

MQTT Version 3.1.1

Committee Specification Draft 01 / Public Review Draft 01

12 December 2013

Specification URIs

This version:

<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd01/mqtt-v3.1.1-csprd01.doc> (Authoritative)
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd01/mqtt-v3.1.1-csprd01.html>
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd01/mqtt-v3.1.1-csprd01.pdf>

Previous version:

N/A

Latest version:

<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.doc> (Authoritative)
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf>

Technical Committee:

OASIS Message Queuing Telemetry Transport (MQTT) TC

Chairs:

Raphael J Cohn (raphael.cohn@stormmq.com), Individual
Richard J Coppen (coppen@uk.ibm.com), IBM

Editors:

Andrew Banks (Andrew_Banks@uk.ibm.com), IBM
Rahul Gupta (rahul.gupta@us.ibm.com), IBM

Abstract:

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet Of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections. Its features include:

- Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message delivery:
 - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
 - "At least once", where messages are assured to arrive but duplicates may occur.

- "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.
- A small transport overhead and protocol exchanges minimized to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

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 version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/mqtt/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/mqtt/ipr.php>).

Citation format:

When referencing this specification the following citation format should be used:

[mqtt-v3.1.1]

MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 12 December 2013. OASIS Committee Specification Draft 01 / Public Review Draft 01. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd01/mqtt-v3.1.1-csprd01.html>.

Notices

Copyright © OASIS Open 2013. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

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

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

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

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 Committee Specification or OASIS Standard, 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 specification.

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 specification 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 specification. 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 document 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 Committee Specification or OASIS Standard, 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 specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

Table of Contents

1	Introduction.....	7
1.1	Terminology.....	7
1.2	Normative references.....	8
1.3	Non normative references.....	9
	Acknowledgements.....	11
1.4	Data representations.....	12
1.4.1	Bits.....	12
2	MQTT Control Packet format.....	15
2.1	Fixed header.....	15
2.1.1	MQTT Control Packet types.....	15
2.1.2	Flags.....	16
2.2	Remaining Length.....	18
2.3	Variable header.....	20
2.3.1	Packet Identifier.....	20
2.3.2	Payload.....	21
3	MQTT Control Packets.....	22
3.1	CONNECT – Client requests a connection to a Server.....	22
3.1.1	Fixed header.....	22
3.1.2	Variable header.....	22
3.1.3	Payload.....	27
3.1.4	Response.....	29
3.2	CONNACK – Acknowledge connection request.....	29
3.2.1	Fixed header.....	30
3.2.2	Variable header.....	30
3.2.3	Payload.....	31
3.3	PUBLISH – Publish message.....	31
3.3.1	Fixed header.....	31
3.3.2	Variable header.....	32
3.3.3	Payload.....	33
3.3.4	Response.....	33
3.3.5	Actions.....	33
3.4	PUBACK – Publish acknowledgement.....	33
3.4.1	Fixed header.....	33
3.4.2	Variable header.....	34
3.4.3	Payload.....	34
3.4.4	Actions.....	34
3.5	PUBREC – Publish received (QoS 2 publish received, part 1).....	34
3.5.1	Fixed header.....	34
3.5.2	Variable header.....	35
3.5.3	Payload.....	35
3.5.4	Actions.....	35
3.6	PUBREL – Publish release (QoS 2 publish received, part 2).....	35
3.6.1	Fixed header.....	35

3.6.2	Variable header	36
3.6.3	Payload.....	36
3.6.4	Actions.....	36
3.7	PUBCOMP – Publish complete (QoS 2 publish received, part 3)	36
3.7.1	Fixed header.....	36
3.7.2	Variable header	37
3.7.3	Payload.....	37
3.7.4	Actions.....	37
3.8	SUBSCRIBE - Subscribe to topics	37
3.8.1	Fixed header.....	37
3.8.2	Variable header	38
3.8.3	Payload.....	38
3.8.4	Response	39
3.9	SUBACK – Subscribe acknowledgement.....	40
3.9.1	Fixed header.....	40
3.9.2	Variable header	41
3.9.3	Payload.....	41
3.10	UNSUBSCRIBE – Unsubscribe from topics	42
3.10.1	Fixed header.....	42
3.10.2	Variable header	42
3.10.3	Response	43
3.11	UNSUBACK – Unsubscribe acknowledgement.....	44
3.11.1	Fixed header.....	44
3.11.2	Variable header	44
3.11.3	Payload.....	44
3.12	PINGREQ – PING request	44
3.12.1	Fixed header.....	45
3.12.2	Variable header	45
3.12.3	Payload.....	45
3.12.4	Response	45
3.13	PINGRESP – PING response	45
3.13.1	Fixed header.....	45
3.13.2	Variable header	46
3.13.3	Payload.....	46
3.14	DISCONNECT – Disconnect notification.....	46
3.14.1	Fixed header.....	46
3.14.2	Variable header	46
3.14.3	Payload.....	46
3.14.4	Response	46
4	Operational behavior	48
4.1	Storing state.....	48
4.2	Network Connections.....	48
4.3	Quality of Service levels and flows	49
4.3.1	QoS 0: At most once delivery.....	49
4.3.2	QoS 1: At least once delivery	49

4.3.3 QoS 2: Exactly once delivery	50
4.4 Message delivery retry.....	51
4.5 Message receipt	51
4.6 Message ordering	51
4.7 Topic Names and Topic Filters	52
4.7.1 Topic wildcards.....	52
4.7.2 Topics beginning with \$.....	53
4.7.3 Topic semantic and usage	54
4.8 Handling protocol violations	55
5 Security.....	56
5.1 MQTT solutions: security and certification.....	56
5.2 Lightweight cryptography and constrained devices.....	57
5.3 Implementation notes	57
5.3.1 Authentication of Clients by the Server	57
5.3.2 Authorization of Clients by the Server	58
5.3.3 Authentication of the Server by the Client.....	58
5.3.4 Integrity of Application Messages and Control Packets	58
5.3.5 Privacy of Application Messages and Control Packets	58
5.3.6 Non-repudiation of message transmission.....	59
5.3.7 Detecting compromise of Clients and Servers	59
5.3.8 Detecting abnormal behaviors.....	59
5.3.9 Other security considerations.....	60
5.3.10 Use of SOCKS	60
5.3.11 Security profiles	61
6 Using WebSocket as a network transport.	63
7 Conformance.....	64
7.1 Conformance Targets	64
7.1.1 MQTT Server.....	64
7.1.2 MQTT Client	64
Appendix A. Mandatory normative statements.....	65
Appendix B. Revision history.....	72

1 Introduction

This specification is split into seven chapters:

- Introduction and concepts
- Control Packet format
- The specific details of each Control Packet type
- Operational behavior of the Client and Server
- Security considerations
- Using WebSocket as a network transport
- Conformance requirements for this version of the specification

1.1 Terminology

The key words "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].

Network Connection:

A construct provided by the underlying transport protocol that is being used by MQTT.

- It connects the Client to the Server,
- It provides the means to send an ordered, lossless, stream of bytes in both directions.

For examples see section 4.2.

Client:

A program or device that uses MQTT. A Client always establishes the Network Connection to the Server.

It can

- Publish Application Messages that other Clients might be interested in.
- Subscribe to request Application Messages that it is interested in receiving.
- Unsubscribe to remove a request for Application Messages.
- Disconnect from the Server.

Server:

Accepts connections from Clients. It is the intermediary between a Client publishing Application Messages and the Clients which have made Subscriptions.

Application Message:

The data carried by the MQTT protocol across the network for the application. When Application Messages are transported by MQTT they have an associated Quality of Service and a Topic Name.

32 **Topic Name:**

33 The label attached to an Application Message which is matched against the Subscriptions known to the
34 Server. The Server sends a copy of the Application Message to each Client that has a matching
35 Subscription.

36 **Topic Filter:**

37 An expression contained in a Subscription, to indicate an interest in one or more topics. A Topic Filter
38 may include wildcard characters.

39 **Subscription:**

40 A Subscription comprises a Topic Filter and its maximum QoS. A Subscription is associated with a single
41 Session. A Session can contain more than one Subscription. Each Subscription within a session MUST
42 have a different Topic Filter [MQTT-1.1.0-1].

43 **Session:**

44 A stateful interaction between a Client and a Server. Some Sessions only last as long as the Network
45 Connection, others span multiple Network Connections.

46 **MQTT Control Packet:**

47 A packet of information that flows across the Network Connection. The MQTT specification defines 14
48 types of Control Packet, one of which (the PUBLISH packet) is used to convey Application Messages.

49 **1.2 Normative references**

50 **[RFC793]**

51 *Postel, J. Transmission Control Protocol. STD 7, IETF RFC 793, September 1981.*

52 <http://www.ietf.org/rfc/rfc793.txt>

53

54 **[RFC2119]**

55 *S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. IETF RFC 2119, March 1997.*

56 <http://www.ietf.org/rfc/rfc2119.txt>

57

58 **[RFC3629]**

59 *F. Yergeau. UTF-8, a transformation format of ISO 10646 IETF RFC 3629, November 2003.*

60 <http://www.ietf.org/rfc/rfc3629.txt>

61

62 **[Unicode63]**

63 *Unicode 6.3.0 Specification*

64 <http://www.unicode.org/versions/Unicode6.3.0/>

65

66 **[RFC5246]**

67 *T. Dierks. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008*

68 <http://tools.ietf.org/html/rfc5246>

69

70 **[RFC6455]**
71 *I Fette. The WebSocket Protocol, IETF RFC 6455, December 2011*
72 <http://tools.ietf.org/html/rfc6455>

73
74 **[AES]**
75 *Advanced Encryption Standard (AES) (FIPS PUB 197).*
76 <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
77

78 **[DES]**
79 *Data Encryption Standard (DES).*
80 <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
81

82 **[PCIDSS]**
83 *PCI SSC Data Security Standards*
84 https://www.pcisecuritystandards.org/security_standards/
85

86 **[SARBANES]**
87 *Sarbanes-Oxley Act of 2002. Corporate responsibility.*
88
89 <http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm>
90

91 **[USEUSAFEHARB]**
92 *U.S.-EU Safe Harbor*
93 http://export.gov/safeharbor/eu/eg_main_018365.asp

94 **1.3 Non normative references**

95
96 **[MQTTV31]**
97 *MQTT V3.1 Protocol Specification.*
98 [http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-](http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html)
99 [v3r1.html](http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html)<http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html>.

100
101 **[RFC1928]**
102 *M Leech. SOCKS Protocol Version 5, March 1996.*
103 <http://www.ietf.org/rfc/rfc1928.txt>
104

105 **[RFC4511]**
106 *J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol, June 2006.*
107 <http://tools.ietf.org/html/rfc4511>
108

109 **[RFC6749]**
110 *D Hardt The OAuth 2.0 Authorization Framework, October 2012*
111 <http://tools.ietf.org/html/rfc6749>
112
113 **[RFC3546]**
114 *S. Blake-Wilson Transport Layer Security (TLS) Extensions, June 2003.*
115 <http://tools.ietf.org/html/rfc3546>
116
117 **[RFC5077]**
118 *J. Salowey Transport Layer Security (TLS) Session Resumption without Server-Side State, January 2008.*
119 <http://tools.ietf.org/html/rfc5077>
120
121 **[RFC6960]**
122 *S. Santesson X.509 Internet Public Key Infrastructure online Certificate Status Protocol – OCSP, June*
123 *2013.*
124 <http://tools.ietf.org/html/rfc6960>
125
126 **[IEEE 802.1AR]**
127 *IEEE Standard for Local and metropolitan area networks - Secure Device Identity*
128 <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>
129
130 **[NISTCSF]**
131 *Improving Critical Infrastructure Cybersecurity Executive Order 13636*
132 <http://www.nist.gov/itl/upload/preliminary-cybersecurity-framework.pdf>
133
134 **[NIST7628]**
135 *NISTIR 7628 Guidelines for Smart Grid Cyber Security*
136 http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf
137
138 **[FIPS1402]**
139 *Federal Information Processing Standards (FIPS-140-2)*
140 <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>
141
142 **[PCIDSS]**
143 *PCI-DSS Payment Card Industry Data Security Standard*
144 https://www.pcisecuritystandards.org/security_standards/
145
146 **[NSAB]**
147 *NSA Suite B Cryptography*
148 http://www.nsa.gov/ia/programs/suiteb_cryptography/
149
150 **[RFC6960]**
151 *S. Santesson X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*
152 <http://tools.ietf.org/html/rfc6960>
153

154 **[RFC5280]**
155 D Cooper Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
156 <http://tools.ietf.org/html/rfc5280>

157

158 **[ISO29192]**
159 Information technology -- Security techniques -- Lightweight cryptography -- Part 1: General
160 http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56425

161 **Acknowledgements**

162

- 163 • Sanjay Aiyagari (VMware, Inc.)
- 164 • Ben Bakowski (IBM)
- 165 • Andrew Banks (IBM)
- 166 • Arthur Barr (IBM)
- 167 • William Bathurst (Machine-to-Machine Intelligence (M2MI) Corporation)
- 168 • Ken Borgendale (IBM)
- 169 • Geoff Brown (Machine-to-Machine Intelligence (M2MI) Corporation)
- 170 • James Butler (Cimetrics Inc.)
- 171 • Marco Carrer (Eurotech S.p.A.)
- 172 • Raphael Cohn (Individual)
- 173 • Sarah Cooper (Machine-to-Machine Intelligence (M2MI) Corporation)
- 174 • Richard Coppen (IBM)
- 175 • AJ Dalola (Telit Communications S.p.A.)
- 176 • Mark Darbyshire (TIBCO Software Inc.)
- 177 • Scott deDeugd (IBM)
- 178 • Paul Duffy (Cisco Systems)
- 179 • John Fallows (Kaazing)
- 180 • Pradeep Fernando (WSO2)
- 181 • Paul Fremantle (WSO2)
- 182 • Thomas Glover (Cognizant Technology Solutions)
- 183 • Rahul Gupta (IBM)
- 184 • Steve Huston (Individual)
- 185 • Wes Johnson (Eurotech S.p.A.)
- 186 • Christopher Kelley (Cisco Systems)
- 187 • James Kirkland (Red Hat)
- 188 • Alex Kritikos (Software AG, Inc.)
- 189 • Louis-P. Lamoureux (Machine-to-Machine Intelligence (M2MI) Corporation)
- 190 • David Locke (IBM)
- 191 • Shawn McAllister (Solace Systems)
- 192 • Manu Namboodiri (Machine-to-Machine Intelligence (M2MI) Corporation)
- 193 • Peter Niblett (IBM)
- 194 • Arlen Nipper (Individual)

- 195 • Julien Niset (Machine-to-Machine Intelligence (M2MI) Corporation)
- 196 • Mark Nixon (Emerson Process Management)
- 197 • Nicholas O'Leary (IBM)
- 198 • Dominik Obermaier (dc-square GmbH)
- 199 • Pavan Reddy (Cisco Systems)
- 200 • Andrew Schofield (IBM)
- 201 • Wadih Shaib (BlackBerry)
- 202 • Ian Skerrett (Eclipse Foundation)
- 203 • Joe Speed (IBM)
- 204 • Allan Stockdill-Mander (IBM)
- 205 • Gary Stuebing (Cisco Systems)
- 206 • Steve Upton (IBM)
- 207 • T. Wyatt (Individual)
- 208 • SHAWN XIE (Machine-to-Machine Intelligence (M2MI) Corporation)
- 209 • Dominik Zajac (dc-square GmbH)

210

Secretary:

Geoff Brown (geoff.brown@m2mi.com), M2MI

1.4 Data representations

1.4.1 Bits

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

1.4.1.1 Integer data values

Integer data values are 16 bits 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).

1.4.1.2 UTF-8 encoded strings

Many of the fields in the Control Packets are encoded as UTF-8 strings. UTF-8 [\[RFC3629\]](#) is an efficient encoding of Unicode [\[Unicode63\]](#) characters that optimizes the encoding of ASCII characters in support of text-based communications.

225

Each of these strings is prefixed with a two byte length field that gives the number of bytes in the UTF-8 encoded string itself, as shown in table below. Consequently there is a limit on the size of a string that can be passed in one of these UTF-8 encoded string components; you cannot use a string that would encode to more than 65535 bytes.

Unless stated otherwise all UTF-8 encoded strings can have any length in the range 0 to 65535 bytes

231

Bit	7	6	5	4	3	2	1	0
byte 1	String byte length MSB							

byte 2	String byte length LSB
byte 3	UTF-8 Encoded Character Data, if length > 0.

232
 233 The encoded data MUST be well-formed UTF-8 as defined by the Unicode spec [Unicode63] and
 234 restated in RFC 3629 [RFC 3629]. In particular the encoded data MUST NOT include encodings of code
 235 points between U+D800 and U+DFFF. If a receiver (Server or Client) receives a control packet containing
 236 ill-formed UTF-8 it MUST close the network connection. [MQTT-1.4.0-1].
 237

238 The UTF-8 encoded string MUST NOT include an encoding of the null character U+0000. If a receiver
 239 (Server or Client) receives a control packet containing U+0000 it MUST close the network
 240 connection. [MQTT-1.4.0-2]

241 The data SHOULD NOT include encodings of the Unicode [Unicode63] code points listed below. If a
 242 receiver (Server or Client) receives a control packet containing any of them it MAY close the network
 243 connection.

- 244
- 245 U+0001..U+001F control characters
- 246 U+007F..U+009F control characters

247 Code points defined in the Unicode specification [Unicode63] to be non-characters (for example
 248 U+0FFFF)

249
 250 The UTF-8 encoded sequence 0xEF 0xBB 0xBF is always to be interpreted to mean U+FEFF ("ZERO
 251 WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped
 252 off by a packet receiver. [MQTT-1.4.0-3]

253
 254 **Non normative example.**

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

258

Bit	7	6	5	4	3	2	1	0
byte 1	Message Length MSB (0x00)							
	0	0	0	0	0	0	0	0
byte 2	Message Length LSB (0x05)							
	0	0	0	0	0	1	0	1
byte 3	'A' (0x41)							
	0	1	0	0	0	0	0	1
byte 4	(0xF0)							
	1	1	1	1	0	0	0	0
byte 5	(0xAA)							
	1	0	1	0	1	0	1	0
byte 6	(0x9B)							
	1	0	0	1	1	0	1	1

byte 7	(0x94)							
	1	0	0	1	0	1	0	0

259

2 MQTT Control Packet format

260

261 The MQTT protocol works by exchanging a series of MQTT Control Packets in a defined way. This
262 section describes the format of these packets. An MQTT Control Packet consists of up to three parts,
263 always in the following order:

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

264

265 Unless stated otherwise, if either the Server or Client receives a Control Packet which does not meet this
266 specification, it MUST close the Network Connection [MQTT-2.0.0-1].

267

2.1 Fixed header

268

Each MQTT Control Packet contains a fixed header. The table below shows the fixed header format.

269

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

270

271

2.1.1 MQTT Control Packet types

272

273 **Position:** byte 1, bits 7-4.

274

Represented as a 4-bit unsigned value, the values are shown in the table below.

275

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment

PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

276

277 **2.1.2 Flags**

278 The remaining bits [3-0] of byte 1 in the fixed header contain flags specific to each MQTT Control Packet
 279 type as detailed in the table below. Where a bit is marked as “Reserved”, it MUST be set as shown in the
 280 table and is reserved for future use. **If invalid flags are received, the receiver MUST close the Network**
 281 **Connection [MQTT-2.1.2-1].**

282

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	Dup	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0

UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

283

284 Dup = Duplicate delivery of Control Packet

285 QoS = Quality of Service

286 RETAIN = Retain flag

287 2.1.2.1 Dup

288 **Position:** byte 1, bit 3.

289 If Dup is 0 then the flow is the first occasion that the Client or Server has attempted to send the MQTT
 290 PUBLISH Packet. If Dup is 1 then this indicates that the flow might be re-delivery of an earlier packet.
 291 [MQTT-2.1.2-2].

292 The Dup flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet
 293 [MQTT-2.1.2-3].

294 The Dup flag MUST be 0 for all QoS 0 messages. [MQTT-2.1.2-4].

295 The value of the Dup flag from an incoming PUBLISH packet is not propagated when the PUBLISH
 296 Packet is sent to subscribers by the Server. The Dup flag in the outgoing PUBLISH packet MUST BE set
 297 independently to the incoming PUBLISH packet. [MQTT-2.1.2-5].

298

299 **Non Normative comment.**

300 The recipient of a Control Packet that contains the Dup flag set to 1 cannot assume that it has seen an
 301 earlier copy of this packet.

302

303 **Non Normative comment.**

304 It is important to note that the Dup flag refers to the Control Packet itself and not to the Application
 305 Message that it contains. When using QoS 1, it is possible for a Client to receive a PUBLISH Packet with
 306 DUP set to 0 that contains a repetition of an Application Message that it received earlier, but with a
 307 different Packet Identifier. See section 2.3.1 Packet Identifier.

308 2.1.2.2 QoS

309 **Position:** byte 1, bit 2-1.

310 This field indicates the level of assurance for delivery of an Application Message. The QoS levels are
 311 shown in the table below.

312

QoS value	bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
3	1	1	Reserved (MUST NOT be used)

313 **2.1.2.3 RETAIN**

314 **Position:** byte 1, bit 0.

315

316 This flag is only used on the PUBLISH Packet.

317

318 If the retain flag is set to 1, in a PUBLISH Packet sent by a Client to a Server, the Server MUST store the
319 application message and its QoS, so that it can be delivered to future subscribers whose subscriptions
320 match its topic name. [MQTT-2.1.2-6] When a new subscription is established, the last retained message,
321 if any, on each matching topic name MUST be sent to the subscriber. [MQTT-2.1.2-7] If the Server
322 receives a QoS 0 message with the RETAIN flag set to 1 it MUST discard any message previously
323 retained for that topic. It SHOULD store the new QoS 0 message as the new retained message for that
324 topic, but MAY discard it at any time. If this happens there will be no retained message for that topic.
325 [MQTT-2.1.2-8] See Section 4.1 storing state.

326

327 When sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag to 1 if a message is
328 sent as a result of a new subscription being made by a Client [MQTT-2.1.2-9]. It MUST set the RETAIN
329 flag to 0 when a PUBLISH Packet is sent to a Client because it matches an established subscription
330 regardless of how the flag was set in the message it received [MQTT-2.1.2-10].

331

332 A PUBLISH Packet with a retain flag set to 1 and a payload containing zero bytes will be processed as
333 normal by the Server and sent to Clients with a subscription matching the topic name. Additionally any
334 existing retained message with the same topic name MUST be removed and any future subscribers for
335 the topic will not receive a retained message. [MQTT-2.1.2-11] "As normal" means that the Retain flag is
336 not set in the message received by existing Clients.

337

338 If the RETAIN flag is 0, in a PUBLISH Packet sent by a Client to a Server, the Server MUST NOT store
339 the message and MUST NOT remove or replace any existing retained message. [MQTT-2.1.2-12]

340

341 **Non normative comment.**

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

344

345 **2.2 Remaining Length**

346 **Position:** starts at byte 2.

347

348 The Remaining Length is the number of bytes remaining within the current packet, including data in the
349 variable header and the payload. The Remaining Length does not include the bytes used to encode the
350 Remaining Length.

351

352 The Remaining Length is encoded using a variable length encoding scheme which uses a single byte for
353 values up to 127. Larger values are handled as follows. The least significant seven bits of each byte
354 encode the data, and the most significant bit is used to indicate that there are following bytes in the
355 representation. Thus each byte encodes 128 values and a "continuation bit". The maximum number of
356 bytes in the Remaining Length field is four.

357

358 **Non normative comment.**

359 For example, the number 64 decimal is encoded as a single byte, decimal value 64, hexadecimal 0x40.
360 The number 321 decimal (= 65 + 2*128) is encoded as two bytes, least significant first. The first byte
361 65+128 = 193. Note that the top bit is set to indicate at least one following byte. The second byte is 2.

362

363 **Non normative comment.**

364 This allows applications to send Control Packets of size up to 268,435,455 (256 MB). The representation
365 of this number on the wire is: 0xFF, 0xFF, 0xFF, 0x7F.

366 The table below shows the Remaining Length values represented by increasing numbers of bytes.

367

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

368

369 **Non normative comment.**

370 The algorithm for encoding a non negative integer (X) into the variable length encoding scheme is as
371 follows:

```
372     do
373         encodedByte = X MOD 128
374         X = X DIV 128
375         // if there are more data to encode, set the top bit of this byte
376         if ( X > 0 )
377             encodedByte = encodedByte OR 128
378         endif
379         'output' encodedByte
380     while ( X > 0 )
```

381

382 Where MOD is the modulo operator (% in C), DIV is integer division (/ in C), and OR is bit-wise or (| in C).

383

384 **Non normative comment.**

385 The algorithm for decoding the Remaining Length field is as follows:

386

```
387     multiplier = 1
388     value = 0
389     do
390         encodedByte = 'next byte from stream'
391         value += (encodedByte AND 127) * multiplier
392         multiplier *= 128
393         if (multiplier > 128*128*128)
394             throw Error(Malformed Remaining Length)
```

395 while ((encodedByte AND 128) != 0)

396

397 where AND is the bit-wise and operator (& in C).

398

399 When this algorithm terminates, value contains the Remaining Length value.

400 2.3 Variable header

401 Some types of MQTT Control Packets contain a variable header component. It resides between the fixed
402 header and the payload. The content of the variable header varies depending on the Packet type,
403 however one field - the Packet Identifier - is common to several packet types.

404 2.3.1 Packet Identifier

Bit	7	6	5	4	3	2	1	0
	Packet Identifier MSB							
	Packet Identifier LSB							

405

406 The variable header component of many of the Control Packet types includes a 2 byte Packet Identifier
407 field. These Control Packets are PUBLISH (where QoS > 0), PUBACK, PUBREC, PUBREL, PUBCOMP,
408 SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

409

410 SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) Control Packets MUST contain a
411 non-zero 16-bit Packet Identifier [MQTT-2.3.1-1]. Each time a Client sends a new packet of one of these
412 types it MUST assign it a currently unused Packet Identifier [MQTT-2.3.1-2]. If a Client resends a
413 particular Control Packet, then it MUST use the same Packet Identifier in subsequent resends of that
414 packet. The Packet Identifier becomes available for reuse after the Client has processed the
415 corresponding acknowledgement packet. In the case of a QoS 1 PUBLISH this is the corresponding
416 PUBACK; in the case of QoS 2 it is PUBCOMP. For SUBSCRIBE or UNSUBSCRIBE it is the
417 corresponding SUBACK or UNSUBACK. [MQTT-2.3.1-3]. The same conditions apply to a Server when it
418 sends a PUBLISH with QoS > 0 [MQTT-2.3.1-4].

419

420 A PUBLISH Packet MUST NOT contain a Packet Identifier if its QoS value is set to 0 [MQTT-2.3.1-5].

421

422 A PUBACK, PUBREC, PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH
423 Packet that initiated the flow [MQTT-2.3.1-6]. Similarly SUBACK and UNSUBACK MUST contain the
424 Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE Packet
425 respectively [MQTT-2.3.1-7].

426

427

428 Control Packets that contain a Packet Identifier

Control Packet	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)
PUBACK	YES

PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES
SUBACK	YES
UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO

429

430

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

433

434 **Non Normative comment.**

435

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

439

```

Client                               Server
440 PUBLISH Packet Identifier=0x1234--->
441 <--PUBLISH Packet Identifier=0x1234
442 PUBACK Packet Identifier=0x1234--->
443 <--PUBACK Packet Identifier=0x1234

```

444

445 2.3.2 Payload

446 Some MQTT Control Packets contain a payload as the final part of the packet, as described in section 3.
 447 In the case of the PUBLISH packet this is the Application Message.

448 3 MQTT Control Packets

449 3.1 CONNECT – Client requests a connection to a Server

450

451 After a Network connection is established by a Client to a Server, the first flow from the Client to the
452 Server MUST be a CONNECT Packet [MQTT-3.1.0-1].

453 A Client can only flow the CONNECT Packet once over a Network Connection. The Server MUST
454 process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client
455 [MQTT-3.1.0-2].

456

457 The payload contains one or more encoded fields. They specify a unique Client identifier for the Client, a
458 Will topic, Will message, User Name and Password. All but the Client identifier are optional and their
459 presence is determined based on flags in the variable header.

460 3.1.1 Fixed header

461 The fixed header format is shown in the table below.

462

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
Byte 2...	Remaining Length							

463

464 Remaining Length field

465 Remaining Length is the length of the variable header (10 bytes) plus the length of the Payload. It is
466 encoded in the manner described in section 2.2

467 3.1.2 Variable header

468 The variable header for the CONNECT Packet consists of four fields in the following order: Protocol
469 Name, Protocol Level, Connect Flags, and Keep Alive.

470 3.1.2.1 Protocol Name

471

	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'T'	0	1	0	1	0	1	0	0

byte 6	'T'	0	1	0	1	0	1	0	0
--------	-----	---	---	---	---	---	---	---	---

472

473 The Protocol Name, is a UTF-8 encoded string that represents the protocol name “MQTT”, capitalized as
 474 shown. The string, its offset and length will not be changed by future versions of the MQTT specification.

475

476 If the protocol name is incorrect the Server MAY disconnect the Client, or it MAY continue processing the
 477 CONNECT packet in accordance with some other specification. In the latter case, the Server MUST NOT
 478 continue to process the CONNECT packet in line with this specification [MQTT-3.1.2-1].

479

480

481 **Non normative comment**

482 Packet inspectors, such as firewalls, could use the Protocol Name to identify MQTT traffic.

483 **3.1.2.2 Protocol Level**

	Description	7	6	5	4	3	2	1	0
Protocol Level									
byte 7	Level(4)	0	0	0	0	0	1	0	0

484

485 The 8 bit unsigned value that represents the revision level of the protocol used by the Client. The value of
 486 the Protocol Level field for the version 3.1.1 of the protocol is 4 (0x04). The Server MUST respond to the
 487 CONNECT Packet with a CONNACK return code 0x01 (unacceptable protocol level) and then disconnect
 488 the Client if the Protocol Level is not supported by the Server [MQTT-3.1.2-2].

489

490 **3.1.2.3 Connect Flags**

491 The Connect Flags byte contains a number of parameters specifying the behavior of the MQTT
 492 connection. It also indicates the presence or absence of fields in the payload.

493

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

494 The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and
 495 disconnect the Client if it is not zero. [MQTT-3.1.2-3]

496

497 **3.1.2.4 Clean Session**

498 **Position:** bit 1 of the Connect Flags byte.

499

500 If set to 0, the Server resumes communications with the Client based on state from the current Session
 501 (as identified by the Client identifier). If there is no Session associated with the Client identifier the Server
 502 creates a new Session. The Client and Server MUST store the Session after the Client and Server are
 503 disconnected [MQTT-3.1.2-4]. After disconnection, the Server MUST store further QoS 1 and QoS 2

504 messages that match any subscriptions that the client had at the time of disconnection as part of the
505 Session state [MQTT-3.1.2-5]. It MAY also store QoS 0 messages that meet the same criteria.

506

507 If set to 1, the Client and Server MUST discard any previous Session and start a new one. This Session
508 lasts as long as the Network Connection. State data associated with this session MUST NOT be reused
509 in any subsequent Session [MQTT-3.1.2-6].

510

511 The Session state in the Client consists of:

- 512 • QoS 1 and QoS 2 messages for which transmission to the Server is incomplete.
- 513 • The Client MAY store QoS 0 messages for later transmission.

514

515 The Session state in the Server consists of:

- 516 • The Client's subscriptions.
- 517 • All QoS 1 and QoS 2 messages for which transmission to the Client is incomplete or where
518 transmission to the Client has not yet been started.
- 519 • The Server MAY store QoS 0 messages for which transmission to the Client has not yet been started.

520

521 Retained publications do not form part of the Session state in the Server, they MUST NOT be deleted
522 when the Session ends [MQTT-3.1.2.7].

523

524 See section 4.1 for details and limitations of stored state.

525

526 When Clean Session is set to 1 the Client and Server need not process the deletion of state atomically.

527

528 **Non Normative comment.**

529 Consequently, in the event of a failure to connect the Client should repeat its attempts to connect with
530 Clean Session set to 1, until it connects successfully.

531

532 **Non Normative comment.**

533 Typically, a Client will always connect using CleanSession 0 or CleanSession 1 and not swap between
534 the two values. The choice will depend on the application. A Client using CleanSession 1 will not receive
535 old publications and has to subscribe afresh to any topics that it is interested in each time it connects. A
536 Client using CleanSession 0 will receive all QoS 1 or QoS 2 messages that were published whilst it was
537 disconnected. Hence, to ensure that you do not lose messages while disconnected, use QoS 1 or QoS 2
538 with CleanSession 0.

539 **Non Normative comment.**

540 When a Client connects with cleanSession = 0 it is requesting that the Server maintain its MQTT session
541 state after it disconnects. Clients should only connect with cleanSession = 0 if they intend to reconnect to
542 the Server at some later point in time. When a Client has determined that it has no further use for the
543 session it should do a final connect with cleanSession = 1 and then disconnect.

544 **3.1.2.5 Will Flag**

545 **Position:** bit 2 of the Connect Flags.

546

547 If the Will Flag is set to 1 this indicates that a Will Message MUST be published by the Server when the
548 Server detects that the Client is disconnected for any reason other than the Client flowing a
549 DISCONNECT Packet [MQTT-3.1.2-8]. This includes, but is not limited to, the following situations:

- 550
- An I/O error or network failure detected by the Server.
- 551
- The Client fails to communicate within the Keep Alive time.
- 552
- The Client closes the Network Connection without first sending a DISCONNECT Packet.
- 553
- The Server closes the Network Connection because of a protocol error.
- 554

555 If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the
556 Server, and the Will Topic and Will Message fields MUST be present in the payload [MQTT-3.1.2-9].

557

558 The will message MUST be removed from the stored Session state in the Server once it has been
559 published or the Server has received a DISCONNECT packet from the Client. If the Will Flag is set to 0,
560 no will message is published. [MQTT-3.1.2-10]

561 3.1.2.6 Will QoS

562 **Position:** bits 4 and 3 of the Connect Flags.

563

564 These two bits specify the QoS level to be used when publishing the Will Message.

565

566 If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00) [MQTT-3.1.2-11].

567 If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). It MUST NOT be 3
568 (0x03). [MQTT-3.1.2-12].

569 3.1.2.7 Will Retain

570 **Position:** bit 5 of the Connect Flags.

571

572 If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0 [MQTT-3.1.2-13].

573

574 If the Will Flag is set to 1:

575 • If Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained publication
576 [MQTT-3.1.2-14].

577 • If Will Retain is set to 1, the Server MUST publish the Will Message as a retained publication
578 [MQTT-3.1.2-15].

579

580 3.1.2.8 User Name Flag

581 **Position:** bit 7 of the Connect Flags.

582

583 If the User Name Flag is set to 0, a user name MUST NOT be present in the payload [MQTT-3.1.2-16].

584 If the User Name Flag is set to 1, a user name MUST be present in the payload [MQTT-3.1.2-17].

585

586 3.1.2.9 Password Flag

587 **Position:** bit 6 of the Connect Flags byte.

588

589 If the Password Flag is set to 0, a password MUST NOT be present in the payload [MQTT-3.1.2-18].

590 If the Password Flag is set to 1, a password MUST be present in the payload [MQTT-3.1.2-19].

591 If the User Name Flag is set to 0, the Password Flag MUST be set to 0 [MQTT-3.1.2-20].

592

593 3.1.2.10 Keep Alive

Bit	7	6	5	4	3	2	1	0	
byte 9	Keep Alive MSB								
byte 10	Keep Alive LSB								

594

595 The Keep Alive is a time interval measured in seconds. Expressed as a 16-bit word, it is the maximum
596 time interval that is permitted to elapse between two successive Control Packets sent by the Client.

597

598 It is the responsibility of the Client to ensure that the interval between Control Packets being sent does
599 not exceed the Keep Alive value. In the absence of sending any other Control Packets, the Client MUST
600 send a PINGREQ Packet [MQTT-3.1.2-21].

601 The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and use the PINGRESP
602 to determine that the network and the Server are working.

603

604 If the Server does not receive a Control Packet from the Client within one and a half times the Keep Alive
605 time period, it MUST disconnect the Network Connection to the Client as if the network had failed.
606 [MQTT-3.1.2-22]

607

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

610

611 A Keep Alive value of zero (0) has the effect of turning off the keep alive mechanism. This means that, in
612 this case, the Server is NOT REQUIRED to disconnect the Client on the grounds of inactivity.
613 Note that a Server MAY choose to disconnect a Client that it determines to be inactive or non-responsive
614 at any time, regardless of the Keep Alive value provided by that Client.

615

616 Non normative comment.

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

619

620 3.1.2.11 Variable header example, Non normative

621

	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1

byte 5	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0
Protocol Level									
	Description	7	6	5	4	3	2	1	0
byte 7	Level (4)	0	0	0	0	0	1	0	0
Connect Flags									
byte 8	User Name Flag (1)								
	Password Flag (1)								
	Will Retain (0)								
	Will QoS (01)	1	1	0	0	1	1	1	0
	Will Flag (1)								
	Clean Session (1)								
	Reserved (0)								
Keep Alive									
byte 9	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
byte 10	Keep Alive LSB (10)	0	0	0	0	1	0	1	0

622

623 3.1.3 Payload

624 The payload of the CONNECT Packet contains one or more length-prefixed fields, whose presence is
625 determined by the flags in the variable header. These fields, if present, MUST appear in the order Client
626 Identifier, Will Topic, Will Message, User Name, Password. [MQTT-3.1.3-1]

627

628 3.1.3.1 Client Identifier

629 The Client Identifier (ClientId) identifies the Client to the Server. Each Client connecting to the Server has
630 a unique ClientId. The ClientId MUST be used by Clients and by Servers to identify state that they hold
631 relating to this MQTT Session between the Client and the Server. [MQTT-3.1.3-2]

632

633 The Client Identifier (ClientId) MUST be present and MUST be the first field in the payload. [MQTT-3.1.3-
634 3]

635

636 The ClientId MUST comprise only Unicode [Unicode63] characters, and the length of the UTF-8 encoding
637 MUST be at least zero bytes and no more than 65535 bytes. [MQTT-3.1.3-4]

638

639 The Server MAY restrict the ClientId it allows in terms of their lengths and the characters they
640 contain. The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length,
641 and that contain only the characters

642 "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ". [MQTT-3.1.3-5]

643

644

645

646 A Server MAY allow a Client to supply a ClientId that has a length of zero bytes. However if it does so the
647 Server MUST treat this as a special case and assign a unique ClientId to that Client. It MUST then
648 process the CONNECT packet as if the Client had provided that unique ClientId. [MQTT-3.1.3-6]

649

650 If the Client supplies a zero-byte ClientId, the Client MUST also set Clean Session to 1. [MQTT-3.1.3-7]

651

652 If the Client supplies a zero-byte ClientId with Clean Session set to 0, the Server MUST respond to the
653 CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network
654 Connection. [MQTT-3.1.3-8]

655

656 If the Server rejects the ClientId it MUST respond to the CONNECT Packet with a CONNACK return code
657 0x02 (Identifier rejected) and then close the Network Connection. [MQTT-3.1.3-9]

658

659 **Non Normative comment.**

660

661 A Client implementation may provide a convenience method to generate a random ClientId. Use of such
662 a method should be actively discouraged when the Clean Session flag is set to 0.

663 3.1.3.2 Will Topic

664 If the Will Flag is set to 1, the Will Topic is the next field in the payload. The Will Topic is a UTF-8
665 encoded string.

666 3.1.3.3 Will Message

667 If the Will Flag is set to 1 the Will Message is the next field in the payload. The Will Message defines the
668 Application Message that is to be published to the Will Topic if the Client is disconnected for any reason
669 other than the Client sending a DISCONNECT Packet. This field consists of a 2-byte length followed by
670 the payload for the Will Message expressed as a sequence of zero or more bytes. The length gives the
671 number of bytes in the data that follows and does not include the 2 bytes taken up by the length itself.

672

673 When the Will Message is published to the Will Topic its payload consists only of the data portion of this
674 field, not the first two length bytes.

675 3.1.3.4 User Name

676 If the User Name Flag is set to 1, this is the next field in the payload. User Name is a UTF-8 encoded
677 string and can be used by the Server for authentication and authorization.

678 3.1.3.5 Password

679 If the Password Flag is set to 1, this is the next field in the payload. The Password field contains 0 to
680 65535 bytes of binary data prefixed with a 2 byte length field which indicates the number of bytes used by
681 the binary data (it does not include the two bytes taken up by the length field itself).

682

Bit	7	6	5	4	3	2	1	0
byte 1	Data length MSB							
byte 2	Data length LSB							

byte 3	Data, if length > 0.
-------------	----------------------

683

684 3.1.4 Response

685 Note that a Server MAY support multiple protocols (including earlier versions of this protocol) on the same
686 TCP port or other network endpoint. If the Server determines that the protocol is MQTT 3.1.1 then it
687 MUST validate the connection attempt as follows.

688

- 689 1. If the Server does not receive a CONNECT Packet within a reasonable amount of time after the
690 Network Connection is established, the Server SHOULD close the connection.
691
- 692 2. The Server MUST validate that the CONNECT Packet conforms to section 3.1 and close the
693 Network Connection without sending a CONNACK if it does not conform [MQTT-3.1.4-1].
694
- 695 3. The Server MAY check that the contents of the CONNECT Packet meet any further restrictions
696 and MAY perform authentication and authorization checks. If any of these checks fail, it SHOULD
697 send an appropriate CONNACK response with a non zero return code as described in section 3.2
698 and it MUST close the Network Connection.

699

700 If validation is successful the Server MUST perform the following steps.

701

- 702 1. If the ClientId represents a Client already connected to the Server then the Server MUST
703 disconnect the existing Client [MQTT-3.1.4-2].
704
- 705 2. Processing of Clean Session is performed as described in section 3.1.2.4.
706
- 707 3. The Server acknowledges the CONNECT Packet with a CONNACK Packet containing a zero
708 return code.
709
- 710 4. Start message delivery and keep alive monitoring.

711

712 Clients are allowed to send further Control Packets immediately after sending a CONNECT Packet;
713 Clients need not wait for a CONNACK Packet to arrive from the Server. If the Server rejects the
714 CONNECT, it MUST NOT process any data sent by the Client after the CONNECT Packet [MQTT-3.1.4-
715 3].
716

717 Non Normative comment.

718

719 Clients typically wait for a CONNACK Packet, However, if the Client exploits its freedom to send Control
720 Packets before it receives a CONNACK, it might simplify the Client implementation as it does not have to
721 police the connected state. The Client accepts that any data that it sends before it receives a CONNACK
722 packet from the Server will not be processed if the Server rejects the connection.

723

724 3.2 CONNACK – Acknowledge connection request

725

726 The CONNACK Packet is the packet sent by the Server in response to a CONNECT Packet received
 727 from a Client. **The first packet sent from the Server to the Client MUST be a CONNACK Packet [MQTT-**
 728 **3.2.0-1].**

729
 730 If the Client does not receive a CONNACK Packet from the Server within a reasonable amount of time,
 731 the Client SHOULD close the Network Connection. A "reasonable" amount of time depends on the type of
 732 application and the communications infrastructure.

733

734 **3.2.1 Fixed header**

735

736 The fixed header format is shown in the table below.

737

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet Type (2)				Reserved			
	0	0	1	0	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

738

739 **Remaining Length field**

740 This is the length of the variable header. For the CONNACK Packet this has the value 2.

741 **3.2.2 Variable header**

742 The variable header format is shown in the table below.-

	Description	7	6	5	4	3	2	1	0
Reserved for future use									
byte 1		0	0	0	0	0	0	0	0
CONNECT Return code									
byte 2									

743

744 The values for the one byte unsigned CONNECT Return code field are shown in the table below. If a well
 745 formed CONNECT Packet is received by the Server, but the Server is unable to process it for some
 746 reason, then the Server SHOULD attempt to flow one of the following non-zero CONNACK return codes.
 747 **If a server sends a CONNACK packet containing a non-zero return code it MUST then close the Network**
 748 **Connection. [MQTT-3.2.2-1]**

Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not

		allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use

749

750 If none of these return codes are deemed applicable, then the Server MUST close the Network
751 Connection without flowing a CONNACK. [MQTT-3.2.2-2]

752 3.2.3 Payload

753 There is no payload in the CONNACK Packet.

754

755 3.3 PUBLISH – Publish message

756 A PUBLISH Control Packet is sent from a Client to a Server or from Server to a Client to transport an
757 Application Message.

758 3.3.1 Fixed header

759 The table below shows the fixed header format:

760

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)			Dup flag		QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

761

762 Dup flag

763 See Dup section 2.1.2.1 for details.

764

765 QoS level

766 See QoS section 2.1.2.2 for details.

767

768 RETAIN flag

769 See Retain section 2.1.2.3 for details.

770

771 Remaining Length field

772 This is the length of variable header plus the length of the payload.

773

774 **3.3.2 Variable header**

775 The variable header contains the following fields in the order below:

776 **3.3.2.1 Topic Name**

777 The Topic Name is always present as the first field in the variable header. The Topic Name identifies the
778 information channel to which payload data is published.

- 779
- 780 The Topic Name MUST be a UTF-8 encoded string [MQTT-3.3.2-1] as defined in section 1.4.1.2.
- 781 The Topic Name in the PUBLISH Packet MUST NOT contain wildcard characters. [MQTT-3.3.2-2]
- 782 The Topic Name sent to a subscribing Client MUST match the Subscription's Topic Filter. [MQTT-3.3.2-3]
- 783 However, since the Server is permitted to override the Topic Name, it might not be the same as the Topic
- 784 Name in the original PUBLISH Packet.

785 **3.3.2.2 Packet Identifier**

786 The Packet Identifier field is only present in PUBLISH Packets where the QoS level is 1 or 2. See Packet
787 Identifiers section 2.3.1 for more details.

788

789 **3.3.2.3 Variable header example Non Normative**

790

791 The table below illustrates an example of variable header for a PUBLISH Packet.

792

Field	Value
Topic Name	a/b
Packet Identifier	10

793

794 The format of the variable header in this case is shown in the table below.

795

	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Packet Identifier									
byte 6	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 7	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0

796

797

3.3.3 Payload

798 The Payload contains the Application Message that is being published. The content and format of the
799 data is application specific. The length of the payload can be calculated by subtracting the length of the
800 variable header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH
801 Packet to contain a zero length payload.

802

3.3.4 Response

804 The response to a PUBLISH Packet depends on the QoS level. The table below shows the expected
805 responses.

806

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

807

3.3.5 Actions

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

811

812 The Server uses PUBLISH Packets to send an Application Messages to those Clients which have
813 matching subscriptions.

814

815 When Clients make subscriptions with Topic Filters that include wildcards, it is possible for a Client's
816 subscriptions to overlap so that a published message might match multiple filters. In this case the Server
817 MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions
818 [MQTT-3.3.5-1]. In addition, the Server MAY deliver further copies of the message, one for each
819 additional matching subscription and respecting the subscription's QoS in each case.

820

821 The action of the recipient when it receives a PUBLISH Packet depends on the QoS level as described in
822 Section 4.3.

823

824 If a Server implementation does not authorize a PUBLISH to be performed by a Client; it has no way of
825 informing that Client. It MUST either make a positive acknowledgement, according to the normal QoS
826 rules, or close the Network Connection [MQTT-3.3.5-2].

3.4 PUBACK – Publish acknowledgement

828

829 A PUBACK Packet is the response to a PUBLISH Packet with QoS level 1.

3.4.1 Fixed header

831 The table below shows the format of the fixed header.

832

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (4)				Reserved			
	0	1	0	0	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

833

834 Remaining Length field

835 This is the length of the variable header. For the PUBACK Packet this has the value 2.

836 3.4.2 Variable header

837

838 Contains the Packet Identifier from the PUBLISH Packet that is being acknowledged. The table below
839 shows the format of the variable header.

840

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

841

842 3.4.3 Payload

843 There is no payload in the PUBACK Packet.

844 3.4.4 Actions

845 When the sender of a PUBLISH Packet receives a PUBACK Packet it discards the original message.

846 This is fully described in Section 4.3.

847

848 3.5 PUBREC – Publish received (QoS 2 publish received, part 1)

849

850 A PUBREC Packet is the response to a PUBLISH Packet with QoS 2. It is the second packet of the QoS
851 2 protocol flow.

852 3.5.1 Fixed header

853 The table below shows the format of the fixed header.

854

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (5)				Reserved			
	0	1	0	1	0	0	0	0
byte 2	Remaining Length (2)							

	0	0	0	0	0	0	1	0
--	---	---	---	---	---	---	---	---

855

856 **Remaining Length field**

857 This is the length of the variable header. For the PUBREC Packet this has the value 2.

858

859 **3.5.2 Variable header**

860

861 The variable header contains the Packet Identifier from the PUBLISH Packet that is being acknowledged.
862 The table below shows the format of the variable header.

863

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

864

865 **3.5.3 Payload**

866 There is no payload in the PUBREC Packet.

867

868 **3.5.4 Actions**

869 **When the sender of a PUBLISH Packet receives a PUBREC Packet, it MUST reply with a PUBREL**
870 **Packet [MQTT-3.5.4-1].**

871

872 This is fully described in Section 4.3.

873

874 **3.6 PUBREL – Publish release (QoS 2 publish received, part 2)**

875

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

877 **3.6.1 Fixed header**

878 The table below shows the format of the fixed header.

879

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (6)				Reserved			
	0	1	1	0	0	0	1	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

880

881 Bits 3,2,1 and 0 of the fixed header in the PUBREL Control Packet are reserved and MUST be set to
 882 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network
 883 Connection. [MQTT-3.6.1-1]

884

885 **Remaining Length field**

886 This is the length of the variable header. For the PUBREL Packet this has the value 2.

887 **3.6.2 Variable header**

888 The variable header contains the same Packet Identifier as the PUBREC Packet that is being
 889 acknowledged. The table below shows the format of the variable header.

890

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

891

892 **3.6.3 Payload**

893 There is no payload in the PUBREL Packet.

894 **3.6.4 Actions**

895 When the sender of a PUBREC Packet receives a PUBREL Packet it MUST reply with a PUBCOMP
 896 Packet [MQTT-3.6.4-1].

897

898 This is fully described in Section 4.3.

899 **3.7 PUBCOMP – Publish complete (QoS 2 publish received, part 3)**

900

901 The PUBCOMP Packet is the response to a PUBREL Packet. It is the fourth and final packet of the QoS
 902 2 protocol flow.

903

904 **3.7.1 Fixed header**

905 The table below shows the format of the fixed header.

906

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (7)				Reserved			
	0	1	1	1	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

907

908 **Remaining Length field**

909 This is the length of the variable header. For the PUBCOMP Packet this has the value 2.
910

911 3.7.2 Variable header

912 The variable header contains the same Packet Identifier as the PUBREL Packet that is being
913 acknowledged.
914

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

915

916 3.7.3 Payload

917 There is no payload in the PUBCOMP Packet.
918

919 3.7.4 Actions

920 When the sender of a PUBREL receives a PUBCOMP Packet it removes any remaining state associated
921 with the original PUBLISH Packet.
922 This is fully described in Section 4.3.
923

924 3.8 SUBSCRIBE - Subscribe to topics

925 The SUBSCRIBE Packet is sent from the Client to the Server to create one or more Subscriptions. Each
926 Subscription registers a Client's interest in one or more Topics. The Server sends PUBLISH Packets to
927 the Client in order to forward Application Messages that were published to Topics that match these
928 Subscriptions. The SUBSCRIBE Packet also specifies (for each Subscription) the maximum QoS with
929 which the Server can send publications to the Client.
930

931 3.8.1 Fixed header

932 The table below shows the format of the fixed header.
933

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
byte 2	Remaining Length							

934

935 Bits 3,2,1 and 0 of the fixed header of the SUBSCRIBE Control Packet are reserved and MUST be set to
936 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network
937 Connection [MQTT-3.8.1-1].
938

939 **Remaining Length field**

940 This is the length of variable header (2 bytes) plus the length of the payload.

941 **3.8.2 Variable header**

942 The variable header contains a Packet Identifier. See Section 2.3.1 for more details.

943

944 **3.8.2.1 Variable Header Non Normative example**

945 The table below shows an example of the variable header with a Packet Identifier of 10.

946

	Description	7	6	5	4	3	2	1	0
Packet Identifier									
byte 1	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 2	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0

947

948 **3.8.3 Payload**

949 The payload of a SUBSCRIBE Packet contains a list of Topic Filters indicating the Topics to which the
950 Client wants to subscribe. The Topic Filters are UTF-8 encoded strings, which MAY contain special
951 wildcard characters to represent a set of topics, see Section 4.7.1. Each filter is followed by a byte called
952 the Requested QoS. This gives the maximum QoS level at which the Server can send publications to the
953 Client.

954 The Payload of a SUBSCRIBE packet MUST contain at least one Topic Filter / QoS pair. A SUBSCRIBE
955 packet with no payload is a protocol violation [MQTT-3.8.3-1].

956

957 The requested maximum QoS field is encoded in the byte following each UTF-8 encoded topic name, and
958 these Topic Filter / QoS pairs are packed contiguously as shown in the table below.

959

Description	7	6	5	4	3	2	1	0
Topic Filter								
byte 1	Length MSB							
byte 2	Length LSB							
bytes 3..N	Topic Filter							
Requested QoS								
	Reserved						QoS	
byte N+1	0	0	0	0	0	0	X	X

960

961 The upper 6 bits of the Requested QoS byte are not used in the current version of the protocol. They are
962 reserved for future use. The Server MUST treat a SUBSCRIBE packet as malformed and close the
963 Network Connection if any of Reserved bits in the payload are non-zero. [MQTT-3-8.3-2]

964

965

3.8.3.1 Payload Non Normative Example

Topic Name	"a/b"
Requested QoS	0x01
Topic Name	"c/d"
Requested QoS	0x02

966

967 The format of the example payload is shown in the table below.

968

	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length MSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Requested QoS									
byte 6	Requested QoS(1)	0	0	0	0	0	0	0	1
Topic Filter									
byte 7	Length MSB (0)	0	0	0	0	0	0	0	0
byte 8	Length MSB (3)	0	0	0	0	0	0	1	1
byte 9	'c' (0x63)	0	1	1	0	0	0	1	1
byte 10	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 11	'd' (0x64)	0	1	1	0	0	1	0	0
Requested QoS									
byte 12	Requested QoS(2)	0	0	0	0	0	0	1	0

969

970 3.8.4 Response

971 When the Server receives a SUBSCRIBE Packet from a Client, the Server MUST respond with a
 972 SUBACK Packet [MQTT-3.8.4-1]. The SUBACK Packet MUST have the same Packet Identifier as the
 973 SUBSCRIBE Packet that it is acknowledging [MQTT-3.8.4-2].

974

975 The Server MAY start sending PUBLISH packets matching the Subscription before the Server sends the
 976 SUBACK Packet.

977

978 If a Server receives a SUBSCRIBE Packet containing a Topic Filter that is identical to an existing
 979 Subscription's Topic Filter then it MUST completely replace that existing Subscription with a new

980 Subscription. The Topic Filter in the new Subscription will be identical to that in the previous Subscription,
981 although its maximum QoS value could be different. Any existing retained publications matching the Topic
982 Filter MUST be resent, but the flow of publications MUST NOT be interrupted. [MQTT-3.8.4-3]

983

984 Where the Topic Filter is not identical to any existing Subscription's filter, a new Subscription is created
985 and all matching retained publications are sent.

986

987 If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet
988 as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses
989 into a single SUBACK response. [MQTT-3.8.4-4]

990

991 The SUBACK Packet sent by the Server to the Client MUST contain a return code for each Topic
992 Filter/QoS pair. This return code MUST either show the maximum QoS that was granted for that
993 Subscription or indicate that the subscription failed. [MQTT-3.8.4-5] The Server might grant a lower
994 maximum QoS than the subscriber requested. The QoS of Payload Messages sent in response to a
995 Subscription MUST be the minimum of the QoS of the originally published message and the maximum
996 QoS granted by the Server. The server is permitted to send duplicate copies of a message to a
997 subscriber in the case where the original message was published with QoS 1 and the maximum QoS
998 granted was QoS 0.[MQTT-3.8.4-6]

999

1000 **Non-normative examples:**

1001

1002 If a subscribing Client has been granted maximum QoS 1 for a particular Topic Filter, then a QoS 0
1003 Application Message matching the filter is delivered to the Client at QoS 0. This means that at most one
1004 copy of the message is received by the Client. On the other hand a QoS 2 Message published to the
1005 same topic is downgraded by the Server to QoS 1 for delivery to the Client, so that Client might receive
1006 duplicate copies of the Message.

1007

1008 If the subscribing Client has been granted maximum QoS 0, then an Application Message originally
1009 published as QoS 2 might get lost on the hop to the Client, but the Server should never send a duplicate
1010 of that Message. A QoS 1 Message published to the same topic might either get lost or duplicated on its
1011 transmission to that Client.

1012

1013 **Non normative comment.**

1014 Subscribing to a Topic Filter at QoS 2 is equivalent to saying "I would like to receive Messages matching
1015 this filter at the QoS with which they were published". This means a publisher is responsible for
1016 determining the maximum QoS a Message can be delivered at, but a subscriber is able to require that the
1017 Server downgrades the QoS to one more suitable for its usage.

1018

1019 **3.9 SUBACK – Subscribe acknowledgement**

1020

1021 A SUBACK Packet is sent by the Server to the Client to confirm receipt and processing of a SUBSCRIBE
1022 Packet.

1023

1024 A SUBACK Packet contains a list of return codes, that specify the maximum QoS level that was granted
1025 in each Subscription that was requested by the SUBSCRIBE.

1026 **3.9.1 Fixed header**

1027 The table below shows the fixed header format.

1028

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (9)				Reserved			
	1	0	0	1	0	0	0	0
byte 2	Remaining Length							

1029

1030 Remaining Length field

1031 This is the length of variable header (2 bytes) plus the length of the payload.

1032 3.9.2 Variable header

1033 The variable header contains the Packet Identifier from the SUBSCRIBE Packet that is being
1034 acknowledged. The table below shows the format of the variable header.

1035

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

1036 3.9.3 Payload

1037 The payload contains a list of return codes. Each return code corresponds to a Topic Filter in the
1038 SUBSCRIBE Packet being acknowledged. **The order of return codes in the SUBACK Packet MUST**
1039 **match the order of Topic Filters in the SUBSCRIBE Packet.** [MQTT-3.9.3-1]

1040

1041 The table below shows the Return Code field encoded in a byte.

1042

Bit	7	6	5	4	3	2	1	0
	Return Code							
byte 1	X	0	0	0	0	0	X	X

1043 Allowed return codes:

1044 0x00 - Success - Maximum QoS 0

1045 0x01 - Success - Maximum QoS 1

1046 0x02 - Success - Maximum QoS 2

1047 0x80 - Failure

1048

1049 **SUBACK return codes other than 0x00, 0x01, 0x02 and 0x80 are reserved and MUST NOT be**
1050 **used.** [MQTT-3.9.3-2]

1051 3.9.3.1 Payload Non Normative Example

1052

Success - Maximum QoS 0	0
Success - Maximum QoS 2	2

Failure	128
---------	-----

1053
1054
1055

The payload for this example is shown in the table below.

	Description	7	6	5	4	3	2	1	0
byte 1	Success - Maximum QoS 0	0	0	0	0	0	0	0	0
byte 2	Success - Maximum QoS 2	0	0	0	0	0	0	1	0
byte 3	Failure	1	0	0	0	0	0	0	0

1056

3.10 UNSUBSCRIBE – Unsubscribe from topics

1057
1058
1059

An UNSUBSCRIBE Packet is sent by the Client to the Server, to unsubscribe from topics.

3.10.1 Fixed header

1060
1061
1062

The table below shows an example fixed header format.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (10)				Reserved			
	1	0	1	0	0	0	1	0
byte 2	Remaining Length							

1063
1064
1065
1066
1067

Bits 3,2,1 and 0 of the fixed header of the UNSUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection [MQTT-3.10.1-1].

Remaining Length field

1068
1069

This is the length of variable header (2 bytes) plus the length of the payload.

3.10.2 Variable header

1070
1071
1072
1073
1074

The variable header contains a Packet Identifier. See section 2.3.1 for more details.

The table below shows the format of the variable header.

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

1075

1076 **3.10.2.1 Payload**

1077 The payload for the UNSUBSCRIBE Packet contains the list of Topic Filters that the Client wishes to
 1078 unsubscribe from. The Topic Filters are UTF-8 strings, packed contiguously.

1079

1080 **3.10.2.2 Payload Non Normative example**

1081 The table below shows an example payload.

1082

Topic Filter	"a/b"
Topic Filter	"c/d"

1083

1084 The table below shows the format of this payload.

1085

	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length MSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Topic Filter									
byte 6	Length MSB (0)	0	0	0	0	0	0	0	0
byte 7	Length MSB (3)	0	0	0	0	0	0	1	1
byte 8	'c' (0x63)	0	1	1	0	0	0	1	1
byte 9	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 10	'd' (0x64)	0	1	1	0	0	1	0	0

1086 **3.10.3 Response**

1087 The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be
 1088 compared character-by-character with the current set of Topic Filters held by the Server for the Client. If
 1089 any filter matches exactly then its owning Subscription is deleted, otherwise no additional processing
 1090 occurs [MQTT-3.10.3-1].

1091

1092 If a Server deletes a Subscription:

1093

- It MUST stop adding any new messages for delivery to the Client [MQTT-3.10.3-2].

1094

- It MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client [MQTT-3.10.3-3].

1095

1096

- It MAY continue to deliver any existing messages buffered for delivery to the Client.

1097
 1098 The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet. The
 1099 UNSUBACK Packet MUST have the same Packet Identifier as the UNSUBSCRIBE Packet [MQTT-
 1100 3.10.3-4]. Even where no Topic Subscriptions are deleted, the Server MUST respond with an
 1101 UNSUBACK [MQTT-3.10.3-5].

1102
 1103 If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters it MUST handle that
 1104 packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one
 1105 UNSUBACK response [MQTT-3.10.3-6].

1106 3.11 UNSUBACK – Unsubscribe acknowledgement

1107
 1108 The UNSUBACK Packet is sent by the Server to the Client to confirm receipt of an UNSUBSCRIBE
 1109 Packet.

1110 3.11.1 Fixed header

1111 The table below shows the fixed header format.

1112

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (11)				Reserved			
	1	0	1	1	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

1113 Remaining Length field

1114 This is the length of the variable header. For the UNSUBACK Packet this has the value 2.

1115 3.11.2 Variable header

1116 The variable header contains the Packet Identifier of the UNSUBSCRIBE Packet that is being
 1117 acknowledged. The table below shows the format of the variable header.

1118

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

1119

1120 3.11.3 Payload

1121 The UNSUBACK packet has no payload.

1122

1123 3.12 PINGREQ – PING request

1124

1125 The PINGREQ Packet is sent from a Client to the Server. It can be used to:

- 1126 1. Indicate to the Server that the Client is alive in the absence of any other Control Packets flowing
 1127 from the Client to the Server.
 1128 2. Request that the Server responds to confirm that it is alive.
 1129 3. Exercise the network to indicate that the Network Connection is active.

1130
 1131 This Packet is used in Keep Alive processing, see Section 3.1.2.10 for more details.
 1132

1133 3.12.1 Fixed header

1134 The table below shows the fixed header format.
 1135

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (12)				Reserved			
	1	1	0	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

1136
 1137 **3.12.2 Variable header**

1138 There is no variable header.

1139 3.12.3 Payload

1140 There is no payload.

1141 3.12.4 Response

1142 The Server MUST send a PINGRESP Packet in response to a PINGREQ Packet [MQTT-3.12.4-1].
 1143

1144 3.13 PINGRESP – PING response

1145
 1146 A PINGRESP Packet is sent by the Server to the Client in response to a PINGREQ Packet. It indicates
 1147 that the Server is alive.

1148
 1149 This Packet is used in Keep Alive processing, see Section 3.1.2.10 for more details.
 1150

1151 3.13.1 Fixed header

1152 The table below shows the fixed header format.
 1153

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (13)				Reserved			

	1	1	0	1	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

1154

1155 3.13.2 Variable header

1156 There is no variable header.

1157 3.13.3 Payload

1158 There is no payload.

1159

1160 3.14 DISCONNECT – Disconnect notification

1161

1162 The DISCONNECT Packet is the final Control Packet sent from the Client to the Server. It indicates that
1163 the Client is disconnecting cleanly.

1164 3.14.1 Fixed header

1165 The table below shows the fixed header format.

1166

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (14)				Reserved			
	1	1	1	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

1167 The Server MUST validate that reserved bits are set to zero and disconnect the Client if they are not zero
1168 [MQTT-3.14.1-1].

1169 3.14.2 Variable header

1170 There is no variable header.

1171 3.14.3 Payload

1172 There is no payload.

1173 3.14.4 Response

1174 After sending a DISCONNECT Packet the Client:

- 1175 • MUST close the Network Connection [MQTT-3.14.4-1].
- 1176 • MUST NOT send any more Control Packets on that Network Connection [MQTT-3.14.4-2].

1177

1178 On receipt of DISCONNECT the Server:

- 1179 • MUST discard the Will Message without publishing it [MQTT-3.14.4-3], see Section 3.1.2.5.

1180

- SHOULD close the Network Connection if the Client has not already done so.

1181 4 Operational behavior

1182 4.1 Storing state

1183 The Client and Server implement data storage independently and the duration for which data persists can
1184 be different in each. **The Client and Server MUST store data for at least as long as the Network**
1185 **Connection lasts [MQTT-4.1.0-1].** Qualities of Service guarantees are only valid so long as both Client
1186 and Server store data. Subscriptions and retained publications only survive as long as the Server stores
1187 them.

1188

1189 Non normative comment

1190

1191 Normal operation of the Client of Server may mean that stored state is lost or corrupted because of
1192 administrator action, hardware failure or software failure. An administrator action could be an automated
1193 response to defined conditions. These actions might be prompted by resource constraints or for other
1194 operational reasons. For example the server may determine that based on external knowledge, a
1195 message or messages can no longer be delivered to any current or future client.

1196

1197 Non normative comment.

1198 An MQTT user should evaluate the storage capabilities of the MQTT Client and Server implementations
1199 to ensure that they are sufficient for their needs.

1200

1201 For example, a user wishing to gather electricity meter readings may decide that they need to use QoS 1
1202 messages because they need to protect the readings against loss over the network, however they may
1203 decide that the power supply is sufficiently reliable that the data in the Client and Server can be stored in
1204 volatile memory without too much risk of its loss.

1205

1206 Conversely a parking meter payment application provider might decide that there are no circumstances
1207 where a payment message can be lost so they require that all data are force written to non-volatile
1208 memory before it is transmitted across the network.

1209 4.2 Network Connections

1210 **The Network Connection used to transport the MQTT protocol MUST be an ordered, lossless, stream of**
1211 **bytes from the Client to Server and Server to Client [MQTT-4.2.0-1].**

1212

1213 Non normative comment.

1214 The initial transport protocol used to carry MQTT was TCP/IP as defined in **[RFC793]** The following are
1215 also suitable:

- 1216 • TLS [\[RFC5246\]](#)
- 1217 • WebSocket [\[RFC6455\]](#)

1218

1219 Connectionless network transports such as User Datagram Protocol (UDP) are not suitable on their own
1220 because they might lose or reorder data.

1221

1222 4.3 Quality of Service levels and flows

1223
1224 MQTT delivers Application Messages according to the Quality of Service (QoS) levels defined here. The
1225 delivery protocol is symmetric, in the diagrams below the Client and Server can each take the role of
1226 either Sender or Receiver. In the case of the Client, “Deliver Application Message” means give the
1227 message to the application. In the case of the Server it means send a copy of the Message to each Client
1228 with a matching subscription.

1229 4.3.1 QoS 0: At most once delivery

1230 The message is delivered according to the capabilities of the underlying network. No response is sent by
1231 the receiver and no retry is performed by the sender. The message arrives at the receiver either once or
1232 not at all.

1233
1234 The diagram below shows the QoS 0 protocol flow.

Sender Action	Control Packet	Receiver Action
PUBLISH QoS 0		
	----->	
		Deliver Application Message

1236 4.3.2 QoS 1: At least once delivery

1237 The receiver of a QoS 1 PUBLISH Packet acknowledges receipt with a PUBACK Packet. If the Client
1238 reconnects and the Session is resumed, the sender MUST resend any in-flight QoS 1 messages setting
1239 their Dup flags to 1 [MQTT-4.3.2-1].

1240 The message arrives at the receiver at least once.

1241
1242 A QoS 1 message has a Packet Identifier in its variable header, see Section 2.3.1.

1243
1244 The diagram below shows the QoS 1 protocol flow.

Sender Action	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		

1246
1247 ¹ The receiver is not required to complete delivery of the Application Message before sending the
1248 PUBACK. When its original sender receives the PUBACK packet, ownership of the Application Message

1249 is transferred to the receiver. A Server MUST store the message in accordance to its QoS properties and
 1250 ensure onward delivery to applicable subscribers [MQTT-4.3.2-2].

1251

1252 When it receives a PUBLISH Packet with Dup set to 1 the receiver MUST perform the same actions as
 1253 above (setting Dup to 0 on each first attempt at onwards delivery to a new Client). This might result in a
 1254 redelivery of the Application Message [MQTT-4.3.2-3].

1255

1256 4.3.3 QoS 2: Exactly once delivery

1257 This is the highest quality of service, for use when neither loss nor duplication of messages are
 1258 acceptable. There is an increased overhead associated with this quality of service.

1259 A QoS 2 message has a Packet Identifier in its variable header see Section 2.3.1.

1260 The receiver of a QoS 2 PUBLISH Packet acknowledges receipt with a PUBREC Packet. If the Client
 1261 reconnects and the Session is resumed, the sender MUST resend any in-flight QoS 2 messages setting
 1262 their Dup flags to 1 [MQTT-4.3.3-1].

1263

1264 The diagram below shows the QoS 2 protocol flow. There are two ways in which this can be handled by
 1265 the receiver. They differ in the point within the flow at which the message is made available for onward
 1266 delivery. The choice of approach is implementation specific and does not affect the guarantees of a QoS
 1267 2 flow.

1268

1269

Sender Action	Control Packet	Receiver Action
Store message		
PUBLISH QoS 2 <Packet Identifier> Dup 0		
	----->	
		Store message or Store <Packet Identifier> then Initiate onward delivery of the Application Message ¹
		PUBREC <Packet Identifier>
	<-----	
Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	----->	
		Initiate onward delivery of the Application Message ¹ then discard message or Discard <Packet Identifier>

		Send PUBCOMP <Packet Identifier>
	<-----	
Discard stored state		

1270

1271 ¹ The receiver is not required to complete delivery of the Application Message before sending the
 1272 PUBREC or PUBCOMP. When its original sender receives the PUBREC packet, ownership of the
 1273 Application Message is transferred to the receiver. The Server MUST store the message in accordance to
 1274 its QoS properties and ensure onward delivery to applicable subscribers [MQTT-4.3.3-2].

1275 4.4 Message delivery retry

1276

1277 When a Client reconnects with CleanSession = 0, both the Client and Server MUST redeliver any
 1278 previous in-flight QoS 1 and QoS 2 messages. This means re-sending any unacknowledged PUBLISH
 1279 Packets (where QoS > 0) and PUBREL Packets. [MQTT-4.4.0-1] This is the only circumstance where a
 1280 Client or Server is REQUIRED to redeliver messages. Clients MAY resend SUBSCRIBE and
 1281 UNSUBSCRIBE Packets on reconnect but are not REQUIRED to do this.

1282

1283 While a modern TCP network is unlikely to lose packets, a Client or Server is permitted to attempt
 1284 redelivery of unacknowledged packets at other times. However, redelivery is not encouraged unless a
 1285 network failure has been detected.

1286

1287 The PUBLISH packet MUST have the Dup flag set to 1 when it is redelivered [MQTT-4.4.0-2].

1288

1289 Non Normative comment.

1290

1291 Historically retransmission of Control Packets was required to overcome data loss on some older TCP
 1292 networks. This might remain a concern where MQTT 3.1.1 implementations are to be deployed in such
 1293 environments.

1294 4.5 Message receipt

1295

1296 Under normal circumstances Clients receive messages in response to subscriptions they have created. A
 1297 Client could also receive messages that do not match any of its explicit subscriptions. This can happen if
 1298 the Server automatically assigned a subscription to the Client or while an UNSUBSCRIBE operation is in
 1299 progress. The Client MUST acknowledge any Publish Packet it receives according to the applicable QoS
 1300 rules regardless of whether it elects to process the application message that it contains [MQTT-4.5.0-1].

1301

1302 4.6 Message ordering

1303 A Client MUST follow these rules when implementing the protocol flows defined elsewhere in this chapter:

- 1304 • When it resends any PUBLISH packets, it MUST resend them in the order in which the original
 1305 PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages) [MQTT-4.6.0-1]
- 1306 • It MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were
 1307 received (QoS 1 messages) [MQTT-4.6.0-2]
- 1308 • It MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were
 1309 received (QoS 2 messages) [MQTT-4.6.0-3]
- 1310 • It MUST send PUBREL packets in the order in which the corresponding PUBREC packets were
 1311 received (QoS 2 messages) [MQTT-4.6.0-4]

1312

1313 A Server MUST by default treat each Topic as an "Ordered Topic". It MAY provide an administrative or
1314 other mechanism to allow one or more Topics to be treated as an "Unordered Topic" [MQTT-4.6.0-5].

1315

1316 When a Server processes a message that has been published to an Ordered Topic, it MUST follow the
1317 rules listed above when delivering messages to each of its subscribers. In addition it MUST send
1318 PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from
1319 any given Client [MQTT-4.6.0-6].

1320

1321 **Non Normative comment.**

1322

1323 The rules listed above ensure that when a stream of messages is published and subscribed to with QoS =
1324 1, the final copy of each message received by the subscribers will be in the order that they were originally
1325 published in, but the possibility of message duplication could result in a resend of an earlier message
1326 being received after one of its successor messages. For example a publisher might send messages in the
1327 order 1,2,3,4 and the subscriber might receive them in the order 1,2,3,2,3,4.

1328

1329 If both Client and Server make sure that no more than one message is "in-flight" at any one time (by not
1330 sending a message until its predecessor has been acknowledged), then no QoS 1 message will be
1331 received after any later one - for example a subscriber might receive them in the order 1,2,3,3,4 but not
1332 1,2,3,2,3,4. Setting an in-flight window of 1 also means that order will be preserved even if the publisher
1333 sends a sequence of messages with different QoS levels on the same topic.

1334 4.7 Topic Names and Topic Filters

1335 4.7.1 Topic wildcards

1336 The topic level separator is used to introduce structure into the Topic Name. If present, it divides the
1337 Topic Name into multiple "topic levels".

1338 A subscription's Topic Filter may contain special wildcard characters, which allow you to subscribe to
1339 multiple topics at once.

1340 The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name
1341 [MQTT-4.7.1-1].

1342 4.7.1.1 Topic level separator

1343 The forward slash ('/' U+002F) is used to separate each level within a topic tree and provide a hierarchical
1344 structure to the Topic Names. The use of the topic level separator is significant when either of the two
1345 wildcard characters are encountered in Topic Filters specified by subscribing Clients. Topic level
1346 separators may appear anywhere in a Topic Filter or Topic Name. Adjacent Topic level separators
1347 indicate a zero length topic level.

1348 4.7.1.2 Multi-level wildcard

1349 The number sign ('#' U+0023) is a wildcard character that matches any number of levels within a
1350 topic. The multi-level wildcard represents the parent and any number of child levels. The multi-level
1351 wildcard character MUST be specified either on its own or following a topic level separator. In either case
1352 it MUST be the last character specified in the Topic Filter [MQTT-4.7.1-2].

1353

1354 **Non normative comment.**

1355 For example, if a Client subscribes to "sport/tennis/player1/#", it would receive messages published using
1356 these topic names:

1357

1358 "sport/tennis/player1"

1359 "sport/tennis/player1/ranking"

1360 "sport/tennis/player1/score/wimbledon"

1361

1362

1363 **Non normative comment.**

1364

1365 • "sport/#" also matches the singular "sport", since # includes the parent level.

1366 • "#" is valid and will receive every publication

1367 • "sport/tennis/#" is valid

1368 • "sport/tennis#" is not valid

1369 • "sport/tennis#/ranking" is not valid

1370

1371 **4.7.1.3 Single level wildcard**

1372 The plus sign ('+' U+002B) is a wildcard character that matches only one topic level.

1373

1374 The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where
1375 it is used it MUST occupy an entire level of the filter [MQTT-4.7.1-3]. It can be used at more than one
1376 level in the Topic Filter and can be used in conjunction with the multilevel wildcard.

1377

1378 **Non normative comment.**

1379 For example, "sport/tennis/+" matches "sport/tennis/player1" and "sport/tennis/player2", but not
1380 "sport/tennis/player1/ranking". Also, because the single-level wildcard matches only a single level,
1381 "sport/+" does not match "sport" but it does match "sport/".

1382

1383 **Non normative comment.**

1384 • "+" is valid

1385 • "+/tennis/#" is valid

1386 • "sport+" is not valid

1387 • "sport+/player1" is valid

1388 • "/finance" matches "+/+" and "/+", but not "+"

1389 **4.7.2 Topics beginning with \$**

1390

1391 MQTT Server implementations MAY define Topic Names that start with a leading \$ character

1392

1393 **Non normative comment.**

1394

1395 • \$SYS/ has been widely adopted as a prefix to topics that contain Server-specific information or
1396 control APIs

1397 • Applications cannot use a topic with a leading \$ character for their own purposes

1398
1399

1400 4.7.2.1 Subscription handling

1401

1402 A Topic Filter that starts with a wildcard character (# or +) does not match Topic Names that begin with a
1403 \$ character

1404

1405 **Non normative comment.**

1406

- 1407 • A subscription to “#” will not receive any messages published to a topic beginning with a \$
- 1408 • A subscription to “+/monitor/Clients” will not receive any messages published to
- 1409 “\$SYS/monitor/Clients”
- 1410 • A subscription to “\$SYS/#” will receive messages published to topics beginning with “\$SYS/”
- 1411 • A subscription to “\$SYS/monitor/+” will receive messages published to “\$SYS/monitor/Clients”
- 1412 • For a Client to receive messages from topics that begin with \$SYS/ and from topics that don't
- 1413 begin with a \$, it must subscribe to both “#” and “\$SYS/#”

1414

1415 4.7.3 Topic semantic and usage

1416

1417 The following rules apply to Topic Names and Topic Filters

1418

- 1419 • All Topic Names and Topic Filters MUST be at least one character long [MQTT-4.7.3-1]
- 1420 • Topic Names and Topic Filters are case sensitive
- 1421 • Topic Names and Topic Filters can include the space character
- 1422 • A leading or trailing "/" creates a distinct Topic Name or Topic Filter
- 1423 • A Topic Name or Topic Filter consisting only of the "/" character is valid
- 1424 • Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000)
- 1425 [Unicode63] [MQTT-4.7.3-2]
- 1426 • Topic Names and Topic Filters are UTF-8 encoded strings, they MUST NOT encode to more than
- 1427 65535 bytes [MQTT-4.7.3-3]. See Section 2.1.2
- 1428 • There is no limit to the number of levels in a Topic Name or Topic Filter, other than that imposed
- 1429 by the overall length of the UTF-8 encoded string.
- 1430 • When it performs subscription matching the Server does not perform any normalization of Topic
- 1431 Names or Topic Filters, or any modification or substitution of unrecognised characters. Each non-
- 1432 wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name
- 1433 character for character for the match to succeed.

1434

1435 **Non-normative comment.**

1436 The UTF-8 encoding rules mean that the comparison of Topic Filter and Topic Name could be performed
1437 either by comparing the encoded UTF-8 bytes, or by comparing decoded Unicode characters

1438

1439 **Non normative comment.**

- 1440 • “ACCOUNTS” and “Accounts” are two different topic names

- 1441
- “Accounts payable” is a valid topic name
- 1442
- “/finance” is different from “finance”

1443

1444 **Non Normative comment.**

1445

1446 A publication is sent to each Client Subscription whose Topic Filter matches the Topic Name in the
1447 publication. The topic resource may be either predefined in the Server by an administrator or it may be
1448 dynamically created by the Server when it receives the first subscription or publication with that Topic
1449 Name. The Server may also use a security component to selectively authorize actions on the topic
1450 resource for a given Client.

1451 **4.8 Handling protocol violations**

1452 **If the Client or Server encounters a transient error while processing an inbound Control Packet it MUST**
1453 **close the Network Connection on which it received that packet [MQTT-4.8.0-1].** If a Server detects a
1454 transient error it SHOULD NOT disconnect or have any other affect on its interactions with any other
1455 Client.

1456 5 Security

1457

1458 The recommendations contained in this chapter are provided for guidance only and are not intended to
1459 serve as a complete reference on the subject.

1460

1461 There are a number of threats that solution providers should consider. For example:

1462

- 1463 • Devices may be compromised
- 1464 • Data at rest in Clients and Servers may be accessible
- 1465 • Protocol behaviors may have side effects (e.g., 'timing attacks')
- 1466 • Denial of Service (DoS) attacks
- 1467 • Communications may be intercepted, altered, re-routed or disclosed
- 1468 • Injection of spoofed Control Packets

1469

1470 MQTT solutions are often deployed in hostile communication environments. In such cases,
1471 implementations will often need to provide mechanisms for:

1472

- 1473 • Authentication of users and devices
- 1474 • Authorization of access to Server resources
- 1475 • Integrity of MQTT Control Packets and application data contained therein
- 1476 • Privacy of MQTT Control Packets and application data contained therein

1477

1478 As a transport protocol, MQTT is concerned only with message transmission and it is the implementer's
1479 responsibility to provide appropriate security features. This is commonly achieved by using TLS
1480 **[RFC5246]**.

1481 .

1482 Server implementations that offer TLS **[RFC5246]** SHOULD use TCP port 8883 [IANA service name:
1483 secure-mqtt].

1484

1485 In addition to technical security issues there may also be geographic (e.g., European SafeHarbour
1486 **[USEUSAFEHARB]**), industry specific (e.g., PCI DSS **[PCIDSS]**) and regulatory considerations (e.g.,
1487 Sarbanes-Oxley **[SARBANES]**).

1488

1489 **The remainder of this chapter is Non Normative.**

1490

1491 5.1 MQTT solutions: security and certification

1492

1493 An implementation may want to provide conformance with specific industry security standards such as
1494 NIST Cyber Security Framework [NISTCSF], PCI-DSS [PCIDSS], FIPS-140-2 [FIPS1402] and NSA
1495 Suite B [NSAB].

1496 .
1497 Guidance on using MQTT within the NIST Cyber Security Framework [NISTCSF] can be found in MQTT
1498 Supplemental Publication Version 1.0 Part 1: NIST Cyber Security Framework :[http://docs.oasis-](http://docs.oasis-open.org/mqtt/mqtt-security-sp/v1.0/csd01/part1-nist/mqtt-security-sp-v1.0-csd01-part1-nist.doc)
1499 [open.org/mqtt/mqtt-security-sp/v1.0/csd01/part1-nist/mqtt-security-sp-v1.0-csd01-part1-nist.doc](http://docs.oasis-open.org/mqtt/mqtt-security-sp/v1.0/csd01/part1-nist/mqtt-security-sp-v1.0-csd01-part1-nist.doc)

1500
1501 The use of industry proven, independently verified and certified technologies will help meet compliance
1502 requirements.

1503

1504 **5.2 Lightweight cryptography and constrained devices**

1505
1506 Advanced Encryption Standard [AES] and Data Encryption Standard [DES] are widely adopted.

1507
1508 ISO 29192 [ISO29192] makes recommendations for cryptographic primitives specifically tuned to perform
1509 on constrained 'low end' devices.

1510

1511 **5.3 Implementation notes**

1512
1513 There are many security concerns to consider when implementing or using MQTT. The following section
1514 should not be considered a “check list”.

1515
1516 An implementation might want to achieve some, or all, of the following:

1517

1518 **5.3.1 Authentication of Clients by the Server**

1519
1520 The CONNECT Packet contains Username and Password fields. Implementations can choose how to
1521 make use of the content of these fields. They may provide their own authentication mechanism, use an
1522 external authentication system such as LDAP [RFC4511] or OAuth [RFC6749] tokens, or leverage
1523 operating system authentication mechanisms.

1524
1525 Implementations passing authentication data in clear text, obfuscating such data elements or requiring no
1526 authentication data should be aware this may give rise to Man-in-the-Middle and replay attacks. Section
1527 5.3.5 introduces approaches to ensure data privacy.

1528
1529 A Virtual Private Network (VPN) between the Clients and Servers can provide confidence that data is only
1530 being received from authorized Clients.

1531
1532 Where TLS [RFC5246] is used, SSL Certificates flowed from the Client can be used by the Server to
1533 authenticate the Client.

1534
1535 An implementation might allow for authentication where the credentials are flowed in an Application
1536 Message from the Client to the Server.
1537

1538 **5.3.2 Authorization of Clients by the Server**

1539
1540 An implementation may restrict access to Server resources based on information provided by the Client
1541 such as User Name, Client Identifier, the hostname/IP address of the Client, or the outcome of
1542 authentication mechanisms.

1543
1544

1545 **5.3.3 Authentication of the Server by the Client**

1546
1547 The MQTT protocol is not trust symmetrical: it provides no mechanism for the Client to authenticate the
1548 Server.

1549
1550 Where TLS [RFC5246] is used, SSL Certificates flowed from the Server can be used by the Client to
1551 authenticate the Server. Implementations providing MQTT service for multiple hostnames from a single IP
1552 address should be aware of section-3.1 of the Server Name Indication extension to TLS [RFC3546] .This
1553 allows a Client to tell the Server the hostname of the Server it is trying to connect to.

1554
1555 An implementation may allow for authentication where the credentials are flowed in an Application
1556 Message from the Server to the Client.

1557
1558 A VPN between Clients and Servers can provide confidence that Clients are connecting to the intended
1559 Server.

1560

1561 **5.3.4 Integrity of Application Messages and Control Packets**

1562
1563 Applications can independently include hash values in their Application Messages. This can provide
1564 integrity of the contents of Publish Control Packets across the network and at rest.

1565
1566 TLS [RFC5246] provides hash algorithms to verify the integrity of data sent over the network.

1567
1568 The use of VPNs to connect Clients and Servers can provide integrity of data across the section of the
1569 network covered by a VPN.

1570

1571 **5.3.5 Privacy of Application Messages and Control Packets**

1572

1573 TLS [RFC5246] can provide encryption of data sent over the network. There are valid TLS cipher suites
1574 that include a NULL encryption algorithm that does not encrypt data. To ensure privacy Clients and
1575 Servers should avoid these cipher suites.

1576
1577 An application may independently encrypt the contents of its Application Messages. This could provide
1578 privacy of the Application Message both over the network and at rest. This would not provide privacy for
1579 other properties of the Application Message such as Topic Name.

1580
1581 Client and Server implementations may provide encrypted storage for data at rest such as Application
1582 Messages stored as part of a Session.

1583
1584 The use of VPNs to connect Clients and Servers can provide privacy of data across the section of the
1585 network covered by a VPN.

1586

1587 **5.3.6 Non-repudiation of message transmission**

1588
1589 Application designers might need to consider appropriate strategies to achieve end to end non-
1590 repudiation.

1591

1592 **5.3.7 Detecting compromise of Clients and Servers**

1593
1594 Client and Server implementations using TLS [RFC5246] should provide capabilities to ensure that any
1595 SSL certificates provided when initiating a TLS [RFC5246] connection are associated with the hostname
1596 of the Client connecting or Server being connected to.

1597
1598 Client and Server implementations using TLS [RFC5246] may choose to provide capabilities to check
1599 Certificate Revocation Lists (CRLs [RFC5280]) and Online Certificate Status Protocol (OCSP) [RFC6960]
1600 to prevent revoked certificates from being used.

1601
1602 Physical deployments might combine tamper-proof hardware with the transmission of specific data in
1603 Application Messages. For example a meter might have an embedded GPS to ensure it is not used in an
1604 unauthorized location. [IEEE 802.1AR] is a standard for implementing mechanisms to authenticate a
1605 device's identity using a cryptographically bound identifier.

1606

1607 **5.3.8 Detecting abnormal behaviors**

1608
1609 Server implementations might monitor Client behavior to detect potential security incidents. For example:

1610

- 1611 • Repeated connection attempts
- 1612 • Repeated authentication attempts
- 1613 • Abnormal termination of connections

- 1614
- Topic scanning (attempts to send or subscribe to many topics)
- 1615
- Sending undeliverable messages (no subscribers to the topics)
- 1616
- Clients that connect but do not send data

1617

1618 Server implementations might disconnect Clients that breach its security rules.

1619

1620 Server implementations detecting unwelcome behavior might implement a dynamic block list based on
1621 identifiers such as IP address or Client Identifier.

1622

1623 Deployments might use network level controls (where available) to implement rate limiting or blocking
1624 based on IP address or other information.

1625

1626

1627

1628 **5.3.9 Other security considerations**

1629

1630 If Client or Server SSL certificates are lost or it is considered that they might be compromised they should
1631 be revoked (utilising CRLs [RFC5280] and/or OSCP [RFC6960]).

1632

1633 Client or Server authentication credentials, such as User Name and Password, that are lost or considered
1634 compromised should be revoked and/or reissued.

1635

1636 In the case of long lasting connections (such as meters):

1637

- Client and Server implementations using TLS [RFC5246] should allow for session renegotiation
1639 to establish new cryptographic parameters (replace session keys, change cipher suites, change
1640 authentication credentials).

- Servers may disconnect Clients and require them to re-authenticate with new credentials.

1642

1643 Constrained devices and Clients on constrained networks can make use of TLS session resumption
1644 [RFC5077], in order to reduce the costs of reconnecting TLS [RFC5246] sessions.

1645

1646 Clients connected to a Server have a transitive trust relationship with other Clients connected to the same
1647 Server and who have authority to publish data on the same topics.

1648

1649 **5.3.10 Use of SOCKS**

1650

1651 Implementations of Clients should be aware that some environments will require the use of SOCKSv5
1652 [RFC1928] proxies to make outbound Network Connections. Some MQTT implementations may make
1653 use of alternative secured tunnels (e.g. SSH) through the use of SOCKS. Where implementations choose
1654 to use SOCKS, they should support both anonymous and user-name password authenticating SOCKS
1655 proxies. In the latter case, implementations should be aware that SOCKS authentication may occur in
1656 plain-text and so should avoid using the same credentials for connection to a MQTT Server.

1657

1658 **5.3.11 Security profiles**

1659

1660 Implementers and solution designers may wish to consider security as a set of profiles which can be
1661 applied to the MQTT protocol. An example of a layered security hierarchy is presented below.

1662

1663 **5.3.11.1 Clear communication profile**

1664

1665 MQTT protocol running over an open network with no additional secure communication mechanisms in
1666 place.

1667

1668

1669

1670 **5.3.11.2 Secured network communication profile**

1671

1672 MQTT protocol running over a physical or virtual network which has security controls e.g., VPNs or
1673 physically secure network.

1674

1675 **5.3.11.3 Secured transport profile**

1676

1677 MQTT protocol running over a physical or virtual network and using TLS **[RFC5246]** which provides
1678 authentication, integrity and privacy.

1679

1680 TLS **[RFC5246]** Client authentication may be used in addition to – or in place of – MQTT Client
1681 authentication as provided by the Username and Password fields.

1682

1683 **5.3.11.4 Industry specific security profile**

1684

1685 It is anticipated that the MQTT protocol will be designed into industry specific application profiles, each
1686 defining a threat model and the specific security mechanisms to be used to address these threats.
1687 Recommendations for specific security mechanisms will often be taken from existing works including:

1688

1689 **[NISTCSF]** NIST Cyber Security Framework

1690 **[NIST7628]** NISTIR 7628 Guidelines for Smart Grid Cyber Security

1691 **[FIPS1402]** Federal Information Processing Standards (FIPS-140-2)

1692 **[PCIDSS]** PCI-DSS Payment Card Industry Data Security Standard

1693 **[NSAB]** NSA Suite B Cryptography

1694

1695 An MQTT supplemental publication: MQTT security standards will provide further information related to
1696 the usage of various industry security frameworks and standards.

1697 **6 Using WebSocket as a network transport.**

1698 MQTT can be transported over a WebSocket **[RFC6455]** connection using the following conventions:

- 1699 • WebSocket binary frames are used. A single frame may contain multiple or partial MQTT Control
1700 Packets; they are not required to be aligned.
- 1701 • The WebSocket Protocol Name consists of the MQTT Protocol Name concatenated with the
1702 ASCII representation of the MQTT Protocol Version number. For MQTT v3.1.1, this will be
1703 "MQTT4".
- 1704 • No restriction is placed on the path portion of the WebSocket url.

1705 7 Conformance

1706

1707 The MQTT specification defines conformance for MQTT Client implementations and MQTT Server
1708 implementations.

1709

1710 A single entity MAY conform as both an MQTT Client and MQTT Server implementation. For example, a
1711 Server that both accepts inbound connections and establishes outbound connections to other Servers
1712 MUST conform as both an MQTT Client and MQTT Server.

1713

1714 Conformant implementations SHALL NOT require the use of any extensions defined outside of this
1715 specification in order to interoperate with any other conformant implementation.

1716 7.1 Conformance Targets

1717 7.1.1 MQTT Server

1718

1719 An MQTT Server accepts Network Connections from MQTT Clients.

1720

1721 An MQTT Server conforms to this specification only if it satisfies all the statements below:

- 1722 1. The syntax of all Control Packets that it sends matches the syntax described in Chapters 2 and 3.
1723 2. It follows the Topic matching rules described in Section 4.7.
1724 3. It satisfies all of the MUST level requirements in the following chapters that are identified for the Server:
- 1725 - [MQTT0001] Chapter 2 - MQTT Control Packet format
 - 1726 - [MQTT0002] Chapter 3 - MQTT Control Packets
 - 1727 - [MQTT0003] Chapter 4 - Operational behavior
 - 1728 - [MQTT0004] Chapter 5 - Security

1729

1730 7.1.2 MQTT Client

1731

1732 An MQTT Client creates a Network Connection to an MQTT Server.

1733

1734 An MQTT Client conforms to this specification only if it satisfies all the statements below:

- 1735 1. The syntax of all Control Packets that it sends matches the syntax described in chapters 2 and 3.
1736 2. It satisfies all of the MUST level requirements in the following chapters that are identified for the Client:
- 1737 - [MQTT0005] Chapter 2 - MQTT Control Packet format
 - 1738 - [MQTT0006] Chapter 3 - MQTT Control Packets
 - 1739 - [MQTT0007] Chapter 4 - Operational behavior
 - 1740 - [MQTT0008] Chapter 5 – Security

Appendix A. Mandatory normative statements

Normative Statement Number	Normative Statement
[MQTT-1.1.0-1]	A Session can contain more than one Subscription. Each Subscription within a session MUST have a different Topic Filter.
[MQTT-1.4.0-1]	The encoded data MUST be well-formed UTF-8 as defined by the Unicode spec [Unicode63] and restated in RFC 3629 [RFC 3629]. In particular the encoded data MUST NOT include encodings of code points between U+D800 and U+DFFF. If a receiver (Server or Client) receives a control packet containing ill-formed UTF-8 it MUST close the network connection.
[MQTT-1.4.0-2]	The UTF-8 encoded string MUST NOT include an encoding of the null character U+0000. If a receiver (Server or Client) receives a control packet containing U+0000 it MUST close the network connection.
[MQTT-1.4.0-3]	The UTF-8 encoded sequence 0xEF 0xBB 0xBF is always to be interpreted to mean 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-2.0.0-1]	Unless stated otherwise, if either the Server or Client receives a Control Packet which does not meet this specification, it MUST close the Network Connection.
[MQTT-2.1.2-1]	If invalid flags are received, the receiver MUST close the Network connection.
[MQTT-2.1.2-2].	If Dup is 0 then the flow is the first occasion that the Client or Server has attempted to send the MQTT PUBLISH Packet. If Dup is 1 then this indicates that the flow might be re-delivery of an earlier packet.
[MQTT-2.1.2-3]	The Dup flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet.
[MQTT-2.1.2-4]	The Dup flag MUST be 0 for all QoS 0 messages
[MQTT-2.1.2-5]	The value of the Dup flag from an incoming PUBLISH packet is not propagated when the PUBLISH Packet is sent to subscribers by the Server. The Dup flag in the outgoing PUBLISH packet MUST BE set independently to the incoming PUBLISH packet.
[MQTT-2.1.2-6]	If the retain flag is set to 1 , in a PUBLISH Packet sent by a Client to a Server, the Server MUST store the application message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name.
[MQTT-2.1.2-7]	When a new subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber.
[MQTT-2.1.2-8]	If the Server receives a QoS 0 message with the RETAIN flag set to 1 it MUST discard any message previously retained for that topic. It SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY discard it at any time. If this happens there will be no retained message for that topic.
[MQTT-2.1.2-9]	When sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag to 1 if a message is sent as a result of a new subscription being made by a Client .

[MQTT-2.1.2-10]	It MUST set the RETAIN flag to 0 when a PUBLISH Packet is sent to a Client because it matches an established subscription regardless of how the flag was set in the message it received
[MQTT-2.1.2-11]	A PUBLISH Packet with a retain flag set to 1 and a payload containing zero bytes will be processed as normal by the Server and sent to Clients with a subscription matching the topic name. Additionally any existing retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message.
[MQTT-2.1.2-12]	If the RETAIN flag is 0, in a PUBLISH Packet sent by a Client to a Server, the Server MUST NOT store the message and MUST NOT remove or replace any existing retained message.
[MQTT-2.3.1-1]	SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) Control Packets MUST contain a non-zero 16-bit Packet Identifier.
[MQTT-2.3.1-2]	Each time a Client sends a new packet of one of these types it MUST assign it a currently unused Packet Identifier.
[MQTT-2.3.1-3]	If a Client resends a particular Control Packet, then it MUST use the same Packet Identifier in subsequent resends of that packet. The Packet Identifier becomes available for reuse after the Client has processed the corresponding acknowledgement packet. In the case of a QoS 1 PUBLISH this is the corresponding PUBACK; in the case of QoS 2 it is PUBCOMP. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK.
[MQTT-2.3.1-4]	The same conditions [MQTT-2.3.1-3] apply to a Server when it sends a PUBLISH with QoS >0.
[MQTT-2.3.1-5]	A PUBLISH Packet MUST NOT contain a Packet Identifier if its QoS value is set to 0.
[MQTT-2.3.1-6]	A PUBACK, PUBREC, PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH Packet that initiated the flow.
[MQTT-2.3.1-7]	Similarly to [MQTT-2.3.1-6], SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE Packet respectively
[MQTT-3.1.0-1]	After a Network Connection is established by a Client to a Server, the first flow from the Client to the Server MUST be a CONNECT Packet.
[MQTT-3.1.0-2]	The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client.
[MQTT-3.1.2-1].	If the protocol name is incorrect the Server MAY disconnect the Client, or it MAY continue processing the CONNECT packet in accordance with some other specification. In the latter case, the Server MUST NOT continue to process the CONNECT packet in line with this specification
[MQTT-3.1.2-2]	The Server MUST respond to the CONNECT Packet with a CONNACK return code 0x01 (unacceptable protocol level) and then disconnect the Client if the Protocol Level is not supported by the Server.
[MQTT-3.1.2-3]	The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero.
[MQTT-3.1.2-4]	The Client and Server MUST store the Session after the Client and Server are disconnected.

[MQTT-3.1.2-5]	After disconnection, the Server MUST store further QoS 1 and QoS 2 messages that match any subscriptions that the client had at the time of disconnection as part of the Session state.
[MQTT-3.1.2-6]	If set to 1, the Client and Server MUST discard any previous Session and start a new one. This Session lasts as long as the Network Connection. State data associated with this session MUST NOT be reused in any subsequent Session
[MQTT-3.1.2.7]	Retained publications do not form part of the Session state in the Server, they MUST NOT be deleted when the Session ends.
[MQTT-3.1.2-8]	If the Will Flag is set to 1 this indicates that a Will Message MUST be published by the Server when the Server detects that the Client is disconnected for any reason other than the Client flowing a DISCONNECT Packet.
[MQTT-3.1.2-9]	If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the Server, and the Will Topic and Will Message fields MUST be present in the payload.
[MQTT-3.1.2-10]	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 from the Client. If the Will Flag is set to 0, no will message is published.
[MQTT-3.1.2-11]	If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00).
[MQTT-3.1.2-12]	If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). It MUST NOT be 3 (0x03).
[MQTT-3.1.2-13]	If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0.
[MQTT-3.1.2-14]	If the Will Flag is set to 1 and If Will Retain is set to 0, the Server MUST publish the will message as a non-retained publication.
[MQTT-3.1.2-15]	If the Will Flag is set to 1 and If Will Retain is set to 1, the Server MUST publish the will message as a retained publication.
[MQTT-3.1.2-16]	If the User Name Flag is set to 0, a user name MUST NOT be present in the payload.
[MQTT-3.1.2-17]	If the User Name Flag is set to 1, a user name MUST be present in the payload.
[MQTT-3.1.2-18]	If the Password Flag is set to 0, a password MUST NOT be present in the payload.
[MQTT-3.1.2-19]	If the Password Flag is set to 1, a password MUST be present in the payload.
[MQTT-3.1.2-20]	If the User Name Flag is set to 0 then the Password Flag MUST be set to 0.
[MQTT-3.1.2-21]	It is the responsibility of the Client to ensure that the interval between Control Packets being sent does not exceed the Keep Alive value .In the absence of sending any other Control Packets, the Client MUST send a PINGREQ Packet.
[MQTT-3.1.2-22]	If the Server does not receive a Control Packet from the Client within one and a half times the Keep Alive time period, it MUST disconnect the Network Connection to the Client as if the network had failed.
[MQTT-3.1.3-1]	These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password.
[MQTT-3.1.3-2]	The ClientId MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT connection between the Client and the Server

[MQTT-3.1.3-3]	The Client Identifier (ClientId) MUST be present and MUST be the first field in the payload.
[MQTT-3.1.3-4]	The ClientId MUST comprise only Unicode [Unicode63] characters, and the length of the UTF-8 encoding MUST be at least zero bytes and no more than 65535 bytes.
[MQTT-3.1.3-5]	The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxy Z ABCDEF GHIJKLMNOPQRSTUVWXYZ "
[MQTT-3.1.3-6]	A Server MAY allow a Client to supply a ClientId that has a length of zero bytes. However if it does so the Server MUST treat this as a special case and assign a unique ClientId to that Client. It MUST then process the CONNECT packet as if the Client had provided that unique ClientId.
[MQTT-3.1.3-7]	If the Client supplies a zero-byte ClientId, the Client MUST also set Clean Session to 1.
[MQTT-3.1.3-8]	If the Client supplies a zero-byte ClientId with Clean Session set to 0, the Server MUST respond to the CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network Connection.
[MQTT-3.1.3-9]	If the Server rejects the ClientId it MUST respond to the CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network Connection.
[MQTT-3.1.4-1]	The Server MUST validate that the CONNECT Packet conforms to section 3.1 and close the Network Connection without sending a CONNACK if it does not conform.
[MQTT-3.1.4-2]	If the ClientId represents a Client already connected to the Server then the Server MUST disconnect the existing Client.
[MQTT-3.1.4-3]	If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT Packet.
[MQTT-3.2.0-1]	The first packet sent from the Server to the Client MUST be a CONNACK Packet.
[MQTT-3.2.2-1]	If a server sends a CONNACK packet containing a non-zero return code it MUST then close the Network Connection.
[MQTT-3.2.2-2]	If none of these return codes are deemed applicable, then the Server MUST close the Network Connection without flowing a CONNACK.
[MQTT-3.3.2-1]	The Topic Name MUST be a UTF-8 encoded string.
[MQTT-3.3.2-2]	The Topic Name in the PUBLISH Packet MUST NOT contain wildcard characters.
[MQTT-3.3.2-3]	The Topic Name sent to a subscribing Client MUST match the Subscription's Topic Filter.
[MQTT-3.3.5-1]	The Server MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions.
[MQTT-3.3.5-2]	If a Server implementation does not authorize a PUBLISH to be performed by a Client; it has no way of informing that Client. It MUST either make a positive

	acknowledgement, according to the normal QoS rules or disconnect the TCP session.
[MQTT-3.5.4-1]	When the sender of a PUBLISH Packet receives a PUBREC Packet, it MUST reply with a PUBREL Packet.
[MQTT-3.6.1-1]	Bits 3,2,1 and 0 of the fixed header in the PUBREL Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.6.4-1]	When the sender of a PUBREC Packet receives a PUBREL Packet it MUST reply with a PUBCOMP Packet.
[MQTT-3.8.1-1]	Bits 3,2,1 and 0 of the fixed header of the SUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.8.3-1]	The Payload of a SUBSCRIBE packet MUST contain at least one Topic Filter / QoS pair. A SUBSCRIBE packet with no payload is a protocol violation.
[MQTT-3.8.3-2]	The Server MUST treat a SUBSCRIBE packet as malformed and close the Network Connection if any of Reserved bits in the payload are non-zero.
[MQTT-3.8.4-1]	When the Server receives a SUBSCRIBE Packet from a Client, the Server MUST respond with a SUBACK Packet.
[MQTT-3.8.4-2]	The SUBACK Packet MUST have the same Packet Identifier as the SUBSCRIBE Packet.
[MQTT-3.8.4-3]	A subscribe request which contains a Topic Filter that is identical to an existing Subscription's Topic Filter completely replaces that existing Subscription with a new Subscription. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its maximum QoS value could be different. Any existing retained publications matching the Topic Filter are resent, but the flow of publications is not interrupted.
[MQTT-3.8.4-4]	If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses into a single SUBACK response.
[MQTT-3.8.4-5]	The SUBACK Packet sent by the Server to the Client MUST contain a return code for each Topic Filter/QoS pair. This return code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed.
[MQTT-3.8.4-6]	The Server might grant a lower maximum QoS than the subscriber requested. The QoS of Payload Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published message and the maximum QoS granted by the Server. The server is permitted to send duplicate copies of a message to a subscriber in the case where the original message was published with QoS 1 and the maximum QoS granted was QoS 0.
[MQTT-3.9.3-1]	The order of return codes in the SUBACK Packet MUST match the order of Topic Filters in the SUBSCRIBE Packet.
[MQTT-3.9.3-2]	SUBACK return codes other than 0x00, 0x01, 0x02 and 0x80 are reserved and MUST NOT be used.
[MQTT-3.10.1-1]	Bits 3,2,1 and 0 of the fixed header of the UNSUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat

	any other value as malformed and close the Network Connection.
[MQTT-3.10.3-1]	The Topic Filter (whether containing a wild-card or not) supplied in an UNSUBSCRIBE packet MUST be compared byte-for-byte with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then it is deleted, otherwise no additional processing occurs.
[MQTT-3.10.3-2]	The Server sends an UNSUBACK Packet to the Client in response to an UNSUBSCRIBE Packet, The Server MUST stop adding any new messages for delivery to the Client.
[MQTT-3.10.3-3]	The Server sends an UNSUBACK Packet to the Client in response to an UNSUBSCRIBE Packet, The Server MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client.
[MQTT-3.10.3-4]	The Server sends an UNSUBACK Packet to the Client in response to an UNSUBSCRIBE Packet, The Server MUST send an UNSUBACK packet. The UNSUBACK Packet MUST have the same Packet Identifier as the UNSUBSCRIBE Packet.
[MQTT-3.10.3-5]	Even where no Topic Filters are deleted, the Server MUST respond with an UNSUBACK.
[MQTT-3.10.3-6]	If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one UNSUBACK response.
[MQTT-3.12.4-1]	The Server MUST send a PINGRESP Packet in response to a PINGREQ packet.
[MQTT-3.14.1-1]	The Server MUST validate that reserved bits are set to zero in DISCONNECT Control Packet, and disconnect the Client if they are not zero.
[MQTT-3.14.4-1]	After sending a DISCONNECT Packet the Client MUST close the Network Connection.
[MQTT-3.14.4-2]	After sending a DISCONNECT Packet the Client MUST NOT send any more Control Packets on that Network Connection.
[MQTT-3.14.4-3]	On receipt of DISCONNECT the Server MUST discard the Will Message without publishing it.
[MQTT-4.1.0-1]	The Client and Server MUST store data for at least as long as the Network Connection lasts.
[MQTT-4.2.0-1]	The network connection used to transport the MQTT protocol MUST be an ordered, lossless, stream of bytes from the Client to Server and Server to Client.
[MQTT-4.3.2-1]	The receiver of a QoS 1 PUBLISH Packet acknowledges receipt with a PUBACK Packet. If the Client reconnects and the Session is resumed, the sender MUST resend any in flight QoS 1 messages with the Dup flag set to 1.
[MQTT-4.3.2-2]	The Server MUST store the message in accordance to its QoS properties and ensure onward delivery to applicable subscribers.
[MQTT-4.3.2-3]	When it receives a PUBLISH Packet with Dup set to 1 the receiver MUST perform the same actions as above which might result in a redelivery of the Application Message.
[MQTT-4.3.3-1]	The receiver of a QoS 2 PUBLISH Packet acknowledges receipt with a PUBREC Packet. If the Client reconnects and the Session is resumed, the sender MUST resend any in-flight QoS 2 messages setting their Dup flags to 1.

[MQTT-4.3.3-2]	The Server MUST store the message in accordance to its QoS properties and ensure onward delivery to applicable subscribers.
[MQTT-4.4.0-1]	When a Client reconnects with CleanSession = 0, both the Client and Server MUST redeliver any previous in-flight QoS 1 and QoS 2 messages. This means re-sending any unacknowledged PUBLISH Packets (where QoS > 0) and PUBREL Packets.
[MQTT-4.4.0-2]	The PUBLISH packet MUST have the Dup flag set to 1 when it is redelivered.
[MQTT-4.5.0-1]	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.
[MQTT-4.6.0-1]	When it resends any PUBLISH packets, it MUST resend them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages).
[MQTT-4.6.0-2]	Client MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages).
[MQTT-4.6.0-3]	Client MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages).
[MQTT-4.6.0-4]	Client MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages).
[MQTT-4.6.0-5]	A Server MUST by default treat each Topic as an "Ordered Topic". It MAY provide an administrative or other mechanism to allow one or more Topics to be treated as an "Unordered Topic".
[MQTT-4.6.0-6]	When a Server processes a message that has been published to an Ordered Topic, it MUST follow the rules listed above when delivering messages to each of its subscribers. In addition 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-4.7.1-1]	The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name.
[MQTT-4.7.1-2]	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-4.7.1-3]	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-4.7.3-1]	All Topic Names and Topic Filters MUST be at least one character long.
[MQTT-4.7.3-2]	Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000).
[MQTT-4.7.3-3]	Topic Names and Topic Filters are UTF-8 encoded strings, they MUST NOT encode to more than 65535 bytes.
[MQTT-4.8.0-1]	If the Client or Server encounters a transient error while processing an inbound Control Packet it MUST close Network Connection which was used to send the packet.

1743

Appendix B. Revision history

1744

Revision	Date	Editor	Changes Made
[02]	[29 April 2013]	[A Banks]	[Tighten up language for Connect packet]
[03]	[09 May 2013]	[A Banks]	[Tighten up language in Section 02 Command Message Format]
[04]	[20 May 2013]	[Rahul Gupta]	Tighten up language for PUBLISH message
[05]	[5th June 2013]	[A Banks] [Rahul Gupta]	[Issues -5,9,13] [Formatting and language tighten up in PUBACK, PUBREC, PUBREL, PUBCOMP message]
[06]	[20 th June 2013]	[Rahul Gupta]	[Issue – 17, 2, 28, 33] [Formatting and language tighten up in SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT Control Packets] Terms Command message change to Control Packet Term “message” is generically used, replaced this word accordingly with packet, publication, subscription.
[06]	[21 June 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 12,20,15, 3, 35, 34, 23, 5, 21 Resolved Issues – 32,39, 41
[07]	[03 July 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 18,11,4 Resolved Issues – 26,31,36,37
[08]	[19 July 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 6, 29, 45 Resolved Issues – 36, 25, 24 Added table for fixed header and payload
[09]	[01 August 2013]	[A Banks]	Resolved Issues – 49, 53, 46, 67, 29, 66, 62, 45, 69, 40, 61, 30
[10]	[10 August 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 19, 63, 57, 65, 72 Conformance section added
[11]	[10 September 2013]	[A Banks] [N O’Leary & Rahul Gupta]	Resolved Issues – 56 Updated Conformance section
[12]	[18 September 2013]	[Rahul Gupta] [A Banks]	Resolved Issues – 22, 42, 81, 84, 85, 7, 8, 14, 16, Security section is added Resolved Issue -1

[13]	[27 September 2013]	[A Banks]	Resolved Issues – 64, 68, 76, 86, 27, 60, 82, 55, 78, 51, 83, 80
[14]	[10 October 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 58, 59, 10, 89, 90, 88, 77 Resolved Issues – 94, 96, 93, 92, 95, 87, 74, 71
[15]	[24 October 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 52, 97, 98, 101 Resolved Issues – 100 Added normative statement numbering and Appendix A
[16]	[21 November 2013]	[A Banks]	Resolved Issues -103, 104, 44
[17]	[05 December 2013]	[A Banks] [Rahul Gupta]	Resolved Issues – 105, 70, 102, 106, 107, 108, 109, 110 Updated normative statement numbering and Appendix A

1745

1746