages which match the Topic Filters buffered for delivery to the Client.

The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet [MQTT-3.19.4-4]. The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-3.19.4-5].

3.10 UNSUBACK - Unsubscribe Acknowledgement

Figure 3-22 – UNSUBACK Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Reason Code							

An UNSUBACK packet is sent by a Server to acknowledge the receipt and processing of an UNSUBSCRIBE packet.

3.10.1 UNSUBACK Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.10.2 Packet Identifier

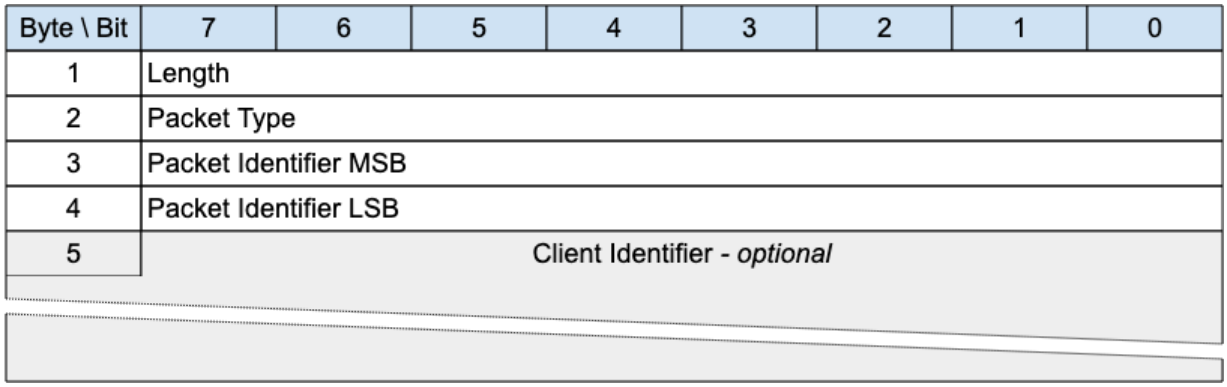
The same value as the Packet Identifier in the UNSUBSCRIBE packet being acknowledged.

3.10.3 Reason Code

The UNSUBACK Reason Codes are shown in Table 9: Reason Code Values. The server sending the UNSUBACK packet MUST use one of the UNSUBACK Reason Codes.

3.11 PINGREQ - Ping Request

Figure 3-23 – PINGREQ Packet



The PINGREQ packet is sent from a Client to the Server. It can be used to:

- Indicate to the Server that the Client is alive in the absence of any other MQTT-SN Control Packets being sent from the Client to the Server. For more information refer to 3.1.6 Keep Alive.
- Request that the Server responds to confirm that it is alive and that it has a Virtual Connection for the Client.
- Exercise the network to determine whether communications are working.

3.11.1 PINGREQ Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.11.2 Packet Identifier

Used to identify the corresponding PINGRESP packet. It should ideally be set to a random integer value.

3.11.3 Client Identifier

An optional field which contains the Client Identifier of the sending Client. This field may be included by a sleeping client when it goes to the Awake state and is waiting for packets to be sent by the Server.

The existence or absence of this field is inferred from the Packet length.

> **Informative Comment**
> The intention of this field is to help the Server link the sender to an existing Session as the maximum time possible for the Client to remain asleep without communicating with the Server is over 128 years. It should not be used in such a way as to allow malicious Clients to take over the session and receive messages not intended for them.

> **Informative Comment**
> If the Server receives a PINGREQ from a Network Address corresponding to a sleeping Client, but the Client Identifier does not match, this would normally be an error, but it is up to the implementation to decide.

3.11.4 PINGREQ Actions

The Server MUST send a PINGRESP packet in response to a PINGREQ packet if it has a Virtual Connection for the sending Client [MQTT-SN-3.11.4-1].

The Server MAY send a DISCONNECT packet in response to a PINGREQ packet if it does not have a Virtual Connection for the sending Client [MQTT-SN-3.11.4-1].

If the Server sends a DISCONNECT packet in response to a PINGREQ packet because it does not have a Virtual Connection for the sending Client, it MUST use Reason Code 244 - No Virtual Connection Exists [MQTT-SN-3.11.4-3].

3.12 PINGRESP - Ping Response

Figure 3-24 – PINGRESP Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							
5	Application Messages Remaining - *optional*							

A PINGRESP Packet is sent by the Server to the Client in response to a PINGREQ packet. It indicates that the Server is alive.

This Packet is used in Keep Alive processing. Refer to 3.1.6 Keep Alive for more details.

A PINGRESP packet is also sent by the Server to inform a Client in the Awake state that it has no more buffered packets for that Client. See 4.14.2 Sleeping Clients for more information about sleeping Clients.

3.12.1 PINGRESP Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.12.2 Packet Identifier

The same value as the Packet Identifier in the PINGREQ Packet being acknowledged.

3.12.3 Application Messages Remaining

The number of Application Messages still queued for delivery at the Server when the PINGRESP is sent to send the Client back to sleep. It is optional, intended as useful information for the Client. This field can be present in the Active state as well as the Awake state.

Values can be:

Figure 3-25 – PINGRESP continuation values

Allowed Values	Description
0	No Application Messages are waiting to be delivered
1 – 254 (incl.)	The number of Application Messages waiting to be delivered
255 (0xFF)	An unspecified positive number of Application Messages waiting to be delivered greater than 0.

The existence or absence of this field is inferred from the Packet length.

3.13 DISCONNECT - Disconnect Notification

Figure 3-26 – DISCONNECT Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Disconnect Flags								
	Reserved					Reason C	Sess Exp	PacketId
3	0	0	0	0	0	X	X	X
Packet Identifier *(I=2 or 0) - only present when PacketId flag is set*								
4	Packet Identifier MSB							
5	Packet Identifier LSB							
Session Expiry Interval *(S=4+I or I) - only present when Sess Exp flag is set*								
4+I	Session Expiry Interval MSB							
5+I	Session Expiry Interval							
6+I	Session Expiry Interval							
7+I	Session Expiry Interval LSB							
Reason Code *(R=1+S or S) - only present when Reason C flag is set*								
4+S	Reason Code							
4+R	Reason String - optional							

The DISCONNECT packet is sent by a Client to indicate that it is going to delete the Virtual connection and go to the Disconnected state.

DISCONNECT may be sent by a Server to indicate that it cannot continue with the Virtual Connection and is deleting it - for instance the Server might be shutting down. It should use an appropriate and allowed Reason Code - 0x8B for Server shutting down, for instance.

A Client may receive an unsolicited DISCONNECT from a Server whether or not it has a Virtual Connection to that Server. This may happen for example when the Server, due to an error, cannot identify the Client to which a received packet belongs.

If a Client or Server receives a packet which requires a Virtual Connection (all packets except CONNECT, ADVERTISE, GWINFO, SEARCHGW and PUBWOS), and no Virtual Connection exists, it MAY send a DISCONNECT in response to the originator with Reason Code 0xF4 - No Virtual Connection exists.

3.13.1 DISCONNECT Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.13.2 DISCONNECT Flags

The DISCONNECT Flags is a 1 byte field which contains flags specifying the contents of the DISCONNECT packet. Bits 7-3 of the DISCONNECT Flags are reserved and MUST be set to 0.

The receiver MUST validate that the reserved flags in the DISCONNECT packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.13.2.1 Packet Identifier Flag

Position: bit 0 of the DISCONNECT Flags. Labelled *PacketId* in Figure 3-27.

If the Packet Identifier Flag is set to 0, a Packet Identifier MUST NOT be present in the Packet [MQTT-SN-3.16.2.2-1].

If the Packet Identifier Flag is set to 1, a Packet Identifier MUST be present in the Packet [MQTT-SN-3.16.2.2-2].

3.13.2.2 Session Expiry Interval Flag

Position: bit 1 of the DISCONNECT Flags. Labelled *Sess Exp* in Figure 3-27.

If the Session Expiry Interval Flag is set to 0, a Session Expiry Interval MUST NOT be present in the Packet [MQTT-SN-3.16.2.3-1].

If the Session Expiry Interval Flag is set to 1, a Session Expiry Interval MUST be present in the Packet [MQTT-SN-3.16.2.3-2].

3.13.2.3 Reason Code Flag

Position: bit 2 of the DISCONNECT Flags. Labelled *Reason C* in Figure 3-27.

If the Reason Code Flag is set to 0, a Reason Code MUST NOT be present in the Packet [MQTT-SN-3.16.2.4-1].

If the Reason Code Flag is set to 1, a Reason Code MUST be present in the Packet [MQTT-SN-3.16.2.4-2].

3.13.3 Packet Identifier

Optional. This can be used by a Server when responding to a Client packet for which there is no current Virtual Connection. In this case, the DISCONNECT packet can be sent by the Server, setting the Reason Code to 0xF4 (No Virtual Connection Exists) and including the Packet Identifier of the erroneous packet, to help with problem diagnosis.

3.13.4 Reason Code

The Reason Code for the DISCONNECT packet is optional. If not provided, 0x00 (Normal disconnection) is assumed.

The values for Reason Codes are shown in 2.3 Reason Code. The sender of the DISCONNECT packet MUST use one of the Reason Code values applicable to DISCONNECT [MQTT-SN-3.23.3-1].

3.13.5 Session Expiry Interval

The Session Expiry Interval is a four-byte integer time interval measured in seconds. If the Session Expiry Interval is absent, the Session Expiry Interval in the CONNECT packet is used.

The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server [MQTT–SN-3.23.4-1].

If the Session Expiry Interval in the CONNECT packet was zero, then it is a Protocol Error to set a non-zero Session Expiry Interval in the DISCONNECT packet sent by the Client. If such a non-zero Session Expiry Interval is received by the Server, it does not treat it as a valid DISCONNECT packet. The Server uses DISCONNECT with Reason Code 0x82 (Protocol Error) as described in 4.12 Handling errors.

3.13.6 Reason String

Fixed Length UTF-8 Encoded String representing a clear text description of the reason for the disconnection.

This field is optional - its existence or absence is inferred from the Packet length.

3.13.7 DISCONNECT Actions

After sending a DISCONNECT packet the sender:

- MUST NOT send any more MQTT-SN Control Packets on that Virtual Connection [MQTT-SN-3.23.6-1].
- MUST delete the Virtual Connection [MQTT-SN-3.23.6-2].

On receipt of DISCONNECT with a Reason Code of 0x00 (Success) the Server:

- MUST discard any Will Message associated with the current Connection without publishing it [MQTT-SN-3.23.6-3], as described in 3.1.3 Will Flags.

On receipt of DISCONNECT, the receiver:

- MUST NOT send any more MQTT-SN Control Packets on the Virtual Connection, if one exists.
- SHOULD delete any existing Virtual Connection.

After receiving a DISCONNECT, a Client can make a new Virtual Connection by sending a CONNECT Packet to the Server.

3.14 WAKEUP - Wake up request

Figure 3-27 – WAKEUP Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							

The wakeup packet is a signal sent from the Server to a client. It is an indication from the Server that the client should wake up. The client is not obliged to honor this request, nor may it even receive the packet. It can choose to ignore the request, or undertake one of the sequences outlined in 4.14.2 Sleeping Clients. The client need not respond to this packet.

3.14.1 WAKEUP Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.14.2 WAKEUP Actions

The Client MAY choose to follow the AWAKE procedure in response to receiving a WAKEUP packet [MQTT-SN-3.21.4-2].

3.15 SLEEPREQ - Sleep request

Figure 3-28 – SLEEPREQ Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
SLEEPREQ Flags								
	Reserved							Retain T
3	0	0	0	0	0	0	0	X
4	Packet Identifier MSB							
5	Packet Identifier LSB							
6	Sleep Duration MSB							
7	Sleep Duration							
8	Sleep Duration							
9	Sleep Duration LSB							

The SLEEPREQ packet is sent from the Client to the Server to indicate that it is going to sleep (that is, transitioning to the Asleep state).

3.15.1 SLEEPREQ Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.15.2 SLEEPREQ Flags

The SLEEPREQ Flags is a 1 byte field which contains flags specifying the contents of the SLEEPREQ packet. Bits 7-1 of the SLEEPREQ Flags are reserved and MUST be set to 0.

The receiver MUST validate that the reserved flags in the SLEEPREQ packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

3.15.2.1 Retain Topic Aliases

Position: bit 0 of the SLEEPREQ Flags. Labelled *Retain T* in Figure 3-28.

Specifies whether Session Topic Aliases should be retained by the Server during the Asleep state. “0” indicates Topic Aliases should be removed during the sleeping period and

renegotiated when Awake or Active. “1” indicates Topic Aliases should be retained during the Asleep period, and therefore not negotiated when Awake or Active.

Predefined Topic aliases MUST NOT be removed by the setting of the Retain Topic Aliases flag to 1.

3.15.3 Packet Identifier

Used to identify the corresponding SLEEPRESP packet. It should ideally be set to a random integer value.

3.15.4 Sleep Duration

The Sleep Duration is a four-byte integer time interval measured in seconds. It is the maximum amount of time that a client may stay asleep without being disconnected by the Server. For more information on sleeping clients, and the purpose of Sleep Duration, see 4.14.2 Sleeping Clients.

The Sleep Duration MUST be greater than 0.

3.15.5 SLEEPREQ Actions

The Server MUST send a SLEEPRESP packet in response to a SLEEPREQ packet [MQTT-SN-3.24.4-1].

After sending a SLEEPREQ packet the Client:

- MAY wait for a SLEEPRESP packet in response from the Server.

A Client will wait for a response if it wishes to ascertain that the Server has received and processed its sleep request. By doing so it will avoid having to:

1. reestablish a Virtual Connection on wakening if the Server did not receive the SLEEPREQ, or
2. retransmit the SLEEPREQ if it does not receive a SLEEPRESP from the server.

It is important for the Client to verify that the Server recognizes that the Client is going to sleep by waiting for a SLEEPRESP response.

A Client will not wait, or stop waiting, if it is concerned that it will use excess power to determine that the Server has received the SLEEPREQ.

3.16 SLEEPRESP - Sleep response

Figure 3-29 – SLEEPRESP Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Packet Identifier MSB							
4	Packet Identifier LSB							

3.16.1 SLEEPRESP Header

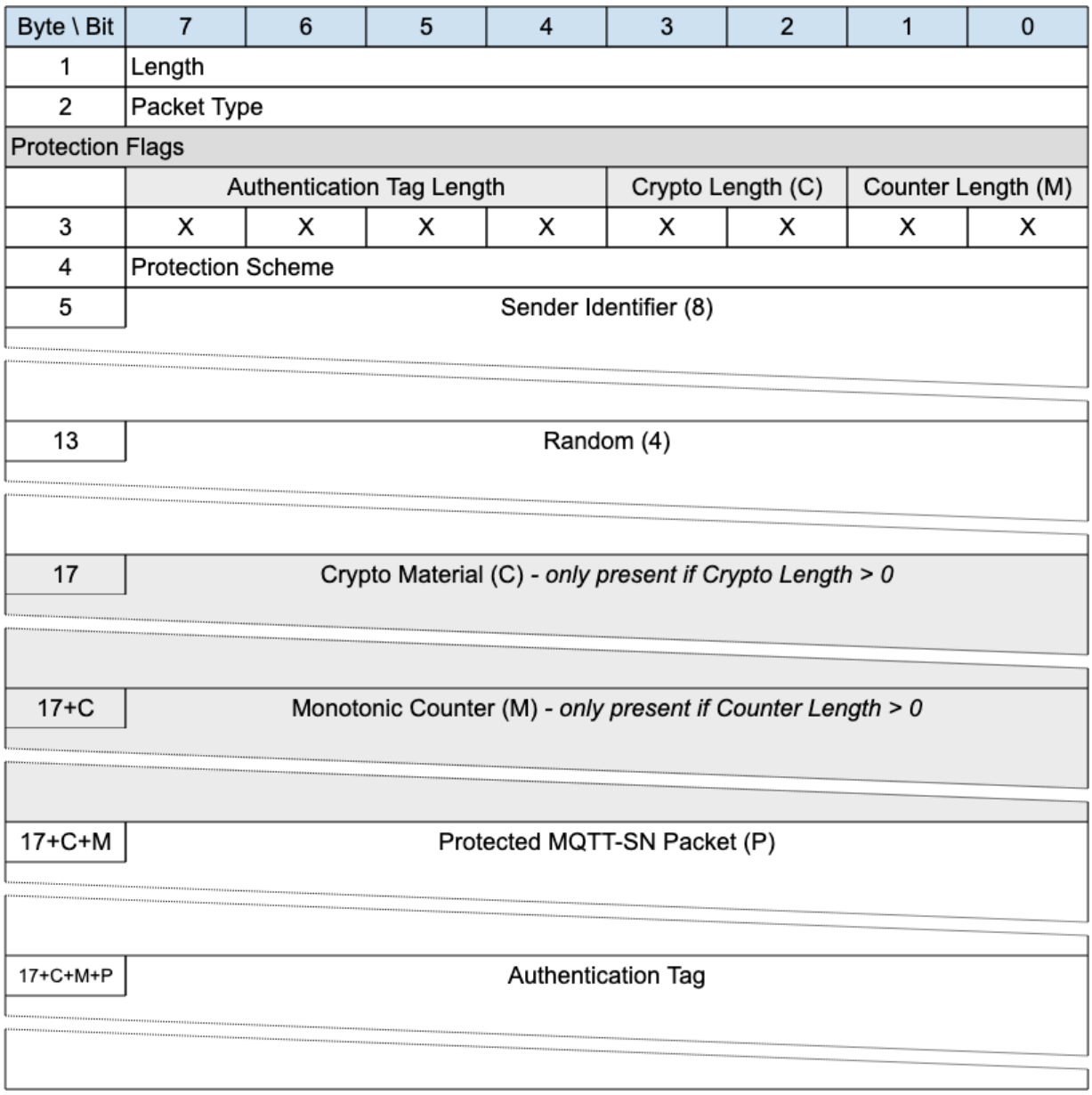
The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.16.2 Packet Identifier

The same value as the Packet Identifier in the SLEEPREQ Packet being acknowledged.

3.17 Protection Encapsulation

Figure 3-31 – Format of an Protection Encapsulated MQTT-SN Packet



Protection encapsulation provides a secure envelope for any MQTT-SN packet (with the exception of the Forward Encapsulation packet). The fields provided by the Protection Encapsulation provide a means by which the sender is identified and the packet is protected, using a number of prescribed protection schemes.

The Sender identified by Sender Identifier is the originator of the protected MQTT-SN Packet and responsible for its protection. This responsibility MUST NOT be delegated to a third party like a Forwarder.

The Sender Identification is required as the Sender and the Receiver of the protected packet must have access to the same shared key to be used directly or after derivation. The Sender

Identifier may not be related to the Network Address of the Sender. The authentication of the Sender and the Receiver, their authorizations and the provisioning of the shared keys used to protect integrity and optionally confidentiality of the protected packet content are out of scope.

A protected packet, like any other one, can be the payload of a Forwarder Encapsulated packet.

When the PROTECTION packet is handled by a Server, it MUST be used to protect all MQTT-SN packets exchanged with a Client for which a shared key (indexed by its Sender Identifier) is available.

> **Informative Comment**
>
> If the Client is not enrolled to the Server (so the Server has no access to a key shared with it on the basis of its Sender Identifier) and the Client and Server are not in a private network, it is recommended that the Server process only MQTT-SN packets received over a DTLS session initiated with mutual authentication by the Client.

When the PROTECTION packet is handled by a Client, it MUST be used to protect all MQTT-SN packets exchanged with a Server for which a shared key (indexed by its Server Identifier) is available.

> **Informative Comment**
>
> If the Server is not enrolled to the Client (so the Client has no access to a key shared with it on the basis of its Server Identifier) and the Client and Server are not in a private network, it is recommended for the Client to open a DTLS session and process only MQTT-SN packets received over it.

3.17.1 Protection Encapsulation Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.17.2 Protection Flags

The Protection Flags is a 1 byte field specifying the properties of the Protection Encapsulation.

3.17.2.1 Monotonic Counter Length

Position: bits 0 and 1 of the Protection Flags. Labelled *Counter Length* in Figure 3-35.

Specifies the number of bytes forming the monotonic counter in big-endian order. Only 3 of the 4 possible values are allowed:

- the value 0x3 is Reserved;
- if 0x2, a monotonic counter field of 32 bits (4 bytes) is present;
- if 0x1, a monotonic counter field of 16 bits (2 bytes) is present;
- if 0x0, the monotonic counter field is not present.

3.17.2.2 Cryptographic Material Length

Position: bits 2 and 3 of the Protection Flags. Labelled *Crypto Length* in Figure 3-35.

Specifies the number of 16 bit groups forming the cryptographic material in big-endian order. Below the meaning of each possible value:

- if 0x3, a cryptographic material field of 96 bits (12 bytes) is present

- if 0x2, a cryptographic material field of 32 bits (4 bytes) is present
- if 0x1, a cryptographic material field of 16 bits (2 bytes) is present
- if 0x0, the cryptographic material field is not present.

3.17.2.3 Authentication Tag Length

Position: bits 4 through 7 of the Protection Flags.
Defines the size of the authentication tag.

- Only 14 of the 16 possible values are allowed:
  - if 0x00, the authentication tag length is provider defined;
  - if 0x01, the authentication tag length is defined by the protection scheme nominal tag length and this is the only value allowed if the protection scheme is not “Authentication Only”;
  - the values from 0x2 to 0x3 are Reserved;
  - any other value 0xZ, so between 0x4 and 0xF, is allowed only for the truncation of “Authentication Only” protection schemes and the authentication tag length will be (0xZ)*16 bits; for example
    - if the value is 0xF, the Authentication tag length will be (0xF)*16=240 bits;
    - if the value is 0x4, the Authentication tag length will be (0x4)*16=64 bits.
- If a truncation of the output of the authentication algorithm is required, it has to be taken in most significant bits first order (leftmost bits).
- Some 0xZ values are not allowed for some “Authentication Only” protection schemes because they define a tag size bigger than the nominal tag size. For example, values from 0x09 (144 bits) to 0x0F (240 bits) are not allowed for “Authentication Only” protection schemes with a nominal tag size less than 144 bits, such as CMAC-128, CMAC-192, CMAC-256.

3.17.3 Protection Scheme

A (1 byte) field which should contain one of the not Reserved indexes in the following table. In general two types of protection scheme are considered: **Authentication only** (like HMAC or CMAC) and **AEAD** (Authenticated Encryption with Associated Data, like GCM, CCM or ChaCha20/Poly1305).

Figure 3-32 – Protection Schemes

Index	Name	Authentication Only	Key Size	Nominal Tag Size
0x00	HMAC-SHA256 (Note 1)	Yes	Any size (Note 2)	256 bits
0x01	HMAC-SHA3_256 (Note 1)	Yes	Any size (Note 2)	256 bits
0x02	CMAC-128 (Note 3)	Yes	128 bits	128 bits
0x03	CMAC-192 (Note 3)	Yes	192 bits	128 bits
0x04	CMAC-256 (Note 3)	Yes	256 bits	128 bits

0x05-0x3B	RESERVED			
0x3C-0x3F	Provider defined	Yes	Provider defined	Provider defined
0x40	AES-CCM-64-128 (Notes 4,5)	No	128 bits	64 bits
0x41	AES-CCM-64-192 (Notes 4,5)	No	192 bits	64 bits
0x42	AES-CCM-64-256 (Notes 4,5)	No	256 bits	64 bits
0x43	AES-CCM-128-128 (Notes 4,5)	No	128 bits	128 bits
0x44	AES-CCM-128-192 (Notes 4,5)	No	192 bits	128 bits
0x45	AES-CCM-128-256 (Notes 4,5)	No	256 bits	128 bits
0x46	AES-GCM-128-128 (Notes 6,7)	No	128 bits	128 bits
0x47	AES-GCM-128-192 (Notes 6,7)	No	192 bits	128 bits
0x48	AES-GCM-128-256 (Notes 6,7)	No	256 bits	128 bits
0x49	ChaCha20/Poly1305 (Notes 8,9)	No	256 bits	128 bits
0x4A-0xEF	RESERVED			
0xF0-0xFF	Provider defined	No	Provider defined	Provider defined

Note(s):

1. Reference https://www.rfc-editor.org/rfc/rfc2104
2. Reference https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf
3. Reference https://www.rfc-editor.org/rfc/rfc4493 and https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf and https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/AES_CMAC.pdf
4. Reference https://www.rfc-editor.org/rfc/rfc3610 and security considerations on https://www.rfc-editor.org/rfc/rfc8152#section-10.2.1
5. AES CCM requires a 13 bytes nonce as indicated in https://www.rfc-editor.org/rfc/rfc8152#section-10.2 and the nonce is obtained by performing SHA256, truncated to the leftmost 104 bits, of the sequence Byte 1 to Byte R (all packet fields until Protected MQTT-SN Packet)
6. Reference https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf and security considerations on https://www.rfc-editor.org/rfc/rfc8152#section-10.1.1
7. AES GCM requires a 12 bytes IV as indicated in https://www.rfc-editor.org/rfc/rfc8152#section-10.1 and the IV is obtained by performing SHA256, truncated to the leftmost 96 bits, of the sequence Byte 1 to Byte R (all packet fields until Protected MQTT-SN Packet)

8. Reference: https://www.rfc-editor.org/rfc/rfc7539 and security considerations on https://www.rfc-editor.org/rfc/rfc8152#section-10.3.1
9. ChaCha20/Poly1305 requires a 12 bytes nonce as indicated in https://www.rfc-editor.org/rfc/rfc8152#section-10.3 obtained by performing SHA256 truncated to 96 bit of the sequence Byte 1 to Byte R (all packet fields until Protected MQTT-SN Packet)

3.17.4 Sender Identifier

The Sender Id field (8 bytes) should contain:

If the message is originated by the ***Server***:

- The SHA256 of the Server Identifier truncated to the leftmost 64 bits (8 bytes);

If the message is originated by the ***Client***:

- **If a session is available**: the SHA256 of the [Client Identifier] truncated to the leftmost 64 bits (8 bytes);
- **If a session is not available**: a unique value per sender over 8 bytes (like a MAC address, or other identifying characteristics). The methods to guarantee the uniqueness of the Sender Id in this case are out of scope.

Informative comment

8 bytes for the Sender Identifier field is calculated with a cryptographic hash, so the probability of collision is $1/2^{64}=5.42 \times 10^{-20}$.

Informative comment

In order to create a whitelist of authorized senders, the Client should store a map of Gateway Identifiers and SHA256 (Gateway Identifier) truncated to the leftmost 64 bits. The Gateway Identifier can be obtained from pre-configuration, from an ADVERTISE packet or from a GWINFO packet.

Informative comment

In order to create a whitelist of authorized senders, the MQTT-SN Server should store a map of Client Identifier and SHA256(Client Identifier) truncated to the leftmost 64 bits (8 bytes for each registered Client Identifier) for the Clients having an active Session and store a list of authorized Sender Identifiers for the clients not capable to establish sessions.

3.17.5 Random

The 4 byte Random field should contain a random number which is not guessable, generated at the time of Protection Encapsulation packet creation.

Informative comment

In case of CCM, in the worst case scenario where the “Cryptographic Material” and the “Monotonic Counter” optional fields are not present, the recommended nonce on 13 bytes will be calculated as SHA256 truncated to 104 bits of the sequence Byte 1 to Byte 16 (all packet fields until Protected MQTT-SN Packet). So considering the same Sender Id, the same nonce can be generated with a probability of $1/2^{32}=2.33 \times 10^{-10}$. With a shorter Random field of 2 bytes, the same nonce would be calculated with a probability of only $1/2^{16}=1.53 \times 10^{-5}$. As CCM is a derivation of CTR (see https://en.wikipedia.org/wiki/CCM_mode), the nonce should never be reused for the same key so the probability to generate two identical nonces should be kept as low as possible. Same for GCM and ChaCha20/Poly1305, the security depends on choosing a unique IV of 12 bytes for every encryption performed with the same key (https://en.wikipedia.org/wiki/Galois/Counter_Mode).

3.17.6 Cryptographic Material

The optional field Cryptographic Material contains 0, 2, 4 or 12 bytes of cryptographic material that when defined it can be used to derive, from a shared master secret, the same keys on the two endpoints and/or, when filled partially or totally with a random value, to further reduce the probability of IV/nonce reuse for CCM or GCM or ChaCha20/Poly1305. For instance when the Cryptographic material length is set to 0x03, the Cryptographic Material field can be partially filled with a random value of 9 bytes (the remaining 3 bytes can be set to 0 if not used) in order to reach the 13 bytes used only once recommended for the nonce used by CCM or it can be partially filled with a random value of 8 bytes in order to reach the 12 bytes used only once recommended for the IV/nonce used by GCM or ChaCha20/Poly1305 .

3.17.7 Monotonic Counter

The optional field Monotonic Counter contains a 0, 2 or 4 byte number that when defined, is increased by the Client or Server for every packet sent. The counters should be considered independent of session or destination. For example, the Client will keep a counter independently from the Server.

3.17.8 Protected MQTT-SN Packet

The field Protected MQTT-SN Packet contains the MQTT-SN packet that is being secured, encoded as per its packet type.

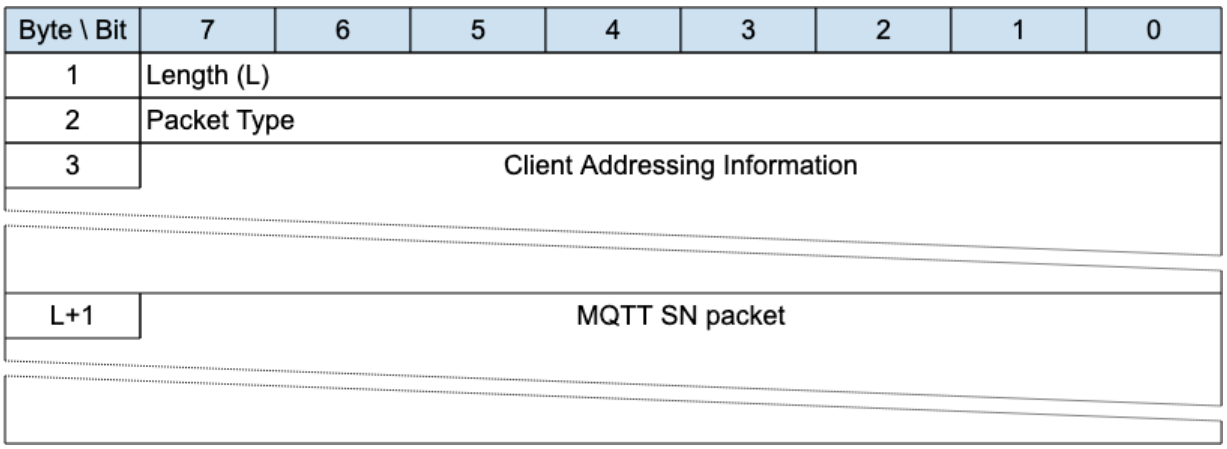
The Protected MQTT-SN Packet MUST NOT be a Forwarder Encapsulated Packet as the shared key used directly or after derivation for the protection must belong to the originator of the content and not to a Forwarder that, in general, is not able to securely identify the originator.

3.17.9 Authentication Tag

The field Authentication Tag field has a length depending on the Authentication Tag Length flag and it is calculated, on the basis of the Protection Scheme selected in Byte 4, on ALL the preceding fields.

3.18 Forwarder Encapsulation

Figure 3-30 – Format of an Forwarder Encapsulated MQTT-SN Packet



An MQTT-SN Client can access a Server through a Forwarder in case the Server is not directly attached to the same Underlying Network as the Client. The Forwarder encapsulates the MQTT-SN Packets it receives from the Client and sends them unchanged to the Server. In the opposite direction, it decapsulates the Packets it receives from the Server and sends them unchanged to the Clients.

The Forwarder Encapsulation contains the addressing information needed by the Forwarder to allow MQTT-SN Packets reach their intended destination(s). Refer to C.1.3 Forwarder for examples.

3.18.1 Forwarder Encapsulation Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

The Length field specifies the number of bytes up to the end of the Wireless Node Identifier field, including the Length field itself.

3.18.2 Client Addressing Information

Identifies the MQTT-SN Client which has sent or should receive the encapsulated MQTT-SN packet. The receiving Gateway can pass this information back to the Forwarder in the MQTT-SN response packet encapsulation, to allow the Forwarder to send packets to the appropriate destination.

The mapping between this information and the address of the sending or receiving Client node is implemented by the Forwarder, if needed. It can contain any other information needed to allow the packets to reach the correct destination.

3.18.3 MQTT-SN Packet

The MQTT-SN packet, encoded according to the packet type.

3.19 Gateway Discovery Packets

The Packets in this section are optional. A description of how this functionality works can be found in C.2 Gateway Advertisement and Discovery.

3.19.1 ADVERTISE - Gateway Advertisement

Figure 3-1 – ADVERTISE Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Gateway Identifier							
4	Duration MSB							
5	Duration LSB							

The ADVERTISE packet is sent periodically by a Gateway to advertise its presence. The time interval until the next transmission is indicated by the *Duration* field.

> **Informative comment**
>
> If the Transport Layer supports multicast, like UDP/IP, the ADVERTISE packet is generally sent using a Multicast Address as the destination.

3.19.1.1 ADVERTISE Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.19.1.2 Gateway Identifier

The *Gateway Identifier* field is 1 byte and uniquely identifies a Gateway which is advertising its presence on the network.

The MQTT-SN protocol itself does not guarantee the uniqueness of the *Gateway Identifier*.

3.19.1.3 Duration

The *Duration* field is a 2-byte integer. It specifies the time interval in seconds until the next ADVERTISE packet is transmitted by this Gateway.

The maximum value that can be encoded is approximately 18 hours.

3.19.2 SEARCHGW - Search for A Gateway

Figure 3-2 – SEARCHGW Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Additional Network Information - *optional*							

The SEARCHGW packet is sent by a Client to find a Gateway to send Application Messages to, and receive Application Messages from.

Informative comment

If the Transport Layer supports multicast, like UDP/IP, the SEARCHGW packet is generally sent using the multicast address as the destination.

To prevent flooding the network, the transmission radius of the SEARCHGW packet may be limited, if the underlying Transport Layer supports the concept.

3.19.2.1 SEARCHGW Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.19.2.2 Additional Network Information

Any extra information that the underlying network needs to control the search process can be included in this variable length field. It will be available to the receiver of this packet and could be used to affect the transmission of the GWINFO response packet.

This field is optional - its existence or absence is inferred from the Packet length.

Informative Comment

In ZigBee mesh networks, this field could contain the ZigBee 1-byte broadcast radius, for instance.

3.19.3 GWINFO - Gateway Information

Figure 3-3 – GWINFO Packet

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
3	Gateway Identifier							
4	Gateway Address - *optional*							

The GWINFO packet is sent as response to a SEARCHGW packet. If sent by a Gateway, it contains only the identifier of the sending Gateway; otherwise, if sent by a client, it also includes the Network Address of the Gateway.

Informative comment

If the Transport Layer supports multicast, like UDP/IP, the GWINFO packet is generally sent using a Multicast Address as destination.

3.19.3.1 GWINFO Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

3.19.3.2 Gateway Identifier

The *Gateway Identifier* field is 1-byte long and uniquely identifies a Gateway in the network.

3.19.3.3 Gateway Address

The *Gateway Address* field has a variable length and contains the Network Address of a Gateway. Its length depends on the type of network over which MQTT-SN operates and is specified by the Length byte.

This field is optional - its existence or absence is inferred from the Packet length. It is only included if the Packet is sent by a Client.

4 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.

4.1 Session state

In order to implement QoS 1 and QoS 2 protocol flows the Client and Server need to associate state with the Client Identifier, this is referred to as the Session State. The Server also stores the subscriptions as part of the Session State.

The Session can continue across a sequence of Virtual Connections. It lasts as long as the latest Virtual Connection plus the Session Expiry Interval.

The Session State in the Client consists of:

- QoS 1 and QoS 2 PUBLISH Packets which have been sent to the Server, but have not been completely acknowledged.
- QoS 2 PUBLISH Packets which have been received from the Server, but have not been completely acknowledged.
- Session Topic Alias mappings.

The Session State in the Server consists of:

- The existence of a Session, even if the rest of the Session State is empty.
- The Client's subscriptions.
- QoS 1 and QoS 2 PUBLISH Packets which have been sent to the Client, but have not been completely acknowledged.
- QoS 1 and QoS 2 PUBLISH Packets pending transmission to the Client and OPTIONALLY QoS 0 PUBLISH Packets pending transmission to the Client.
- QoS 2 PUBLISH Packets which have been received from the Client, but have not been completely acknowledged.
- The Will Message and associated Will data.
- If the Session is currently not connected, the time at which the Session will end and Session State will be discarded.
- Session Topic Alias mappings.

Retained messages do not form part of the Session State in the Server, they are not deleted as a result of a Session ending.

4.1.1 Storing Session State

The Server MUST NOT discard the Session State while the Virtual Connection exists [MQTT-SN-4.1.1-1].

The Client MUST NOT discard the Session State while the Virtual Connection exists [MQTT-SN-4.1.1-2].

The Server MUST discard the Session State when the Virtual Connection is deleted and the Session Expiry Interval has passed [MQTT-SN-4.1.1-3]. A Session Expiry Interval of 0xFFFFFFFF is an infinite amount of time, so never passes.

Informative comment

The storage capabilities of Client and Server implementations will of course have limits in terms of capacity and may be subject to administrative policies. Stored Session State can be discarded as a result of an administrator action, including an automated response to defined conditions. This has the effect of terminating the Session. These actions might be prompted by resource constraints or for other operational reasons. It is possible that hardware or software failures may result in loss or corruption of Session State stored by the Client or Server. It is prudent to evaluate the storage capabilities of the Client and Server to ensure that they are sufficient.

4.1.2 Session Establishment

An MQTT-SN Client needs to create a session on the server, unless it is only publishing using PUBWOS packets. The procedure for setting up a session with a server is illustrated in figures 4-1 and 4-2.

The CONNECT packet contains flags to communicate to the Server that authentication interactions, with the AUTH packet, should take place.

Figure 4-1 – Connect Procedure (without Auth flag set, or no further authentication data required)

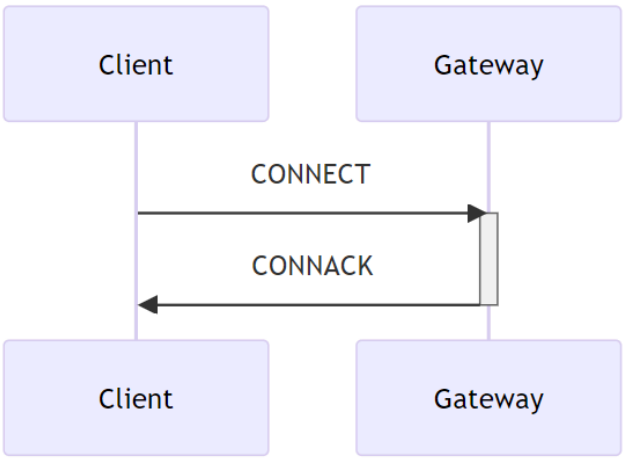
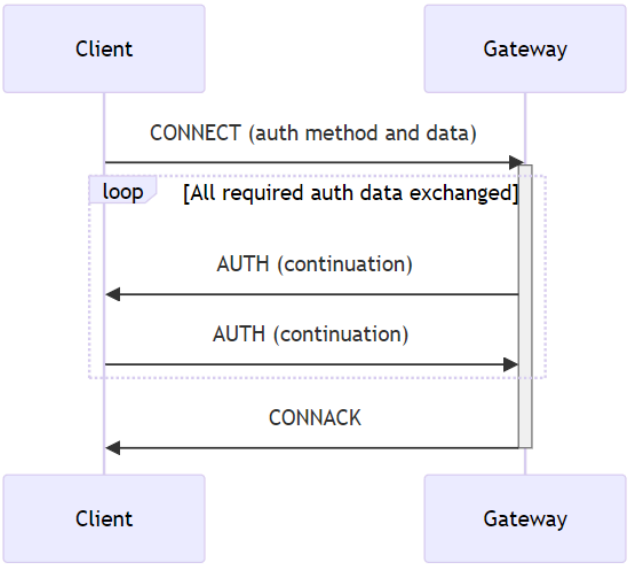


Figure 4-2 – Connect Procedure (with Auth flag set and further authentication data required)



If the Server can not accept the CONNECT request the Server returns a CONNACK packet with the rejection Reason Code.

If the Client provides no client identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier.

An Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server.

4.2 Networks and Virtual Connections

The MQTT-SN protocol requires an Underlying Network to create a Virtual Connection. This carries Packets from a Client to a Server and from a Server to a Client.

The Underlying Network may also be able to send Packets from a Client to more than one Server at once, and from a Server to more than one Client (multicast).

MQTT-SN Packets which are received must be unaltered and complete.

- The Underlying Network does not need to be reliable, it is expected that Packets can be lost or delivered out of order.
- If the Underlying Network might deliver a Packet more than once, for connection-oriented communications (CONNECT, DISCONNECT and other packets in between) the PROTECTION ENCAPSULATION Monotonic Counter MUST be used to eliminate duplicates. (In the case that a protected packet is duplicated, the Monotonic Counter will be the same on all the duplicates of a packet).
- The MQTT-SN protocol will tolerate out of order Packets and it will retransmit lost Packets in the case that an expected acknowledgement has not been received.
- There is no packet error correction in MQTT-SN. If a corrupted or partial packet is received it will cause a protocol error.

- The MQTT-SN implementation may use the origin network address, the DTLS connection identifier, the sender identifier in the PROTECTION ENCAPSULATION, or some other identifier in the Underlying Network to determine the identity of the Virtual Connection.
- The Underlying Network may be connectionless. Virtual Connections do not need to have an Underlying Network event that signals their creation or deletion.
- The Underlying Network may be a radio network.

  Informative comment

  UDP as defined in [RFC0768] can be used for MQTT-SN if the Maximum Transmission Unit is configured to be more than the maximum MQTT-SN Packet size used and no Packet fragmentation occurs. Depending on the network configuration, UDP can duplicate Packets. If this can happen, the PROTECTION ENCAPSULATION monotonic counter should be used.

  Examples of possible consequences of allowing duplicate Packets are:
  – DISCONNECT Packet applied to the wrong Virtual Connection
  – SUBSCRIBE and UNSUBSCRIBE Packets applied to the wrong Virtual Connection
  – PUBLISH QOS=2 published more than once

  The following transport protocols are also suitable but if not capable of multicast the implementation of the optional ADVERTISE, SEARCHGW, GWINFO packets may not be possible:

  - DTLS v1.2 [RFC6347]
  - DTLS v1.3 [RFC9147]
  - QUIC [RFC9000]
  - Non-IP protocols
  - TCP/IP [RFC0793]
  - TLS [RFC5246]
  - WebSocket [RFC6455].

  Informative comment

  Both TCP and UDP ports 1883 and 8883 are registered with IANA for MQTT and secure communication respectively.

A Virtual Connection, which associates a Network Address with a Session, is:

- created with a CONNECT packet
- deleted by any of:
  - a retry timeout
  - DISCONNECT packet
  - protocol error

- required for any MQTT-SN packet to be sent between MQTT-SN clients and servers, except any of the following Packets, including if they are Protection or Forwarder Encapsulated:
    - CONNECT, which creates a Virtual Connection
    - PUBWOS (and PUBLISH QoS -1 if implemented)
    - ADVERTISE, SEARCHGW, GWINFO

4.3 Quality of Service levels and protocol flows

MQTT-SN delivers Application Messages according to the Quality of Service (QoS) levels defined in the following sections. The delivery protocol is symmetric - in the description below the role of the sender can be taken by the Client with the Server being the receiver, or the role of the sender can be taken by the Server with the Client being the receiver. When the Server is delivering an Application Message to more than one Client, each Client is treated independently. The QoS level used to deliver an Application Message outbound to the Client could differ from that of the inbound Application Message.

4.3.1 Publish without session

No Session or Virtual Connection is required to send a message. The message is delivered according to the capabilities of the underlying network. No response is sent by the receiver and no retry is performed by the sender. The message arrives at the receiver either once or not at all.

The sender:

- MUST send a PUBWOS packet.

The receiver:

- MAY decide to accept ownership of the message when it receives a PUBWOS packet.
- MUST treat any accepted messages as QoS 0.

  Informative Comment:

  PUBWOS packets may be Multicast or Unicast.

4.3.2 QoS 0: At most once delivery

The message is delivered according to the capabilities of the underlying network. No response is sent by the receiver and no retry is performed by the sender. The message arrives at the receiver either once or not at all.

In the QoS 0 delivery protocol, the sender

- MUST send a PUBLISH packet with QoS 0.

In the QoS 0 delivery protocol, the receiver

- Accepts ownership of the message when it receives the PUBLISH packet.

Figure 4-3 – QoS 0 protocol flow, informative example

Sender Action	Control Packet	Receiver Action

PUBLISH QoS 0		
	---------->	
		Deliver Application Message to appropriate onward recipient(s)

4.3.3 QoS 1: At least once delivery

This Quality of Service level ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH packet has a Packet Identifier in its Variable Header and is acknowledged by a PUBACK packet.

In the QoS 1 delivery protocol, the sender

- MUST assign an unused Packet Identifier each time it has a new Application Message to publish [MQTT-SN-4.3.3-1].
- MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to 0 [MQTT-SN-4.3.3-2].
- MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver [MQTT-SN-4.3.3-3].

The Packet Identifier becomes available for reuse once the sender has received the PUBACK packet.

The sender is NOT permitted to send further packets with different Packet Identifiers while it is waiting to receive acknowledgements. At all times a Sender MUST have a maximum of one unacknowledged packet.

In the QoS 1 delivery protocol, the receiver

- MUST respond with a PUBACK packet containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message.
- After it has sent a PUBACK packet the receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new Application Message.

Figure 4-4 – QoS 1 protocol flow, informative example

Sender Action	MQTT-SN Control Packet	Receiver action
Store message		
Send PUBLISH QoS 1, DUP=0, <Packet Identifier>	---------->	
		Initiate onward delivery of the Application Message[1]

	<----------	Send PUBACK <Packet Identifier>
Discard message		

[1]The receiver does not need to complete delivery of the Application Message before sending the PUBACK. When its original sender receives the PUBACK packet, ownership of the Application Message is transferred to the receiver.

4.3.4 QoS 2: Exactly once delivery

This is the highest Quality of Service level, for use when neither loss nor duplication of Application Messages are acceptable. There is an increased overhead associated with QoS 2.

In the QoS 2 delivery protocol, the sender:

- MUST assign an unused Packet Identifier when it has a new Application Message to publish
- MUST send a PUBLISH packet containing this Packet Identifier with QoS equal to 2
- MUST set the DUP flag to 0 when it attempts to send a PUBLISH packet for the first time
- MUST set the DUP flag to 1 when it attempts to resend a PUBLISH packet
- MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver
- MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Reason Code value less than 0x80. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet
- MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver
- MUST NOT resend the PUBLISH once it has sent the corresponding PUBREL packet

The Packet Identifier becomes available for reuse once the sender has received the PUBCOMP packet or a PUBREC with a Reason Code of 0x80 or greater.

In the QoS 2 delivery protocol, the receiver:

- MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message
- If it has sent a PUBREC with a Reason Code of 0x80 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message
- Until it has received the corresponding PUBREL packet, the receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case
- MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL

- After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message, irrespective of the setting of its DUP flag

Figure 4-5 – QoS 2 protocol flow, informative example

Sender Action	MQTT-SN Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP=0 <Packet Identifier>		
	---------->	
		Store <Packet Identifier> and message
		PUBREC <Packet Identifier><Reason Code>
	<----------	
Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	---------->	
		Initiate onward delivery of the Application Message[1] then discard the message and <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<----------	
Discard stored state		

[1] The receiver does not need to complete delivery of the Application Message before sending the PUBREC or PUBCOMP. When its original sender receives the PUBREC packet, ownership of the Application Message is transferred to the receiver. However, the receiver needs to perform all checks for conditions which might result in a forwarding failure (for example, quota exceeded or authorization) before accepting ownership. The receiver indicates success or failure using the appropriate Reason Code in the PUBREC.

4.4 Packet delivery retry

There are two situations when packets that require acknowledgement are resent by the sender:

1. when a Virtual Connection is deleted before the acknowledgement is received by the requester, and Clean Start is 0
2. when no acknowledgment is received by the requester within a configured timeout period during the existence of a Virtual Connection

These situations are described in the following sections.

4.4.1 Virtual Connection End

When a Client reconnects with Clean Start set to 0 and a Session is present, both the Client and Server MUST resend any unacknowledged PUBLISH with QoS 1 and 2 packets (not QoS 0) and PUBREL packets using their original Packet Identifiers [MQTT-SN-4.4-1].

If PUBACK or PUBREC is received containing a Reason Code of 0x80 or greater, the corresponding PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted [MQTT-SN-4.4-2].

The DUP flag MUST be set to 1 by the Client or Server when it attempts to resend a PUBLISH QoS 2 packet. [MQTT-SN-4.4-3].

4.4.2 Unacknowledged Packets

In MQTT-SN, any packet may not be delivered by the underlying Network. If a packet is lost, the response to a request will not arrive. In addition to the Keep Alive timer, MQTT-SN Clients and Servers may also resend any packets for which a response is expected, but not received.

The packets that expect a response are:

- CONNECT
- AUTH
- REGISTER
- PUBLISH QoS 1 and 2
- PUBREL
- SUBSCRIBE
- UNSUBSCRIBE
- SLEEPREQ
- PINGREQ

An MQTT-SN Sender may be configured with two parameters to govern its resending of unacknowledged packets:

1. Retry Interval
2. Maximum Retry Count

on the basis of the expected characteristics of the Underlying Network. Example values for these are suggested in C.4 Timer and Counter Values. See also C.5 Exponential Backoff for guidance on varying the *Retry Interval* to reduce potential network congestion.

When no response to one of the above packets is received in the *Retry Interval*, the Sender may resend the packet, at *Retry Interval* intervals, until the *Maximum Retry Count* is reached.

After the *Maximum Retry Count* is reached and a further *Retry Interval* has passed without a response, it is deemed that there is no response.

In the absence of a response to a packet which expects one, the Sender MUST delete the Virtual Connection. A new connection will have to be established to continue.

If a Packet is retransmitted, it MUST have Protection Encapsulation if the previously transmitted Packet had Protection Encapsulation.

If a Packet is retransmitted it MUST be identical to the previously transmitted Packet. The Protection Encapsulation need not be identical.

Informative comment

The value of the *Retry Interval* is not specified by MQTT-SN, however, to be useful it ought to be longer than the network round trip time. If it is excessively long, the time taken to detect and retransmit lost Packets will also be excessively long. Implementers need to take care not to use a retry interval that might cause the network to become congested with retried Packets.

4.5 Application Message receipt

When a Server takes ownership of an incoming Application Message it MUST add it to the Session State for those Clients that have matching Subscriptions [MQTT-SN-4.5-1]. Matching rules are defined in 4.7.1 Topic Names and Topic Filters.

Under normal circumstances Clients receive Application Messages in response to Subscriptions they have created. A Client could also receive Application Messages that do not match any of its explicit Subscriptions. This can happen if the Server automatically assigned a subscription to the Client. A Client could also receive Application Messages while an UNSUBSCRIBE operation is in progress. The Client MUST acknowledge any PUBLISH packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains [MQTT-SN-4.5-2].

4.6 Application Message ordering

An Ordered Topic is a Topic where the Client can be certain that the Application Messages in that Topic from the same Client and at the same QoS are received in the order they were published. When a Server processes a Application Message that has been published to an Ordered Topic, it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client [MQTT-SN-4.6-1].

By default, a Server MUST treat every Topic as an Ordered Topic when it is forwarding Application Messages [MQTT-SN-4.6-2]. A Server MAY provide an administrative or other mechanism to allow one or more Topics to not be treated as an Ordered Topic.

Informative comment

When a stream of messages is published and subscribed to an Ordered Topic with QoS 1, the final copy of each message received by the subscribers will be in the order that they were published. As no more than one message is “in-flight” at any one time, no QoS 1 message will be received after any later one even on re-connection. For example a subscriber might receive them in the order 1,2,3,3,4 but not 1,2,3,2,3,4.

4.7 Topics

4.7.1 Topic Names and Topic Filters

Topic Names are a label for a message, consisting of a series of topic levels, each level separated by the topic level separator.

Topic Filters can match several Topic Names by replacing the topic levels with wildcards.

4.7.1.1 Topic wildcards

The topic level separator is used to introduce structure into the Topic Name. If present, it divides the Topic Name into multiple “topic levels”.

A subscription’s Topic Filter can contain special wildcard characters, which allow a Client to subscribe to multiple topics at once.

A Topic Name, the target of a PUBLISH packet, MUST NOT contain special wildcard characters. [MQTT-SN-4.7.1-1].

4.7.1.1.1 Topic level separator

The forward slash (‘/’ U+002F) is used to separate each level within a topic tree and provide a hierarchical structure to the Topic Names. The use of the topic level separator is significant when either of the two wildcard characters is encountered in Topic Filters specified by subscribing Clients. Topic level separators can appear anywhere in a Topic Filter or Topic Name. Adjacent Topic level separators indicate a zero-length topic level.

4.7.1.1.2 Multi-level wildcard

The number sign (‘#’ U+0023) is a wildcard character that matches any number of levels within a topic. The multi-level wildcard represents the parent and any number of child levels. The multi-level wildcard character MUST be specified either on its own or following a topic level separator. In either case it MUST be the last character specified in the Topic Filter [MQTT-SN-4.7.1.2-1].

Informative comment

For example, if a Client subscribes to “sport/tennis/player1/#”, it would receive Application Messages published using these Topic Names:

- “sport/tennis/player1”
- “sport/tennis/player1/ranking
- “sport/tennis/player1/score/wimbledon”

Informative comment

- “sport/#” also matches the singular “sport”, since # includes the parent level.
- “#” is valid and will receive every Application Message
- “sport/tennis/#” is valid
- “sport/tennis#” is not valid
- “sport/tennis/#/ranking” is not valid

4.7.1.1.3 Single-level wildcard

The plus sign (‘+’ U+002B) is a wildcard character that matches only one topic level.

The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where it is used, it MUST occupy an entire level of the filter [MQTT-SN-4.7.1.3-1]. It can be used at more than one level in the Topic Filter and can be used in conjunction with the multi-level wildcard.

Informative comment

For example, “sport/tennis/+” matches “sport/tennis/player1” and “sport/tennis/player2”, but not “sport/tennis/player1/ranking”. Also, because the single-level wildcard matches only a single level, “sport/+” does not match “sport” but it does match “sport/”.

- “+” is valid
- “+/tennis/#” is valid
- “sport+” is not valid
- “sport/+/player1” is valid
- “/finance” matches “+/+” and “/+”, but not “+”

4.7.1.2 Topics beginning with $

The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character [MQTT-SN-4.7.2-1]. The Server SHOULD prevent Clients from using such Topic Names to exchange messages with other Clients. Server implementations MAY use Topic Names that start with a leading $ character for other purposes.

Informative comment

- $SYS/ has been widely adopted as a prefix to topics that contain Server-specific information or control APIs
- Applications cannot use a topic with a leading $ character for their own purposes

Informative comment

- A subscription to “#” will not receive any messages published to a topic beginning with a $
- A subscription to “+/monitor/Clients” will not receive any messages published to “$SYS/monitor/Clients”
- A subscription to “$SYS/#” will receive messages published to topics beginning with “$SYS/”
- A subscription to “$SYS/monitor/+” will receive messages published to “$SYS/monitor/Clients”
- For a Client to receive messages from topics that begin with $SYS/ and from topics that don’t begin with a $, it has to subscribe to both “#” and “$SYS/#”

4.7.1.3 Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

- All Topic Names and Topic Filters MUST be at least one character long [MQTT-SN-4.7.3-1]
- Topic Names and Topic Filters are case sensitive
- Topic Names and Topic Filters can include the space character
- A leading or trailing ‘/’ creates a distinct Topic Name or Topic Filter
- A Topic Name or Topic Filter consisting only of the ‘/’ character is valid
- Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode] [MQTT-SN-4.7.3-2]
- Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than 65,535 bytes [MQTT-SN-4.7.3-4]. Refer to 1.7.4 UTF-8 Encoded String.

There is no limit to the number of levels in a Topic Name or Topic Filter, other than that imposed by the overall length of a UTF-8 Encoded String.

When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters [MQTT-SN-4.7.3-4]. Each non-wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name character for character for the match to succeed.

Informative comment

The UTF-8 encoding rules mean that the comparison of Topic Filter and Topic Name could be performed either by comparing the encoded UTF-8 bytes, or by comparing decoded Unicode characters.

Informative comment

- “ACCOUNTS” and “Accounts” are two different Topic Names
- “Accounts payable” is a valid Topic Name
- “/finance” is different from “finance”

An Application Message is sent to each Client Subscription whose Topic Filter matches the Topic Name attached to an Application Message. The topic resource MAY be either predefined in the Server by an administrator or it MAY be dynamically created by the Server when it receives the first subscription or an Application Message with that Topic Name. The Server MAY also use a security component to authorize particular actions on the topic resource for a given Client.

4.7.2 Topic Aliases

A Topic Alias is a 2 byte integer value that is used to identify the Topic instead of using the Topic Name. Topic Aliases can reduce the bandwidth needed when Topic Names are long and the same Topic Names are used repetitively.

There are two types of Topic Alias: Predefined and Session. Predefined and Session Topic Aliases MUST occupy separate value spaces. That is, a Session Topic Alias MUST be able to have the same numerical value as a Predefined Topic Alias.

The only reason for the existence of Topic Aliases is to reduce packet size. Therefore, a Topic Alias is transformed to its mapped Topic Name when received by a Client or Server before any further processing.

A Subscription contains a Topic Filter, which is a Topic Name that is allowed to include wildcards - it does not contain any Topic Aliases.

If a Topic Alias exists for a Topic Name, a Sender (Client or Server) MUST use that Topic Alias and not the Topic Name in any PUBLISH packet.

4.7.2.1 Predefined Topic Aliases

Predefined Topic Aliases are known to both sender and receivers before any communication takes place between them. The set of Predefined Topic Aliases in a Server is the same for all Clients that connect to it.

The definitions of Predefined Topic Aliases are not affected by the sending or receiving of any MQTT-SN Packets - their creation and upkeep is an administrative procedure outside the scope of this specification.

Predefined Topic Aliases MUST NOT change for the duration of any MQTT-SN Session.

If a PUBLISH is sent to a Predefined Topic Alias which is not defined on the receiver it is a Protocol Error.

4.7.2.2 Session Topic Aliases

Session Topic Aliases are allocated and controlled by the Server, not the Client.

Session Topic Aliases MUST be allocated on a per Session basis - they are not shared between Sessions either with the same Client or different Clients.

Session Topic Aliases last for the duration of the Session, except after a SLEEPREQ with Retain Topic Aliases equal to 0.

There are several ways that a Session Topic Alias can be created:

- A Client subscribes to a Topic Filter without wildcards. The Session Topic Alias for that Topic Name is returned in the SUBACK packet.
- A Client sends a REGISTER packet to the Server. If the Server successfully creates a Session Topic Alias, its value is returned in the REGACK packet.
- As a result of a wildcard subscription, the Server needs to send a PUBLISH packet to a Client, and no Topic Alias, Predefined or Session, exists for the Topic Name. The Server creates a Session Topic Alias and informs the Client by sending it a REGISTER packet.
- The Server may need to re-register Topic Aliases in the Awake state, as a result of the Client using the Retain Topic Aliases flag set to 0 on the SLEEPREQ packet when going to sleep.

If a Client subscribes to a Topic Filter which does not include wildcard characters, a Predefined or Session Topic Alias MUST be returned in the SUBACK packet.

If a Client subscribes to a Topic Filter which includes wildcard characters, a Topic Alias (Predefined or Session) MUST NOT be returned in the SUBACK packet.

A Session Topic Alias MUST NOT be allowed to map to the same Topic Name as a Predefined Topic Alias.

If a Client requests a Session Topic Alias for a Topic Name which already has a Predefined Topic Alias, the Server MUST return a REGACK with the Topic Type “Predefined Topic Alias”, the Predefined Topic Alias, and the Reason Code “Topic Alias Exists”.

A Session Topic alias and a Predefined Topic Alias with the same numerical value MUST map to different Topic Names.

4.8 Subscriptions

A Subscription is associated only with the Session that created it. Each Subscription includes a Topic Filter, indicating the topic(s) for which messages are to be delivered on that Session, and Subscription Options. The Server is responsible for collecting messages that match the subscription and transmitting them on the Session's Virtual Connection if and when that Virtual Connection exists.

A Session cannot have more than one Subscription with the same Topic Filter, so the Topic Filter can be used as a key to identify the subscription within that Session.

If there are multiple Clients, each with its own Subscription to the same Topic, each Client gets its own copy of the Application Messages that are published on that Topic. This means that the Subscriptions cannot be used to load-balance Application Messages across multiple consuming Clients as in such cases every message is delivered to every subscribing Client.

4.9 Flow Control

The maximum number of unacknowledged MQTT-SN requests in one direction within a Virtual Connection for both Clients and Servers is 1. The packets which need acknowledgement and are included in this constraint are:

- PUBLISH (QoS 1 and 2), PUBREC and PUBREL
- REGISTER
- SUBSCRIBE
- UNSUBSCRIBE
- PINGREQ
- SLEEPREQ
- AUTH

If a Client or Server receives an MQTT-SN request (from the above list) and there is already a request outstanding from the other party within the same Virtual Connection and a different Packet Identifier, then it MUST issue a DISCONNECT with Reason Code 147 (Receive Maximum Exceeded) and delete the Virtual Connection [MQTT-SN-4.9-1].

A Server or Client MUST NOT send a new Packet of a type from the above list, when it has an acknowledgement outstanding for another Packet for which it has not received an acknowledgement [MQTT-SN-4.9-2].

A sender MAY retry a request (send the same Packet) when it is expecting an acknowledgement and none has been received. See 4.4 Packet delivery retry for more information on Packet retries.

Informative comment

The Sender might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets because a request is outstanding.

Informative Comment

It is possible to publish PUBWOS packets in the middle of a QoS 1 or QoS 2 exchange.

Refer to 3.6.3.7 PUBLISH Actions for a description of how Clients and Servers react if they are sent more than one unacknowledged packet.

4.10 Server redirection

A Server can request that the Client uses another Server by sending a CONNACK or DISCONNECT packet with Reason Codes 0x9C (Use another server), or 0x9D (Server moved) as described in 4.12 Handling errors.

The Reason Code 0x9C (Use another server) specifies that the Client SHOULD temporarily switch to using another Server. The other Server is already known to the Client.

The Reason Code 0x9D (Server moved) specifies that the Client SHOULD permanently switch to using another Server. The other Server is already known to the Client.

4.11 Authentication

The MQTT-SN CONNECT and AUTH packets contain Authentication Method and Data fields for use in authentication.

Authentication in MQTT-SN is equivalent to Enhanced Authentication in MQTT 5.0. For an implementation of MQTT 3.1.1 Authentication or MQTT 5.0 Basic Authentication (User Name and Password), refer to 4.11.1.2 MQTT User Name and Password Support.

Alternatively, the Underlying Network may support authentication technology, such as DTLS in the case that the Underlying Network is UDP.

4.11.1 CONNECT and AUTH packets

The authentication information in MQTT-SN CONNECT and AUTH packets allows a range of options from username and password to challenge / response style authentication. It might involve the exchange of AUTH packets between the Client and the Server after the CONNECT and before the CONNACK packets.

To begin authentication, the Client sets the AUTH flag in the CONNECT packet and includes an Authentication Method and optionally Data, depending on the Authentication Method, used in the CONNECT packet. This specifies the authentication method to use and its parameters. 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) as described in 2.3 Reason Code and MUST delete the Virtual Connection [MQTT-SN-4.10-1].

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

> **Informative comment**
>
> The Authentication Method is commonly a SASL mechanism, and 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 the Authentication Data in the CONNECT packet. 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) [MQTT-SN-4.10-2]. If the authentication method requires the Server to send authentication data to the Client, it is sent in the Authentication Data field of the AUTH packet.

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)

[MQTT-SN-4.10-3]. If the authentication method requires the Client to send authentication data for the Server, it is sent in the Authentication Data field of the AUTH packet.

The Client and Server exchange AUTH packets as needed until the Server accepts the authentication by sending a CONNACK with a Reason Code of 0x00. If the acceptance of the authentication requires data to be sent to the Client, it is sent in the Authentication Data field of the CONNACK packet.

The Client can terminate the Virtual Connection at any point in this process by sending a DISCONNECT packet. The Server can reject the authentication at any point in this process. It MUST send a CONNACK with a Reason Code of 0x80 or above as described in 4.12 Handling errors [MQTT-SN-4.10-4].

If the initial CONNECT packet included an Authentication Method then all AUTH packets, and any successful CONNACK packet MUST include an Authentication Method with the same value as in the CONNECT packet [MQTT-SN-4.10-5].

If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet [MQTT-SN-4.10-6]. If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server [MQTT-SN-4.10-7].

If the Client does not include an Authentication Method in the CONNECT packet, the Server SHOULD authenticate using some or all of the information in the CONNECT packet in conjunction with the underlying transport layer or alternatively use the Protection Encapsulation.

Informative example showing a SCRAM challenge

- Client to Server: CONNECT Authentication Method="SCRAM-SHA-1" Authentication Data=client-first-data
- Server to Client: AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication Data=server-first-data
- Client to Server AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication Data=client-final-data
- Server to Client CONNACK rc=0 Authentication Method="SCRAM-SHA-1" Authentication Data=server-final-data

Informative example showing a Kerberos challenge

- Client to Server CONNECT Authentication Method="GS2-KRB5"
- Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5"
- Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=initial context token
- Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=reply context token

- Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5"
- Server to Client CONNACK rc=0 Authentication Method="GS2-KRB5" Authentication Data=outcome of authentication

4.11.1.1 Re-authentication

If the Client supplied an Authentication Method in the CONNECT packet, it can initiate a re-authentication at any time after receiving a CONNACK. It does this by sending an AUTH packet with a Reason Code of 0x19 (Re-authentication). The Client MUST set the Authentication Method to the same value as the Authentication Method originally used to authenticate the Virtual Connection [MQTT-SN-4.10.1-1]. If the authentication method requires Client data first, this AUTH packet contains the first piece of authentication data in the Authentication Data field.

The Server responds to this re-authentication request by sending an AUTH packet to the Client with a Reason Code of 0x00 (Success) to indicate that the re-authentication is complete, or a Reason Code of 0x18 (Continue authentication) to indicate that more authentication data is needed. The Client can respond with additional authentication data by sending an AUTH packet with a Reason Code of 0x18 (Continue authentication). This flow continues as with the original authentication until the re-authentication is complete or the re-authentication fails.

If the re-authentication fails, the Client or Server MUST send DISCONNECT with an appropriate Reason Code as described in 4.12 Handling errors, and MUST delete the Virtual Connection [MQTT-SN-4.10.1-2].

During this re-authentication sequence, the flow of other packets between the Client and Server is paused, pending the new authentication outcome.

> **Informative comment**
>
> The Server might limit the scope of the changes the Client can attempt in a re-authentication by rejecting the re-authentication. For instance, if the Server does not allow the User Name to be changed it can fail any re-authentication attempt which changes the User Name.

4.11.1.2 MQTT User Name and Password Support

To support the equivalent of the MQTT User Name and Password fields in the CONNECT packet, do the following:

- Set the Authentication Method field to MQTT-BASIC.
- Set the Authentication Data field to to:
    1. MQTT User Name: a Two Byte Integer length followed by a UTF-8 Encoded String as defined in 1.7.4 UTF-8 Encoded String.
    2. MQTT Password: a Two Byte Integer length followed by binary data.

The User Name string and Password binary data must have the same length as the values in their corresponding preceding length fields.

This is a one-way transfer of information - the response MUST be a CONNACK, not an AUTH packet.

Informative comment

The length field in front of the password is not strictly necessary as the Authentication Data field length is known, but including it makes the transfer to and from the MQTT Connect packet simpler.

Figure 4-6 – CONNECT with MQTT User Name and Password, informative example

Byte \ Bit	Description	7	6	5	4	3	2	1	0
1	Length (30)	0	0	0	1	1	1	1	0
2	Packet Type (5)	0	0	0	0	0	1	0	1
Connect Flags									
	Reserved (0)								
	Default Awake Messages (0)								
	Auth Flag (1)								
	Will Flag (0)								
3	Clean Start (1)	0	0	0	0	0	1	0	1
4	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
5	Packet Identifier LSB (1)	0	0	0	0	0	0	0	1
6	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
7	Keep Alive LSB (60)	0	0	1	1	1	1	0	0
8	Max Packet Size MSB (0)	0	0	0	0	0	0	0	0
9	Max Packet Size LSB (100)	0	1	1	0	0	1	0	0
Auth Method (MQTT-BASIC)									
10	Auth Method Length (10)	0	0	0	0	1	0	1	0
11	'M'	0	1	0	0	1	1	0	1
12	'Q'	0	1	0	1	0	0	0	1
19	'I'	0	1	0	0	1	0	0	1
20	'C'	0	1	0	0	0	0	1	1
Authentication Data (MQTT User Name, MQTT Password)									
21	Auth Data Length MSB (0)	0	0	0	0	0	0	0	0
22	Auth Data Length LSB (7)	0	0	0	0	0	1	1	1
MQTT User Name (M)									
23	MQTT User Name Length MSB (0)	0	0	0	0	0	0	0	0
24	MQTT User Name Length LSB (1)	0	0	0	0	0	0	0	1
25	'M'	0	1	0	0	1	1	0	1
MQTT Password (PW)									
26	MQTT Password Length MSB (0)	0	0	0	0	0	0	0	0
27	MQTT Password Length LSB (2)	0	0	0	0	0	0	1	0
28	'P'	0	1	0	1	0	0	0	0
29	'W'	0	1	0	1	0	1	1	1
Client Identifier (C)									
30	'C'	0	1	0	0	0	0	1	1

To support the equivalent of the MQTT User Name and Password together with MQTT Enhanced Authentication, in the CONNECT packet do the following:

- Set the Authentication Method field to MQTT-ENHANCED.
- Set the Authentication Data field to:

a. MQTT User Name: a Two Byte Integer length followed by a UTF-8 Encoded String as defined in 1.7.4 UTF-8 Encoded String.
b. MQTT Password: a Two Byte Integer length followed by binary data.
c. MQTT Authentication Method: a Two Byte Integer length followed by a UTF-8 Encoded String as defined in 1.7.4 UTF-8 Encoded String .
d. MQTT Authentication Data: a Two Byte Integer length followed by binary data.

In any subsequent AUTH and CONNACK packets of the authentication exchange:

1. The MQTT User Name and MQTT Password fields MUST not be included.
2. The Authentication Method MUST remain MQTT-ENHANCED throughout.

4.12 Handling errors

4.12.1 Malformed Packet and Protocol Errors

Definitions of Malformed Packet and Protocol Errors are contained in 1.3 Terminology, some but not all of these error cases are noted throughout the specification. The rigor with which a Client or Server checks an MQTT-SN Control Packet it has received will be a compromise between:

- The size of the Client or Server implementation.
- The capabilities that the implementation supports.
- The degree to which the receiver trusts the sender to send correct Control Packets.
- The degree to which the receiver trusts the network to deliver Control Packets correctly.
- The consequences of continuing to process a packet that is incorrect.

If the sender is compliant with this specification it will not send Malformed Packets or cause Protocol Errors. The Reason Codes used for Malformed Packet and Protocol Errors include:

- 0x81 Malformed Packet
- 0x82 Protocol Error
- 0x93 Receive Maximum exceeded
- 0x95 Packet too large

When a Client detects a Malformed Packet or Protocol Error associated with a Virtual Connection it SHOULD send a DISCONNECT packet containing an appropriate Reason Code and MUST delete the associated Virtual Connection. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in 2.3 Reason Code.

When a Server detects a Malformed Packet or Protocol Error for any packet except ADVERTISE, SEARCHGW, GWINFO, PUBWOS and CONNECT, the Server MAY send a DISCONNECT packet with an appropriate Reason Code and MUST delete the associated Virtual Connection if one exists. [MQTT-4.13.1-1] In the case of an error in a CONNECT packet it MAY send a CONNACK packet containing the Reason Code. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in 2.3 Reason Code. There are no consequences for other Sessions.

If either the Server or Client omits to check some feature of a Control Packet, it might fail to detect an error, consequently it might allow data to be damaged.

4.12.2 Other errors

Errors other than Malformed Packet and Protocol Errors cannot be anticipated by the sender because the receiver might have constraints which it has not communicated to the sender. A receiving Client or Server might encounter a transient error, such as a shortage of memory, that prevents successful processing of an individual Control Packet.

Acknowledgment packets PUBACK, PUBREC, PUBREL, PUBCOMP, REGACK, SUBACK, UNSUBACK with a Reason Code of 0x80 or greater indicate that the received packet, identified by a Packet Identifier, was in error. There are no consequences for other Sessions or other Packets flowing on the same Session.

The CONNACK and DISCONNECT packets allow a Reason Code of 0x80 or greater to indicate that the Virtual Connection will be deleted. If a Reason Code of 0x80 or greater is specified, then the Virtual Connection MUST be deleted whether or not the CONNACK or DISCONNECT is sent [MQTT-4.13.2-1]. Sending one of these Reason Codes has no consequences for any other Session.

If the Control Packet contains multiple errors the receiver of the Packet can validate the Packet in any order and take the appropriate action for any of the errors found.

Refer to 5.4.9 Handling of Disallowed Unicode code points for information about handling Disallowed Unicode code points.

4.13 Retained Messages

If the RETAIN flag is set to 1 in a PUBLISH or PUBWOS packet received by a Server, the Server MUST replace any existing Retained Message for this topic and store the Application Message [MQTT-SN-4.26-1], 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 message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message [MQTT-SN-4.26-2]. A Retained Message with a Publish Data containing zero bytes MUST NOT be stored as a Retained Message on the Server [MQTT-SN-4.26-3].

If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the message as a Retained Message and MUST NOT remove or replace any existing Retained Message [MQTT-SN-4.26-4].

When a new Subscription is made, the last retained message, if any, on each matching topic name is sent to the Client as directed by the Retain Handling Subscribe Flag. These messages are sent with the RETAIN flag set to 1. Which retained messages are sent is controlled by the Retain Handling Subscribe Flag. At the time of the Subscription:

- If Retain Handling is set to 0 the Server MUST send the retained messages matching the Topic Filter of the subscription to the Client [MQTT-SN-4.26-5].
- If Retain Handling is set to 1 then if the subscription did not already exist, the Server MUST send all retained messages matching the Topic Filter of the subscription to the Client, and if the subscription did exist the Server MUST NOT send the retained messages. [MQTT-SN-4.26-6].
- If Retain Handling is set to 2, the Server MUST NOT send the retained messages [MQTT-SN-4.26-7].

Refer to 3.10.2 SUBSCRIBE Flags for a definition of the Subscription Flags.

If the Server receives a PUBLISH packet with the RETAIN flag set to 1, and QoS 0 it SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any time. If this happens there will be no retained message for that topic.

The setting of the RETAIN flag in an Application Message forwarded by the Server from an established Virtual Connection is controlled by the Retain As Published subscription option. Refer to 3.10.2 SUBSCRIBE Flags for a definition of the Subscription Flags.

- If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the received PUBLISH packet [MQTT-SN-4.26-8].
- If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN flag equal to the RETAIN flag in the received PUBLISH packet [MQTT-SN-4.26-9].

Informative comment

Retained messages are useful where publishers send state messages on an irregular basis. A new subscriber will receive the most recent state.

Informative comment

As in MQTT 3.1.1 there is no notion of message expiry in MQTT-SN, including expiry of Retained Messages. It is an administrative decision under what conditions to remove Retained Messages, if at all. They should be kept long enough to support the expectations of the applications that will use the Server, at a minimum.

4.14 Client states

At any time, a Client will be in one of the following states from the perspective of the Server:

Figure 4-7 – Client States

State	State Description	Possible Transitions
None	The Client is unknown to the Server. There is no Session State nor Virtual Connection. A Client may transition from here to **Active** with a CONNECT.	**Active**
Disconnected	The Client is considered offline and not able to receive packets until it has re-established a session with the Server by way of a CONNECT. The Server has Session state for this Client, but there is no Virtual Connection. A Client may transition from here to **Active** with a CONNECT, or to **None** on Session expiry.	**Active** **None**
Active	The Client is actively engaged in the Session. A Virtual Connection exists. It should be able to send and receive packets. Its state is supervised by the Server with the associated Keep Alive timer. A Client may transition from here to **Asleep** by way of a SLEEPREQ or **Disconnected** by way of a DISCONNECT, Keep Alive timeout or Retry timeout**.**	**Asleep** **Disconnected**
Asleep	The Client is engaged in an ongoing Session, and a Virtual Connection exists. The Client cannot receive packets; it can send packets. The Server should not expect a response from the client in this state until further packets are received from the client. A Client may transition from here to **Awake** (by way of PINGREQ), **Active** by way of CONNECT, **Disconnected** (by way of DISCONNECT, Sleep timeout or Retry timeout). The Server may send a WAKEUP packet to the Client, as an indication that messages are waiting - it is up to the Client to act, if it is even able to notice it.	**Awake** **Active** **Disconnected**
Awake	The Client is partially engaged in an ongoing session and a Virtual Connection exists. The client transitions back to the **Asleep** state on receipt of a PINGRESP packet or **Disconnected** (by	**Asleep** **Active** **Disconnected**

	way of DISCONNECT, or on Keep Alive or Retry timeout for the possible PUBACK, PUBREL, PUBREC, PUBCOMP or REGACK packets to be received from the Client). The Client may also move to the **Active** state by way of CONNECT.	

A Server **MUST NOT** attempt to send packets to a Disconnected Client**.**

Any packet except CONNECT received from a Disconnected Client MUST NOT be processed. A DISCONNECT with error should be sent in response, unless the packet received is PUBWOS.

In the Asleep state, a Client MUST only send PINGREQ, CONNECT or DISCONNECT packets to the Server.

In the Awake state, a Client MUST not send ANY packets other than those involved in the receipt of PUBLISH packets (PUBACK, PUBREC, PUBCOMP, REGACK) or CONNECT or DISCONNECT.

Whenever a CONNECT is received by a Server, any existing Virtual Connection for that Client MUST be deleted and a new one created with all CONNECT Packet processing, regardless of the state of the Client.

Transition through these states is governed by a sequence of packets between Client and Server and mediated by timers resident on the Server. A Client is in the Active state when the Server receives a CONNECT packet from that Client. This state is supervised by the Server with the Keep Alive timer. If the Server does not receive any packet from the Client in a defined period, the Server will consider that client as Disconnected and delete the Virtual Connection. The Disconnected state is governed by the Session Expiry timer - on expiry the Server is free to remove the Client session. A Client moves into the Asleep state by issuing a SLEEPREQ packet. To be certain that the Server has also recorded the Client as being asleep, the Client needs to wait for a positive SLEEPRESP response. For more information on the Asleep state, refer to 4.14.2 Sleeping Clients.

See C.5 Client State Diagrams for informative state diagrams to help illustrate these transitions.

> **Informative Comment**
>
> In MQTT-SN 1.2 there existed a Lost state, which was identical to the Disconnected state, except that it was reached by a timer expiry on the Server rather than an explicit Disconnect request from the Client. As the Lost state was not really different from Disconnected except for the history of Client events, similar information may be kept by the Server, in for instance its administrative logs.

4.14.1 Session Timers

The following timers are used by Servers, on a per Client basis, to handle Client states, and by Clients to direct their actions. In general, Clients are able to infer the Server's view of their state by observing the Server's response.

Figure 4-10 – Session Timers

Timer Name	State(s)	Timeout State	Defined in	Information

Keep Alive	Active	Disconnected	CONNECT	3.1.6 Keep Alive
Sleep Duration	Asleep	Disconnected	SLEEPREQ	4.14.2 Sleeping Clients
Session Expiry	Disconnected	None	CONNECT, DISCONNECT	4.1.1 Storing Session State
Retry	Active, Awake, Asleep	Disconnected	Sender configuration	4.4 Packet delivery retry

Only one timer is operational at a time on an MQTT-SN component. The Keep Alive timer is suspended while the Packet Retry timer is running.

For example values of these timers, see C.3 Example Timer and Counter Values.

4.14.2 Sleeping Clients

The Asleep state is intended to allow Clients, which may be running on battery powered devices, to save as much energy as possible. These Clients enter a low-power mode when they are not active, and will wake when they have data to send or receive. The Server needs to be aware of the sleeping state of these Clients and buffer messages destined for them, so that they may be delivered when the Clients wake up.

To go to sleep, a Client sends a SLEEPREQ packet containing a Sleep Duration in seconds. The Server acknowledges that packet with a SLEEPRESP packet and considers the Client to be Asleep.

If the Server does not receive an MQTT-SN Control Packet from the Client within one and a half times the Sleep Duration, it MUST delete the Virtual Connection to the Client. The Client will then be considered to be Disconnected.

During the Asleep state, packets that need to be sent to the client are buffered at the Server. The Server MUST buffer Application Messages of QoS 1 and 2.

> **Informative comment**
>
> The Server may *choose* to buffer messages of QoS 0 while the Client is in the Asleep state.

The Client wakes by sending a PINGREQ, which may contain the Client Identifier. If the Server has buffered packets for the Client, it will send them to the Client, acknowledging the Default Awake Messages value sent in the CONNECT packet. If the number of messages buffered on the Server waiting to be sent exceeds the value specified by the client in the Default Awake Messages field, the Server MUST send only the Default Awake Messages value number of messages. It cuts short the AWAKE cycle, and MUST respond with a PINGRESP with a messages-left value of either the number of messages remaining in the Server buffer or 0xFFFF (meaning undetermined number of messages greater than 0 remaining).

During the Awake state, for each Application Message the Server sends to the Client, the application messages' quality of service MUST be honored - a full packet interaction MUST take place including all normative phases of acknowledgement, including any associated

retransmission logic. If, during the delivery of Application Messages from the Server to the Client, the Server detects a Retry timeout, it should consider the Client disconnected, and send a DISCONNECT packet with an error Reason Code.

The transfer of packets to the Client is closed by the Server by means of a PINGRESP packet. That is, the Server will consider the Client as Asleep and restart the Sleep Duration timer after having sent the PINGRESP packet. If the Server does not have any packets buffered for the client, it MUST respond immediately with a PINGRESP packet, returning the Client back to the Asleep state, and restartings the Sleep Duration timer for that Client.

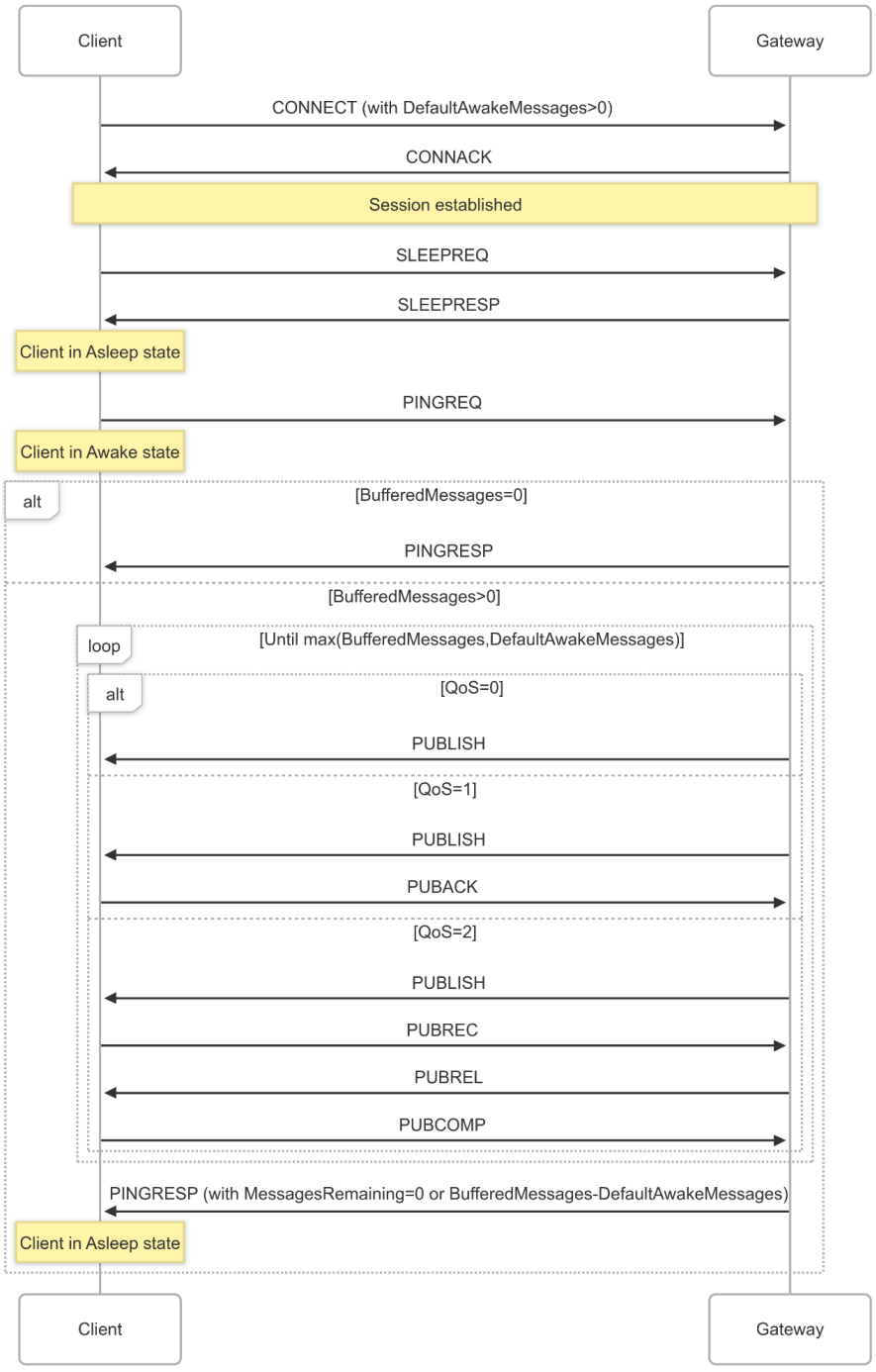
After having sent the PINGREQ to the Server, the client uses the retransmission procedure of 4.4 Packet delivery retry to supervise the arrival of packets sent by the Server. To avoid draining its battery due to excessive retransmission of the PINGREQ packet, the Client should limit the retransmission with a Maximum Retry Count, and go back to sleep when the limit is reached.

From the *Asleep* state, a client can return either to the *Active* state by sending a CONNECT packet or to the *Disconnected* state by sending a DISCONNECT packet. The Client can also modify its sleep configuration by sending a SLEEPREQ Packet with a new value of Sleep Duration.

Note that a sleeping Client should go to the *Awake* state only if it wants to check whether the Server has any Application Messages buffered for it and return as soon as possible to the *Asleep* state without sending any packets to the Server (other than PUBACK, PUBREC, PUBCOMP or REGACK). If it wants to do more than this, it needs to create a new Virtual Connection by sending a CONNECT packet to the Server.

Session Topic Aliases last for the duration of a Session which exists throughout the sleep cycle. However, if the Client wants to save storage by removing the Session Topic Aliases while Asleep, it can set the Retain Topic Aliases flag on the SLEEPREQ packet to 0. The disadvantage being that during the Awake state, Session Topic Aliases will have to be recreated, or Topic Names used instead, increasing network data usage.

Figure 4-11 – Awake PINGRESP Packet flush



4.15 Optional Features

Support for the ADVERTISE, SEARCHGW, GWINFO and PUBWOS packet types is optional.

The Forwarder Encapsulation packet type support is optional. For instance, it is not required if the MQTT-SN Clients are able to directly reach a MQTT-SN Server.

The Protection Encapsulation packet type support is optional. For instance, it is not required if the MQTT-SN Server and the MQTT-SN Clients interact over a secure communication channel, such as DTLS or any communication channel assuring the authenticity and optionally the confidentiality protection.

5 Security (Informative)

5.1 Introduction

MQTT-SN is a transport protocol specification for message transmission, allowing implementers a choice of network, privacy, authentication and authorization technologies. Since the exact security technologies chosen will be context specific, it is the implementer's responsibility to include the appropriate features as part of their design.

MQTT-SN solutions are likely to also include MQTT communications - this section should be read alongside the Security chapters in the MQTT standards: MQTT 3.1.1 and MQTT 5.0.

MQTT-SN implementations will likely need to keep pace with an evolving security landscape. This Chapter provides general implementation guidance so as not to restrict choices available. Examples of threats that solution providers should consider are:

- Devices could be compromised
- Data at rest in Clients and Servers might be accessible
- Protocol behaviors could have side effects - “timing attacks” for example
- Denial of Service (DoS) attacks
- Communications could be intercepted, altered, re-routed or disclosed
- Injection of spoofed MQTT-SN Control Packets

When MQTT-SN solutions are deployed in hostile communication environments, implementations will often need to provide mechanisms for:

- Authentication of users and devices
- Authorization of access to Server resources
- Integrity of MQTT-SN Control Packets and application data contained therein
- Privacy of MQTT-SN Control Packets and application data contained therein

In addition to technical security issues there could also be geographic (for example U.S.-EU Privacy Shield Framework [USEUPRIVSH]), industry specific (for example PCI DSS [PCIDSS]) and regulatory considerations (for example Sarbanes-Oxley [SARBANES]).

5.2 MQTT-SN solutions: security and certification

An implementation might want to provide conformance with specific industry security standards such as NIST Cyber Security Framework [NISTCSF], PCI-DSS [PCIDSS]), FIPS-140-3 [FIPS1403] and Commercial National Security Algorithm Suite (CNSA) 2.0 [CNSA20]. The use of industry proven, independently verified and certified technologies will help meet compliance requirements.

5.3 Lightweight cryptography and constrained devices

MQTT-SN is targeted at the most power saving and constrained devices. In contrast to MQTT where there is principally one underlying network technology - TCP/IP - MQTT-SN is intended to be agnostic to the underlying network as long as it conforms to the requirements outlined in 4.2 Networks and Virtual Connections.

Previous versions of MQTT-SN did not include specific security and integrity features, preferring to leave that to the underlying network. That approach is still supported, but to aid interoperability the Protection Encapsulation is introduced, in 3.17 Protection Encapsulation. In order to disassociate the security model from the rest of the MQTT-SN specification, the Protection Encapsulation allows all the packet types, including CONNECT, to be wrapped in a security envelope. Furthermore the security material is self-contained in each Protection Encapsulation envelope, so it is completely decoupled from the Virtual Connection.

The schemes defined by the Protection Encapsulation are especially suited for implementation by constrained devices:

- HMAC and AES CMAC are widely adopted authentication only standard schemes.
- The Advanced Encryption Standard [AES] is the most widely adopted encryption algorithm. There is hardware support for AES in many processors, but not commonly for embedded processors.
- The encryption algorithm ChaCha20 [CHACHA20] encrypts and decrypts much faster in software, but is not as widely available as AES.

The Protection Encapsulation approach is informed by the OSCORE [RFC 8613] standard and CBOR Initial Algorithms [RFC 9053] informational document. The ISO 29192 [ISO29192] standard makes recommendations for cryptographic primitives specifically tuned to perform on constrained, low end, devices.

The MQTT-SN Protection Encapsulation also allows for user defined protection schemes, although these will of necessity have lower interoperability compared to the built-in schemes, as implementations of both Client and Server will have to be aware of them.

5.4 Implementation notes

When the underlying network layer for MQTT-SN is UDP, DTLS [RFC9147] can be used to secure MQTT-SN communications instead of or in conjunction with the Protection Encapsulation. It is recommended that Server implementations that offer DTLS use UDP port 8883 (IANA service name: secure-mqtt).

For other underlying network technologies, a security solution particular to that technology must be found, which could involve using the MQTT-SN Protection Encapsulation and/or Authentication.

There are many security concerns to consider when implementing or using MQTT-SN. The following section should not be considered a comprehensive checklist.

An implementation might want to achieve some, or all, of the following:

5.4.1 Authentication of Clients by the Server

The CONNECT packet contains an Authentication Data field which can contain a user name and password if the Authentication Method is SASL PLAIN (see 4.11.1.2 MQTT User Name and Password Support). Implementations can choose how to make use of the content of these fields. They may provide their own authentication mechanism, use an external authentication system such as LDAP [RFC4511] or OAuth [RFC6749] tokens, or leverage operating system authentication mechanisms.

MQTT-SN provides an Authentication mechanism as described in 4.11 Authentication. Using this requires support for it in both the Client and Server.

Implementations passing authentication data in clear text, obfuscating such data elements or requiring no authentication data should be aware this can give rise to Man-in-the-Middle and replay attacks. 5.4.5 Privacy of Application Messages and MQTT-SN Control Packets introduces approaches to ensure data privacy.

A Virtual Private Network (VPN) between the Clients and Servers can provide confidence that data is only being received from authorized Clients.

Where DTLS [RFC9147] is used, X.509 Certificates sent from the Client can be used by the Server to authenticate the Client to achieve mutual authentication.

5.4.2 Authorization of Clients by the Server

If a Client has been successfully authenticated, a Server implementation should check that it is authorized before accepting its connection.

Authorization may be based on information provided by the Client such as User Name, the hostname/network address of the Client, or the outcome of authentication mechanisms.

In particular, the implementation should check that the Client is authorized to use the Client Identifier as this gives access to the MQTT-SN Session State (described in 4.1 Session state). This authorization check is to protect against the case where one Client, accidentally or maliciously, provides a Client Identifier that is already being used by some other Client.

An implementation should provide access controls that take place after CONNECT to restrict the Client's ability to publish to particular Topics or to subscribe using particular Topic Filters. An implementation should consider limiting access to Topic Filters that have broad scope, such as the # Topic Filter.

5.4.3 Authentication of the Server by the Client

The MQTT-SN protocol is not trust symmetrical. When using basic Username and Password authentication, there is no mechanism for the Client to authenticate the Server. Some forms of authentication do allow for mutual authentication.

Where DTLS is used, X.509 Certificates sent from the Server can be used by the Client to authenticate the Server.

MQTT-SN provides an Authentication mechanism as described in 4.11 Authentication, which can be used to authenticate the Server to the Client. Using this requires support for it in both the Client and Server.

A VPN between Clients and Servers can provide confidence that Clients are connecting to the intended Server.

5.4.4 Integrity of Application Messages and MQTT-SN Control Packets

Applications can independently include hash values in their Application Messages. This can provide integrity of the contents of Publish packets across the network and at rest.

DTLS and the Protection Encapsulation provide hash algorithms to verify the integrity of data sent over the network.

The use of VPNs to connect Clients and Servers can provide integrity of data across the section of the network covered by a VPN.

5.4.5 Privacy of Application Messages and MQTT-SN Control Packets

DTLS [RFC9147] can provide encryption of data sent over the network. There are valid DTLS cipher suites that include a NULL encryption algorithm that does not encrypt data. To ensure privacy Clients and Servers should avoid these cipher suites.

An application might independently encrypt the contents of its Application Messages. This could provide privacy of the Application Message both over the network and at rest. This would not provide privacy for other Properties of the Application Message such as Topic Name.

Client and Server implementations can provide encrypted storage for data at rest such as Application Messages stored as part of a Session.

The use of VPNs to connect Clients and Servers can provide privacy of data across the section of the network covered by a VPN.

5.4.6 Non-repudiation of message transmission

Application designers might need to consider appropriate strategies to achieve end to end non-repudiation.

5.4.7 Detecting compromise of Clients and Servers

Client and Server implementations using DTLS should provide capabilities to ensure that any X.509 certificates provided when initiating a DTLS session are associated with the hostname of the Client connecting or Server being connected to.

Client and Server implementations using DTLS can choose to provide capabilities to check Certificate Revocation Lists (CRLs [RFC5280]) and Online Certificate Status Protocol (OSCP) [RFC6960] to prevent revoked certificates from being used.

Physical deployments might combine tamper-proof hardware with the transmission of specific data in Application Messages. For example, a meter might have an embedded GPS to ensure it is not used in an unauthorized location. [IEEE8021AR] is a standard for implementing mechanisms to authenticate a device's identity using a cryptographically bound identifier.

5.4.8 Detecting abnormal behaviors

Server implementations might monitor Client behavior to detect potential security incidents. For example:

- Repeated connection attempts
- Repeated authentication attempts
- Abnormal termination of connections
- Topic scanning (attempts to send or subscribe to many topics)
- Sending undeliverable messages (no subscribers to the topics)
- Clients that connect but do not send data

Server implementations might delete the Virtual Connection of Clients that breach its security rules.

Server implementations detecting unwelcome behavior might implement a dynamic block list based on identifiers such as IP address or Client Identifier.

Deployments might use network-level controls (where available) to implement rate limiting or blocking based on IP address or other information.

5.4.9 Handling of Disallowed Unicode code points

1.7.4 UTF-8 Encoded String describes the Disallowed Unicode code points, which should not be included in a UTF-8 Encoded String. A Client or Server implementation can choose whether to validate that these code points are not used in UTF-8 Encoded Strings such as the Topic Name or Properties.

If the Server does not validate the code points in a UTF-8 Encoded String but a subscribing Client does, then a second Client might be able to cause the subscribing Client to delete the Virtual Connection by publishing on a Topic Name or using Properties that contain a Disallowed

Unicode code point. This section recommends some steps that can be taken to prevent this problem.

A similar problem can occur when the Client validates that the payload matches the Payload Format Indicator and the Server does not. The considerations and remedies for this are similar to those for handling Disallowed Unicode code points.

5.4.9.1 Considerations for the use of Disallowed Unicode code points

An implementation would normally choose to validate UTF-8 Encoded strings, checking that the Disallowed Unicode code points are not used. This avoids implementation difficulties such as the use of libraries that are sensitive to these code points, it also protects applications from having to process them.

Validating that these code points are not used removes some security exposures. There are possible security exploits which use control characters in log files to mask entries in the logs or confuse the tools which process log files. The Unicode Noncharacters are commonly used as special markers and allowing them into UTF-8 Encoded Strings could permit such exploits.

5.4.9.2 Interactions between Publishers and Subscribers

The publisher of an Application Message normally expects that the Servers will forward the message to subscribers, and that these subscribers are capable of processing the messages.

These are some conditions under which a publishing Client can cause the subscribing Client to delete the Virtual Connection. Consider a situation where:

- A Client publishes an Application Message using a Topic Name containing one of the Disallowed Unicode code points.
- The publishing Client library allows the Disallowed Unicode code point to be used in a Topic Name rather than rejecting it.
- The publishing Client is authorized to send the publication.
- A subscribing Client is authorized to use a Topic Filter which matches the Topic Name. Note that the Disallowed Unicode code point might occur in a part of the Topic Name matching a wildcard character in the Topic Filter.
- The Server forwards the message to the matching subscriber rather than disconnecting the publisher.
- In this case the subscribing Client might:
    - Delete the Virtual Connection because it does not allow the use of Disallowed Unicode code points, possibly sending a DISCONNECT before doing so. For QoS 1 and QoS 2 messages this might cause the Server to send the message again, causing the Client to delete the Virtual Connection again.
    - Reject the Application Message by sending a Reason Code greater than or equal to 0x80 in a PUBACK (QoS 1) or PUBREC (QoS 2).
    - Accept the Application Message but fail to process it because it contains one of the Disallowed Unicode code points.

- Successfully process the Application Message.

The potential for the Client to delete the Virtual Connection might go unnoticed until a publisher uses one of the Disallowed Unicode code points.

5.4.9.3 Remedies

If there is a possibility that a Disallowed Unicode code point could be included in a Topic Name or other Properties delivered to a Client, the solution owner can adopt one of the following suggestions:

1. Change the Server implementation to one that rejects UTF-8 Encoded Strings containing a Disallowed Unicode code point either by sending a Reason Code greater than or equal to 0x80 or deleting the Virtual Connection.
2. Change the Client library used by the subscribers to one that tolerates the use of Disallowed Code points. The client can either process or discard messages with UTF-8 Encoded Strings that contain Disallowed Unicode code points so long as it continues the protocol.

5.4.10 Other security considerations

If Client or Server X.509 certificates are lost or it is considered that they might be compromised they should be revoked (using CRLs [RFC5280] and/or OSCP [RFC6960]).

Client or Server authentication credentials, such as User Name and Password, that are lost or considered compromised should be revoked and/or reissued.

In the case of long lasting connections:

- Where applicable, Client and Server implementations should allow for session renegotiation to establish new cryptographic parameters (replace session keys, change cipher suites, change authentication credentials).
- Servers may close the Virtual Connection of Clients and require them to re-authenticate with new credentials.
- Servers may require their Client to reauthenticate periodically using the mechanism described in 4.11.1.1 Re-authentication.

Clients connected to a Server have a transitive trust relationship with other Clients connected to the same Server and who have authority to publish data on the same topics.

5.4.11 Use of SOCKS

Implementations of Clients should be aware that some environments will require the use of SOCKSv5 [RFC1928] proxies to transmit data. Some MQTT-SN implementations could make use of alternative secured tunnels through the use of SOCKS. Where implementations choose to use SOCKS, they should support both anonymous and User Name, Password authenticating SOCKS proxies. In the latter case, implementations should be aware that SOCKS

authentication might occur in plain-text and so should avoid using the same credentials for connection to an MQTT-SN Server.

5.4.12 Security profiles

Implementers and solution designers might wish to consider security as a set of profiles which can be applied to the MQTT-SN protocol. An example of a layered security hierarchy is presented below.

5.4.12.1 Clear communication profile

When using the clear communication profile, the MQTT-SN protocol runs over an open network with no additional secure communication mechanisms in place.

5.4.12.2 Secured network communication profile

When using the secured network communication profile, the MQTT-SN protocol runs over a physical or virtual network which has security controls, VPNs or physically secure networks for example.

5.4.12.3 Secured transport profile

When using the secured transport profile, the MQTT-SN protocol runs over a physical or virtual network and uses MQTT-SN Authentication, Protection Encapsulation, DTLS and/or other technologies to provide authentication, integrity and privacy.

DTLS Client authentication can be used in addition to – or in place of – MQTT-SN Client authentication as provided by the Authentication Method and Data fields.

5.4.12.4 Industry specific security profiles

It is anticipated that the MQTT-SN (and MQTT) protocols will be designed into industry specific application profiles, each defining a threat model and the specific security mechanisms to be used to address these threats. Recommendations for specific security mechanisms will often be taken from existing works including:

[NISTCSF] NIST Cyber Security Framework
[NIST7628] NISTIR 7628 Guidelines for Smart Grid Cyber Security
[FIPS1403] Security Requirements for Cryptographic Modules (FIPS PUB 140-3)
[PCIDSS] PCI-DSS Payment Card Industry Data Security Standard
[CNSA20] Commercial National Security Algorithm Suite (CNSA) 2.0
[NSAB] NSA Suite B Cryptography

6 Conformance

(**Note**: The OASIS TC Process 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.

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

Remove this note before submitting for publication.)

Appendix A. Acknowledgments

[Required section.]

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 these yellow notes before submitting for publication.

A.1 Special Thanks

Note: This is an optional subsection to call out contributions from TC members. If a TC wants to thank non-TC members then they should avoid using the term "contribution" and instead thank them for their "expertise" or "assistance".

Substantial contributions to this document from the following individuals are gratefully acknowledged:

[Participant Name, Affiliation | Individual Member]

A.2 Participants

Note: A TC can determine who they list here, however, Observers must not be listed. It is common practice for TCs to list everyone that was part of the TC during the creation of the document, but this is ultimately a TC decision on who they want to list and not list.

The following individuals were members of this Technical Committee during the creation of this document and their contributions are gratefully acknowledged:

[Participant Name, Affiliation | Individual Member]

Appendix B. Mandatory normative statements (informative)

Appendix C. Implementation Guidance (Informative)

C.1 Example MQTT-SN Architectures

Among the kinds of MQTT-SN components, there are *Clients and Servers* (sub-divided into *Gateways, Brokers and Forwarders)*.

MQTT-SN Clients can:

1. connect to a Server
2. communicate with an MQTT Server through an MQTT-SN Gateway
3. send and receive messages to and from other MQTT-SN Clients through a Server which is acting as an MQTT-SN Broker
4. send and receive messages without connecting to a Server by using PUBWOS packets.

An MQTT-SN Server may or may not communicate with an MQTT Server. An MQTT-SN Gateway is a Server which connects to an MQTT Server back end. An MQTT-SN Gateway uses the MQTT protocol between itself and the MQTT Server. A Server which acts as an intermediary between MQTT-SN Clients is called a Broker. If a Server does not act as a Broker itself but is connected to an MQTT Server, the Gateway's main function is the translation between MQTT and MQTT-SN.

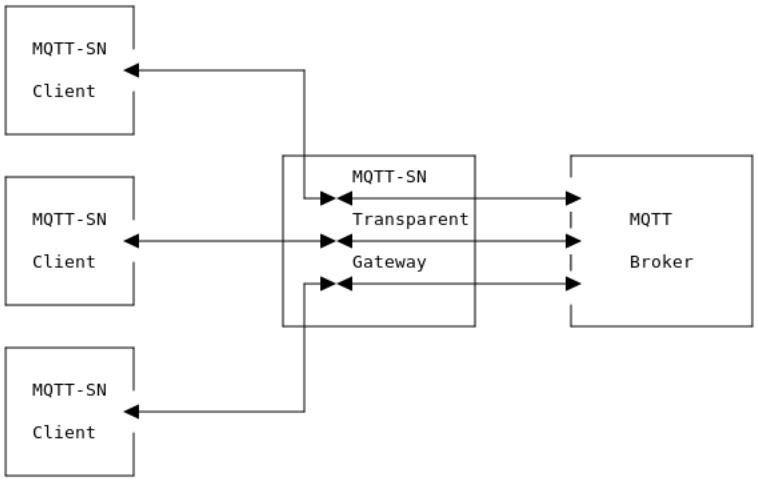
If the Gateway is not directly attached to the Clients' network, MQTT-SN Clients can communicate with a Gateway through an MQTT-SN Forwarder. The forwarder encapsulates (see 3.18 Forwarder Encapsulation) the MQTT-SN frames it receives on the Client side and forwards them unchanged to the Gateway; in the opposite direction, it removes the encapsulation from the frames it receives from the Gateway and sends them unchanged to the Clients.

C.1.1 Transparent Gateway

For each connected MQTT-SN Client a Transparent Gateway will set up and maintain an 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 MQTT Server. There will be as many MQTT connections between the Gateway and the MQTT Server as MQTT-SN clients connected to the Gateway. The Transparent Gateway will perform a translation between the two protocols. Since all packet exchanges are end-to-end between the MQTT-SN client and the MQTT Server, functions and features that are implemented by the MQTT Server can be offered to the MQTT-SN Client.

Although the implementation of the Transparent Gateway may be somewhat simpler than an Aggregating Gateway, 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.

Figure C-1 – Transparent Gateway

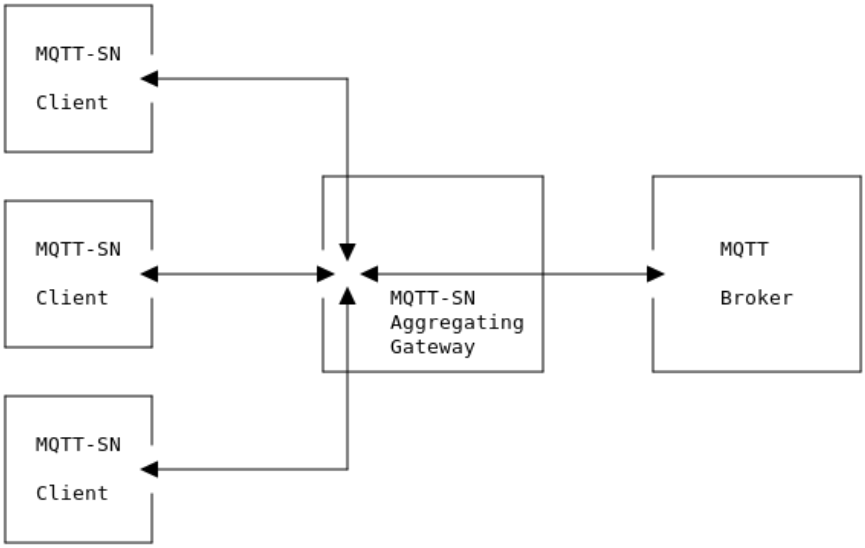


Because PUBWOS packets could be sent at any time by Clients with no Virtual Connection, a Transparent Gateway would need to maintain a dedicated MQTT connection with the MQTT Server to support those packets.

C.1.2 Aggregating Gateway

Instead of having one MQTT connection for each connected MQTT-SN Client, an aggregating Gateway has one MQTT connection to the MQTT Server. All packet exchanges between an MQTT-SN client and an aggregating Gateway end at the Gateway. The Gateway then decides which information will be given further to the MQTT Server. Although its implementation may be more complex than a transparent Gateway, an aggregating Gateway reduces the number of MQTT connections between the Gateway and MQTT Server.

Figure C-2 – Aggregating Gateway



To support PUBWOS packets from MQTT-SN clients without a Virtual Connection, an Aggregating may use any aggregating MQTT connection to forward those packets to an MQTT Server.

A hybrid Gateway may contain elements of both Aggregating and Transparent Gateways, using different approaches depending on the characteristics of the MQTT-SN Clients connecting to them.

C.1.3 Forwarder

An MQTT-SN Forwarder connects two networks which cannot transmit messages directly to and from each other. It serves as a bridge for MQTT-SN messages between the two networks, allowing MQTT-SN Clients in one to connect to an MQTT-SN Gateway in the other. The two networks could be Zigbee on one side and UDP on the other, for instance.

The following diagrams illustrate how a Forwarder may interact with an Aggregating or Transparent Gateway.

Figure C-3 – Forwarder with Transparent Gateway

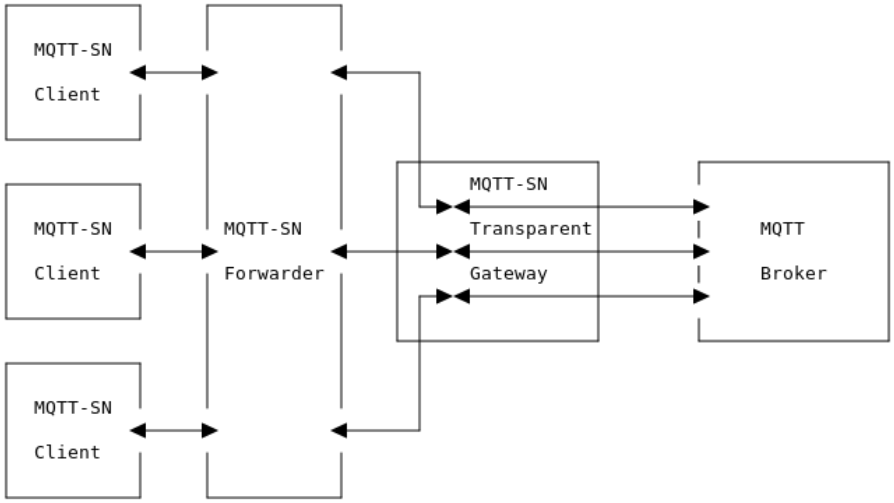
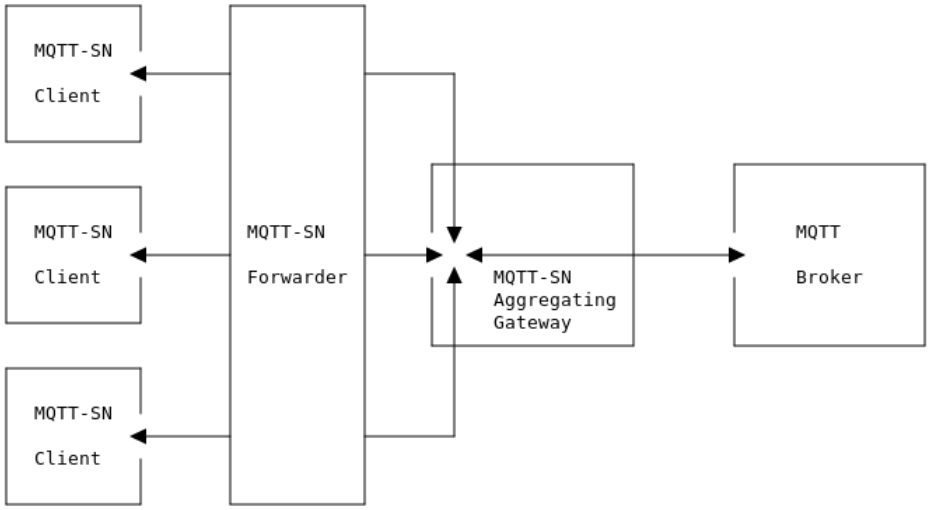


Figure C-4 – Forwarder with Aggregating Gateway

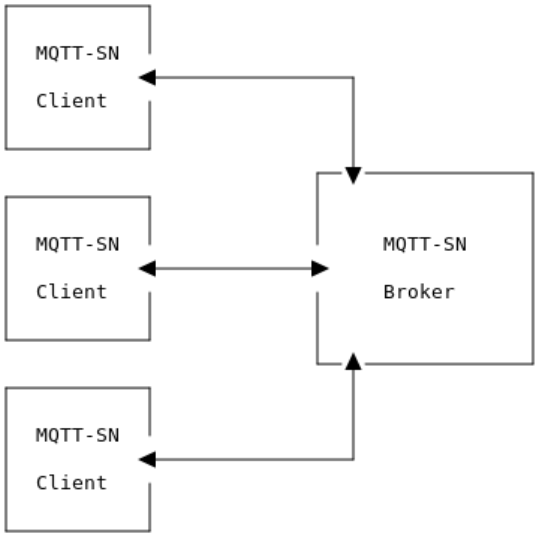


C.1.4 MQTT-SN Broker

An MQTT-SN Server may have no interaction with an MQTT Server, in which case, much like an MQTT Server, it will act as an intermediary between MQTT-SN Clients.

It will allow MQTT-SN Clients to set up subscriptions, and publish messages to other clients which have subscribed to the relevant topics. It may support the receipt and sending of PUBWOS packets - it is an implementation decision on how to handle them.

Figure C-5 – MQTT-SN Broker



An MQTT-SN Server may choose to incorporate elements of a Broker, Aggregating and Transparent Gateway together. Typically, an Aggregating Gateway will also act as an MQTT-SN Broker.

C.2 Server Congestion

For CONNECT, PUBLISH, SUBSCRIBE and REGISTER requests, the Server may return the Reason Code *Congestion*, meaning *try again later*.

The requester should wait a reasonable amount of time (*Congestion Delay)* before sending a new request to the Server. What constitutes *a reasonable amount of time* depends on the implementation characteristics - it should be configured in the client application based on those. See C.4 Timer and Counter Values for an example value.

C.3 Example Timer and Counter Values

Figure C-6 gives some values for the timers and counters defined in this specification derived from implementation experience.

Figure C-6 – Best practice values for timers and counters

Name	Where Defined	Example Value
Advertise Duration	C.6 Gateway Advertisement and Discovery	Greater than 15 minutes
Advertise Count	C.6 Gateway Advertisement and Discovery	2 -3
SEARCHGW Delay	C.6 Gateway Advertisement and Discovery	5 seconds
GWINFO Delay	C.6 Gateway Advertisement and Discovery	5 seconds
Congestion Delay	C.2 Server Congestion	Greater than 5 minutes
Retry Interval	4.4.2 Unacknowledged Packets	Implement C.5 Exponential Backoff with a starting value of 1 second after an initial wait period of 5 seconds. So the first retry will be ~6 seconds.
Max. Retry Count	4.4.2 Unacknowledged Packets	3 – 5
Max. Retry Interval	C.4 Exponential Backoff	60 seconds

To balance reaction speed with reliability, the tolerance of the sleep timers at the Server may depend on the values indicated by the clients. For example, the timer values may be 10% higher than the indicated values for periods larger than 1 minute, and 50% higher if less.

C.4 Exponential Backoff

The *Retry Interval* for unacknowledged packets can be increased on each retry, to avoid overwhelming recipient network nodes while allowing efficient Virtual Connection reestablishment. The client periodically retries a failed packet with increasing delays between attempts, constrained by a Maximum Retry Interval, interleaved with a suitable seed of randomness.

Algorithm:

This algorithm retries requests at doubling intervals, increasing time between retries up to a *Maximum Retry Interval*. For example:

1. Send initial packet sent. The initial *Retry Interval* is 1000 ms.
2. Wait up to 1000 + (random number) ms - retry the operation if no response is received during that time.
3. Wait up to 2000 + (random number) ms - retry the operation if no response is received during that time.
4. Wait up to 4000 + (random number) ms - retry the operation if no response is received during that time.
5. Continue increasing the *Retry Interval* each time, but no larger than *Max. Retry Interval.*
6. Continue waiting and retrying up to *Max. Retry Count* number of retries, without increasing the *Retry Interval* further.

The wait time is (ran is a random number, max is *Max. Retry Interval*):

$$\min(((2^n * sf) + ran), max)$$

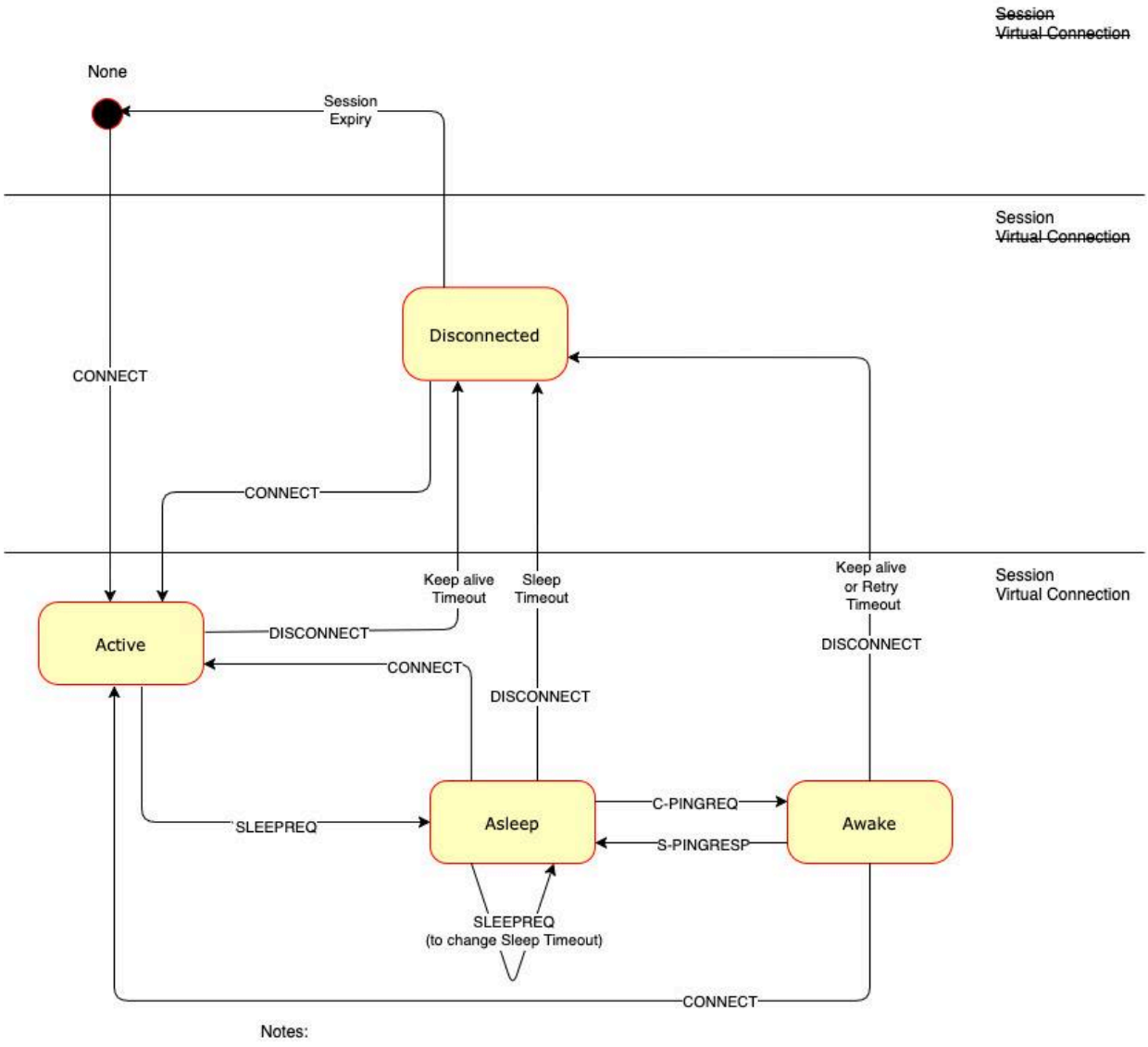
with n incremented by 1 for each iteration (or operation) and the scaling factor (sf) being set to some reasonable value - for example 1000 as in the example above.

The random number helps to avoid cases where many clients are synchronized by some situation, and all retry at once. The value of the random number ran is recalculated after each retry. The random number should be no larger than the initial *Retry Interval*.

C.5 Client State Diagrams

The following diagrams are illustrative, graphical views of the states and transitions. They are not comprehensive but included for guidance.

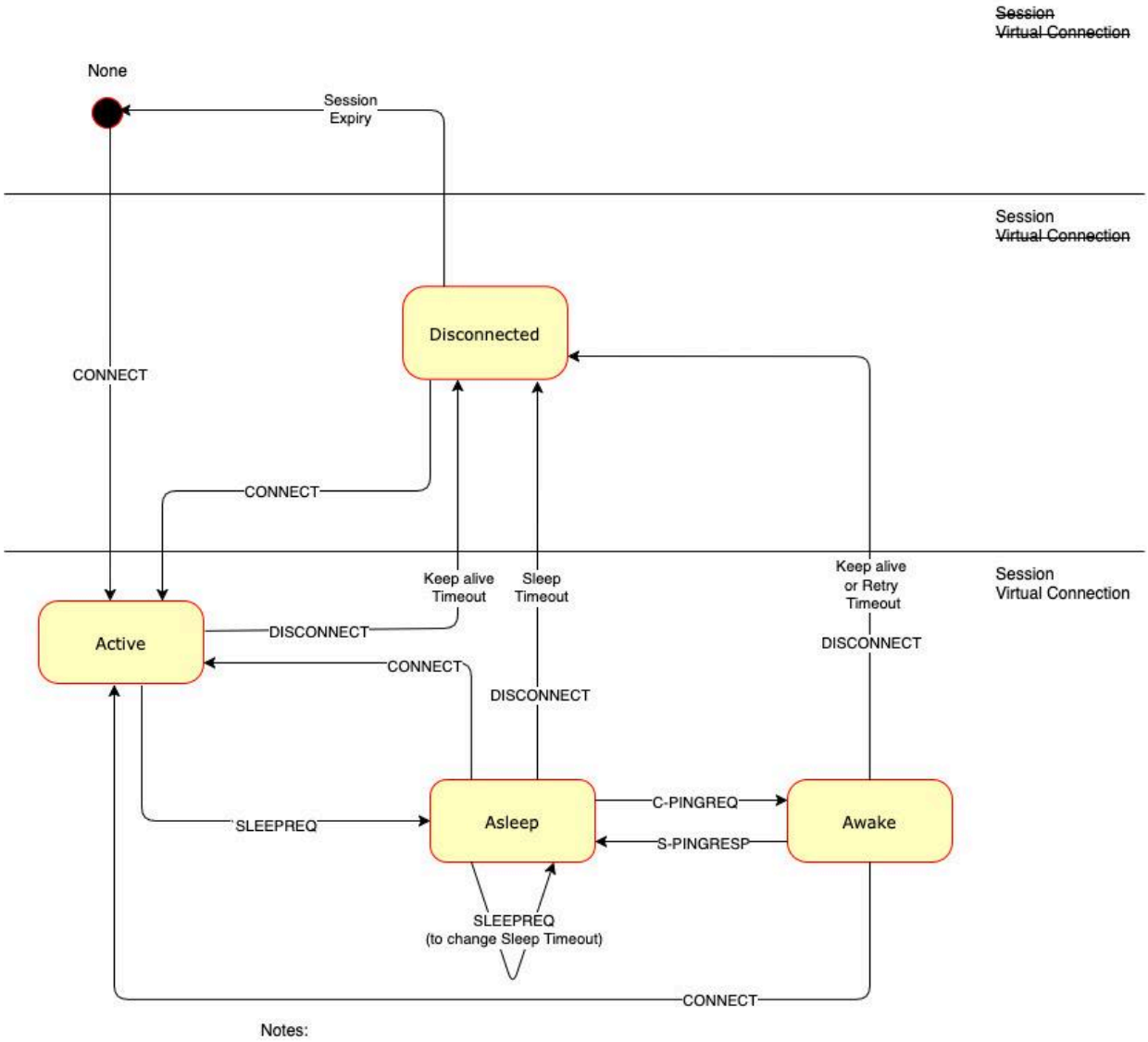
Figure 4-8 – Server View of Client States - informative



Notes:

1. All the DISCONNECT packets shown are sent by the Client.
2. The Timeout transitions can be indicated by the Server sending a DISCONNECT packet to the Client with an appropriate reason code.
3. DISCONNECT packets can also be sent at any time by the Server in the Active, Asleep and Awake states, for instance if the Server is shutting down.

Figure 4-8 – Server View of Client States - informative



Notes:

1. All the DISCONNECT packets shown are sent by the Client.
2. The Timeout transitions can be indicated by the Server sending a DISCONNECT packet to the Client with an appropriate reason code.
3. DISCONNECT packets can also be sent at any time by the Server in the Active, Asleep and Awake states, for instance if the Server is shutting down.

C.6 PUBLISH with QoS -1

Figure C-7 – PUBLISH Packet for QoS -1

Byte \ Bit	7	6	5	4	3	2	1	0
1	Length							
2	Packet Type							
Publish QoS -1 Flags								
	DUP	QoS		Retain	Reserved		Topic Type	
3	0	1	1	X	0	0	X	X
Topic Alias - *only present when Topic Type is Predefined Topic Alias*								
4	Topic Alias MSB							
5	Topic Alias LSB							
Topic Short Name - *only present when Topic Type is Topic Short Name*								
4	Topic Short Name 1							
5	Topic Short Name 2							
Topic Name Length - *only present when Topic Type is Topic Name*								
4	Topic Name Length MSB							
5	Topic Name Length LSB							
6	0x00 - Fixed Field Value							
7	0x00 - Fixed Field Value							
8	Topic Name *(N) - only present when Topic Type is Topic Name*							
8+N	Payload							

This packet is the MQTT-SN 1.2 equivalent of PUBWOS. It could be supported by a Server if there are existing MQTT-SN 1.2 transmitters that the Server wants to listen to, or receivers it wants to send to. Implementation of this packet is optional.

This packet can be used by both Clients and Servers to publish data to a topic without establishing a Virtual Connection or Session.

C.6.1 PUBLISH Header

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to 2.1 Structure of an MQTT-SN Control Packet for a detailed description.

C.6.2 PUBLISH Flags

The PUBLISH Flags is a 1 byte field which contains flags specifying the content of the packet and the Server behavior. Bits 3-2 of the PUBLISH Flags are reserved and are set to 0.

The Client validates that the reserved flags in the PUBLISH packet are set to 0. If any of the reserved flags is not 0 it is a Malformed Packet.

C.6.2.1 Topic Type

Position: bits 0 and 1 of the PUBLISH Flags.

This determines the format of the Topic Data field.

The Topic Type in MQTT-SN 1.2 is different to that in MQTT-SN 2.0. The values applicable to PUBLISH QoS -1 are:

- 0b00 - Topic Name. Its length is defined in the Topic Name Length field.
- 0b01 - Predefined Topic Alias.
- 0b10 - Short Topic Name. A two byte Topic Name, with the same syntax as Topic Name. However, some 1.2 implementations treated this as a binary field.

C.6.2.2 QoS

Position: bits 5 and 6 of the PUBLISH Flags.

Set this field to “0b11” for QoS -1.

C.6.2.3 DUP

Position: bit 7 of the PUBLISH Flags.

Set to 0.

C.6.2.4 Retain

Position: bit 4 of the PUBLISH Flags.

This flag signifies whether the message is published as a retained message or not. See 4.13 Retained Messages for more information.

C.6.3 Topic Alias

Only present if the Topic Type is Predefined Topic Alias. Contain a Topic Alias which is preconfigured to be known to both the sender and receiver.

C.6.4 Topic Short Name

Only present if the Topic Type is Short Topic Name.

This is a two byte Topic Name. This Topic Type does not exist in later versions of MQTT-SN. It existed because the original MQTT-SN 1.2 did not allow a Long Topic Name, so the only other option for this packet was a Predefined Topic Alias.

C.6.5 Topic Name Length

Only present if the Topic Type is Topic Name.

The length of the Topic Name field.

C.6.6 Topic Name

Only present if the Topic Type is Topic Name.

Topic Name is a UTF-8 encoded string of length Topic Name Length.

C.6.7 Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. It is valid for a PUBLISH packet to contain a zero length Payload.

C.6.8 PUBLISH with QoS -1 Actions

The Client or Server uses a PUBLISH QoS -1 packet to send an Application Message to a Network Address, for possible receipt by a Server or another Client.

If received by a Client or Server, the PUBLISH QoS -1 packet is treated as if its QoS were 0 as described in 3.6.3.7 PUBLISH Actions.

C.7 Gateway Advertisement and Discovery

Clients might have foreknowledge of how to reach a Gateway, but in dynamic networks they may not. MQTT-SN supports mechanisms to allow Clients to find available MQTT-SN Gateways. This support is optional - it may not be needed. In some implementations, the underlying network technology might be used for this purpose instead.

In MQTT-SN there are two principal ways for Clients and Gateways to find each other:

1. The Gateway can periodically transmit, or broadcast, an ADVERTISE packet to its neighborhood.
2. The Client can elicit a response from one or more Gateways, or Clients which know the network location of a Gateway, by broadcasting a SEARCHGW packet. The response, from either a Gateway or a Client on the Gateway's behalf, is a GWINFO packet.

A Gateway should only advertise its presence, or respond to SEARCHGW requests, if it is able to accept subscriptions and forward messages. For instance, a Transparent Gateway which is not currently connected to an MQTT Server, should not advertise.

Multiple Gateways may be active at the same time in the same network, in which case they will have different identifiers. It is up to the Client to decide to which Gateway it wants to connect.

A Client can maintain a list of active Gateways together with their network addresses. This list is populated with the information from ADVERTISE and GWINFO packets received.

The time until the Gateway sends the next ADVERTISE packet is indicated in the *Duration* field of the ADVERTISE packet (*Advertise Duration*). 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 several times (*Advertise Count*) consecutively, it may assume that the Gateway is down and remove it from its list of active Gateways. Similarly, Gateways in stand-by mode can become active (and start sending ADVERTISE packets) if they fail to observe successive advertisements from a previously active Gateway.

If the ADVERTISE packets are broadcast into the whole wireless network, the time interval between two consecutive ADVERTISE packets sent by a gateway should be large enough (greater than 15 minutes for example) to avoid bandwidth congestion in the network.

A large interval between ADVERTISE packets can lead to a long waiting time for new Clients which are looking for a Gateway. To shorten this waiting time a client may send a SEARCHGW packet. To prevent network flooding when multiple clients start searching for a Gateway almost at the same time, the sending of the SEARCHGW packet can be delayed by a random time *SEARCHGW Delay*. A client can 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.

Upon receiving a SEARCHGW packet a Gateway replies with a GWINFO packet containing its identifier. Similarly, a Client can answer with a GWINFO packet if it has at least one item in its active Gateway list. If the Client has multiple Gateways in its list, it can select one Gateway out of its list and include that information in the GWINFO packet.

To give priority to Gateways a client delays its sending of the GWINFO packet for a random time *GWINFO Delay*. If during this delay the Client receives a GWINFO packet it cancels the sending of its own GWINFO packet.

If there is no response, the SEARCHGW packet may be retransmitted. In this case the time intervals between consecutive SEARCHGW packets should be increased by an exponential backoff algorithm such as that described in C.4 Exponential Backoff.

Appendix D. Revision History (informative)

[Optional section.]

Revisions made since the initial stage of this numbered Version of this document may be tracked here.

Note: If revision tracking is handled in another system like github, provide a link to it instead of using this table, if desired. Remove this note before submitting for publication.

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]
WD-10	[October 2021]	[Simon Johnson]	[Document format aligned with core specification, removal of introduction, addition of packet ID table, adding error code]

WD-11	[October 2021]	[Simon Johnson]	[MQTT-SN Architecture moved into operational behavior, removal of variable integer definition, addition of session state section, normative comments added to sleeping client operational behaviour]
WD-12	[November 2021]	[Andrew Banks]	Rework 1.5 Background
WD-13	[November 2021]	[Simon Johnson]	[Move Authentication and Retained messages into operational behavior, rationalized tables and figures, separated packet definitions of similar structures into distinct sections.]
WD-14	[Decmeber 2021]	[Simon Johnson]	[First implementation attempt, Fixed table references, Fixed PingResp packet]
WD-15	[December 2021]	[Simon Johnson]	[Tara added as editor, return code additions]
WD-15	[February 2022]	[Tara Walker]	Changed Return Code nomenclature to be more consistent w/5.0. Added Reason Codes to each control packet type
WD-16	March 2022	[Tara Walker]	Updated WILL*Types to correct Packet Type. Added Global Flags Table to Section 2. Updated each Control Packet Flags in Section 3 adding missing Flag Sections. Formatting: Auto update of Table numbering, Auto update of WD Revision numbering for footer.
WD-17	April 2022	[Simon Johnson]	Updated use of topic name and topic filter to be aligned with MQTT 5. Topic alias becomes topic alias type. Added quality of service protocol flow as it differed to MQTT 5 (inflight). Conformance references removed as these will need to be wholly owned OR externally referenced.
WD-18	June 2022	[Tara E. Walker]	Updated items based upon the feedback from Alex Kritikos.
WD-19	August 2022	[Simon Johnson]	Remove change tracking as document was becoming unworkable.

WD-20	September 2022	[Simon Johnson]	Integrate feedback from committee meeting relating to the work by Miroslav Prymek. Added resolution of CONNACK session present per MQTT 585
WD-21	October 2022	[Simon Johnson]	Client States section added to describe the 5 states. Updated the state transition diagram to accommodate new disconnect field and new transitions between Awake -> Lost and Asleep -> Disconnected. Security section added. Figure 2 – MQTT-SN Architecture diagram updated. Font updated to Arial from bespoke font. QoS -1 – Section added to the QoS chapter (NOTE: updated text to allow for bi-directional -1 PUBLISHING). Introduction of Exponential backoff algorithm. Applied issue issue 587 (max messages set in CONNECT flags).
WD-22	November 2022	[Simon Johnson]	Integrate MQTT 591 (sleep behavior) Replace instances of “return code” to “reason code” PINGREQ timeout aligned with Tretry (15 seconds) from the ill defined “reasonable amount of time” Exponential Algo fix (using the factor n assuming it was the product!) Client Identifier size clarification. Publish variants added; distinguish variant based on QoS field to save 2 bytes for single flight PUBLISH packets. Incorporated B4. Into retry timer.
WD-23	December 2022	[Simon Johnson]	CONNECT Client Identifier Informative and Normative définition update. CONNACK Client Identifier Informative and Normative définition update.

			CONNACK reason codes updated. KeepAlive boundary specified removing 0 as an option per the committee call. Added Session Expiry “reasonable” setting statement. Added sequence diagrams for CONNECT, CONNECT with WILL, CONNECT with AUTH. Network Connection Section (IANA Omitted but we need to add this to agenda)
WD-24	December 2022	[Simon Johnson, Davide Lenzarini, Ian Craggs]	Removal of Network Connection references. Modified PUBLISH -1 & 0 tables to remove topic length field Modified PUBLISH 1 & 2 tables to remove topic length field Changed Data field description on the above Updated sleeping device section Ensured the references to the Packet Length and type section was consistent in all packet types.
WD-25	January 2023	[Simon Johnson]	Broken out PUBLISH -1 into its own packet type Disconnect flags field moved and added existence flags for optional fields Introduction titles changed to better sign post where the information resides in the document
WD-26	May 2023	[Simon Johnson, Davide Lenzarini]	Backwards compatible PUBLISH -1, new OOS Publish message to repace it. Removal of security section to allow to rewrite.
WD-27	November 2023	[Simon Johnson, Davide Lenzarini]	Network Transport Layer chapter updated to define the impact of lower layers features on the MQTT-SN protocol.

			Replaced the term MQTT-SN “connection” with the term “Virtual Connection”.
WD-28	December 2023	[Davide Lenzarini, Stefan Hagen]	Ensured document structure is intact and replaced table footnotes with simple text tags and a subsequent notes listing.
	February 2024	[Ian Craggs, Simon Johnson]	Issue 560 resolution - full reason code table and add reason code fields to PUBREC, PUBREL and PUBCOMP. Duplicate reason code tables removed from packet descriptions. Will Data Sent in CONNECT Auth Data Sent in CONNECT & CONNACK Suback granted QoS 0,1,2 now reason codes not flags. Moved PUBLISH -1 to new Backward compatibility appendix
WD-29	March 2024	[Ian Craggs, Davide Lenzarini, Simon Johnson]	Update Terminology section. Add Operational Behavior sections from MQTT 5.0. Workshop revisions to Operational Behavior responding to Davide’s review.
	June 2024	[Ian Craggs]	Change CorrelId to Packet id. Move Virtual Connection semantics to Operational Behavior. Clarify MQTT-SN does not deduplicate. Do not preclude Unicast for PUBWOS.
	July 2024	[Ian Craggs]	Add Actions sections for all packets. Add a retained messages section. Add a flow control section.
	August 2024	[Ian Craggs, Simon Johnson]	Add WAKEUP packet. Update will firing conditions. Update retained messages terminology.
	September 2024	[Ian Craggs]	Remove one index level from the packet chapter. Disallow other packets before CONNACK.
	October 2024	[Ian Craggs]	Add SLEEPREQ and SLEEPRESP packets - disconnect now has no

			response. Rename Enhanced Authentication to Authentication. Add Gateway timer table. Add Security Chapter.
	November 2024	[Ian Craggs]	Add MQTT-BASIC authentication method. Move Appendices to main doc, except those that exist in MQTT.
	February 2025	[Ian Craggs]	Make session expiry in CONNACK optional.
	March 2025	[Ian Craggs]	Reformat all packet diagrams. Remove Topic Short Name. Simplify UN/SUBSCRIBE packet diagrams. Make session expiry in CONNECT optional.
	April 2025	[Ian Craggs]	Make Topic Alias in SUBACK optional. Move Retain Registrations from DISCONNECT to SLEEPREQ. Move informative Operational Behavior sections to Appendices.
	May 2025	[Ian Craggs]	Move Advertise, GWINFO and SEARCHGW to the end of Chapter 3. Move C.2 Advertisement and Discovery Appendix to end of Appendix C. Redraft Client States section. Add Server Keepalive to CONNACK.
	June 2025	[Ian Craggs]	Changed link format from section n.n to n.n xxxx. Updated PINGREQ actions. Reordered Control Packet Type to align with MQTT.