MQTT Version 3.1.1 Plus Errata 01

OASIS Standard Incorporating Approved Errata 01

10 December 2015

Specification URIs

This version:

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.doc (Authoritative)
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.pdf

Previous version:

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

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

Additional artifacts:

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

- *MQTT Version 3.1.1 Errata 01*. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Approved Errata. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os.html.

Related work:

This specification replaces or supersedes:

- *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

This specification is related to:

- *MQTT and the NIST Cybersecurity Framework Version 1.0*. Edited by Geoff Brown and Louis-Philippe Lamoureux. Latest version: http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html.

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 can 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. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt#technical.

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

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 (https://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-plus-errata01]

MQTT Version 3.1.1 Plus Errata 01. Edited by Andrew Banks and Rahul Gupta. 10 December 2015. OASIS Standard Incorporating Approved Errata 01. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

Notices

Table of Contents

Table of Figures and Tables

1 Introduction

1.1 Organization of MQTT

This specification is split into seven chapters:

- Chapter 1 - Introduction
- Chapter 2 - MQTT Control Packet format
- Chapter 3 - MQTT Control Packets
- Chapter 4 - Operational behavior
- Chapter 5 - Security
- Chapter 6 - Using WebSocket as a network transport
- Chapter 7 - Conformance Targets

1.2 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.

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.

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:

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

- Accepts Network Connections from Clients.
- Accepts Application Messages published by Clients.

- Processes Subscribe and Unsubscribe requests from Clients.
- Forwards Application Messages that match Client Subscriptions.

Subscription:

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

Topic Name:

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

Topic Filter:

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

Session:

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

MQTT Control Packet:

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

1.3 Normative references

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

http://www.ietf.org/rfc/rfc2119.txt

[RFC3629]

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

http://www.ietf.org/rfc/rfc3629.txt

[RFC5246]

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

http://www.ietf.org/rfc/rfc5246.txt

[RFC6455]

Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

http://www.ietf.org/rfc/rfc6455.txt

[Unicode]

The Unicode Consortium. The Unicode Standard.

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

1.4 Non normative references

[RFC793]

Postel, J. Transmission Control Protocol. STD 7, IETF RFC 793, September 1981.
http://www.ietf.org/rfc/rfc793.txt

[AES]

Advanced Encryption Standard (AES) (FIPS PUB 197).
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[DES]

Data Encryption Standard (DES).
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[FIPS1402]

Security Requirements for Cryptographic Modules (FIPS PUB 140-2)

http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[IEEE 802.1AR]

IEEE Standard for Local and metropolitan area networks - Secure Device Identity
http://standards.ieee.org/findstds/standard/802.1AR-2009.html

[ISO29192]

ISO/IEC 29192-1:2012 Information technology -- Security techniques -- Lightweight cryptography -- Part 1: General
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56425

[MQTT NIST]

MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure Cybersecurity
http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html

[MQTTV31]

MQTT V3.1 Protocol Specification.

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[NISTCSF]

Improving Critical Infrastructure Cybersecurity Executive Order 13636
http://www.nist.gov/itl/upload/preliminary-cybersecurity-framework.pdf

[NIST7628]

NISTIR 7628 Guidelines for Smart Grid Cyber Security
http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf

[NSAB]

NSA Suite B Cryptography
http://www.nsa.gov/ia/programs/suiteb_cryptography/

[PCIDSS]

PCI-DSS Payment Card Industry Data Security Standard
https://www.pcisecuritystandards.org/security_standards/

[RFC1928]

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, March 1996.

http://www.ietf.org/rfc/rfc1928.txt

[RFC4511]

Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

http://www.ietf.org/rfc/rfc4511.txt

[RFC5077]

Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

http://www.ietf.org/rfc/rfc5077.txt

[RFC5280]

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

http://www.ietf.org/rfc/rfc5280.txt

[RFC6066]

Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

http://www.ietf.org/rfc/rfc6066.txt

[RFC6749]

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

http://www.ietf.org/rfc/rfc6749.txt

[RFC6960]

Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013.
http://www.ietf.org/rfc/rfc6960.txt

[SARBANES]

Sarbanes-Oxley Act of 2002.
http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm

[USEUSAFEHARB]

U.S.-EU Safe Harbor
http://export.gov/safeharbor/eu/eg_main_018365.asp

1.5 Data representations

1.5.1 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.

1.5.2 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).

1.5.3 UTF-8 encoded strings

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

Each of these strings is prefixed with a two byte length field that gives the number of bytes in a UTF-8 encoded string itself, as illustrated in Figure 1.1 Structure of UTF-8 encoded strings below. Consequently 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.

Figure 1.1 Structure of UTF-8 encoded strings

Bit	7	6	5	4	3	2	1	0
byte 1	String length MSB							
byte 2	String length LSB							
byte 3 ….	UTF-8 Encoded Character Data, if length > 0.							

The character data in a UTF-8 encoded string MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular this data MUST NOT include encodings of code points between U+D800 and U+DFFF. If a Server or Client receives a Control Packet containing ill-formed UTF-8 it MUST close the Network Connection [MQTT-1.5.3-1].

A 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-1.5.3-2].

The data SHOULD NOT include encodings of the Unicode [Unicode] code points listed below. If a receiver (Server or Client) receives a Control Packet containing any of them it MAY close the Network Connection:

U+0001..U+001F control characters
U+007F..U+009F control characters

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

A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always 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-1.5.3-3].

1.5.3.1 Non normative example

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

Figure 1.2 UTF-8 encoded string non normative example

Bit	7	6	5	4	3	2	1	0
byte 1	String Length MSB (0x00)							
	0	0	0	0	0	0	0	0
byte 2	String 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

1.6 Editing conventions

Text highlighted in Yellow within this specification identifies conformance statements. Each conformance statement has been assigned a reference in the format [MQTT-x.x.x-y].

2 MQTT Control Packet format

2.1 Structure of an MQTT Control Packet

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

An MQTT Control Packet consists of up to three parts, always in the following order as illustrated in Figure 2.1 - Structure of an MQTT Control Packet.

Figure 2.1 – Structure of an MQTT Control Packet

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

2.2 Fixed header

Each MQTT Control Packet contains a fixed header. Figure 2.2 - Fixed header format illustrates the fixed header format.

Figure 2.2 - Fixed header format

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							

2.2.1 MQTT Control Packet type

Position: byte 1, bits 7-4.

Represented as a 4-bit unsigned value, the values are listed in Table 2.1 - Control packet types.

Table 2.1 - Control packet types

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	Publish message

		Server to Client	
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

2.2.2 Flags

The remaining bits [3-0] of byte 1 in the fixed header contain flags specific to each MQTT Control Packet type as listed in the Table 2.2 - Flag Bits below. Where a flag bit is marked as “Reserved” in Table 2.2 - Flag Bits, it is reserved for future use and MUST be set to the value listed in that table [MQTT-2.2.2-1]. If invalid flags are received, the receiver MUST close the Network Connection [MQTT-2.2.2-2]. See Section 4.8 for details about handling errors.

Table 2.2 - Flag Bits

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[1]	QoS[2]	QoS[2]	RETAIN[3]
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

DUP[1] = Duplicate delivery of a PUBLISH Control Packet

QoS[2] = PUBLISH Quality of Service

RETAIN[3] = PUBLISH Retain flag

See Section 3.3.1 for a description of the DUP, QoS, and RETAIN flags in the PUBLISH Control Packet.

2.2.3 Remaining Length

Position: starts at byte 2.

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

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

Non normative comment

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

Non normative comment

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

Table 2.4 shows the Remaining Length values represented by increasing numbers of bytes.

Table 2.4 Size of Remaining Length field

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)

Non normative comment

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

```
do
       encodedByte = X MOD 128
       X = X DIV 128
      // if there are more data to encode, set the top bit of this byte
      if ( X > 0 )
          encodedByte = encodedByte OR 128
      endif
          'output' encodedByte
 while ( X > 0 )
```

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

Non normative comment

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

```
multiplier = 1
value = 0
do
        encodedByte = 'next byte from stream'
        value += (encodedByte AND 127) * multiplier
        if (multiplier > 128*128*128)
          throw Error(Malformed Remaining Length)
        multiplier *= 128
while ((encodedByte AND 128) != 0)
```

~~multiplier = 1~~
~~value = 0~~
~~do~~
~~encodedByte = 'next byte from stream'~~
~~value += (encodedByte AND 127) * multiplier~~
~~multiplier *= 128~~
~~if (multiplier > 128*128*128)~~
~~throw Error(Malformed Remaining Length)~~

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

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

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

2.3 Variable header

Some types of MQTT Control Packets contain a variable header component. It resides between the fixed header and the payload. The content of the variable header varies depending on the Packet type. The Packet Identifier field of variable header is common in several packet types.

2.3.1 Packet Identifier

Figure 2.3 - Packet Identifier bytes

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

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

SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) Control Packets MUST contain a non-zero 16-bit Packet Identifier [MQTT-2.3.1-1]. Each time a Client sends a new packet of one of these types it MUST assign it a currently unused Packet Identifier [MQTT-2.3.1-2]. If a Client re-sends a particular Control Packet, then it MUST use the same Packet Identifier in subsequent re-sends 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-2.3.1-3]. The same conditions apply to a Server when it sends a PUBLISH with QoS > 0 [MQTT-2.3.1-4].

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

A PUBACK, PUBREC or PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH Packet that was originally sent [MQTT-2.3.1-6]. Similarly SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE Packet respectively [MQTT-2.3.1-7].

Control Packets that require a Packet Identifier are listed in Table 2.5 - Control Packets that contain a Packet Identifier.

Table 2.5 - 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

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

Non normative comment

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

```
Client                    Server
PUBLISH Packet Identifier=0x1234--->
<--PUBLISH Packet Identifier=0x1234
PUBACK Packet Identifier=0x1234--->
<--PUBACK Packet Identifier=0x1234
```

2.4 Payload

Some MQTT Control Packets contain a payload as the final part of the packet, as described in Chapter 3. In the case of the PUBLISH packet this is the Application Message. Table 2.6 - Control Packets that contain a Payload lists the Control Packets that require a Payload.

Table 2.6 - Control Packets that contain a Payload

Control Packet	Payload
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None

PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	None
PINGREQ	None
PINGRESP	None
DISCONNECT	None

371

3 MQTT Control Packets

3.1 CONNECT – Client requests a connection to a Server

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

A Client can only send the CONNECT Packet once over a Network Connection. The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client [MQTT-3.1.0-2]. See section 4.8 for information about handling errors.

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

3.1.1 Fixed header

Figure 3.1 – CONNECT Packet fixed header

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							

Remaining Length field

Remaining Length is the length of the variable header (10 bytes) plus the length of the Payload. It is encoded in the manner described in section 2.2.3.

3.1.2 Variable header

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

3.1.2.1 Protocol Name

Figure 3.2 - Protocol Name bytes

	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

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

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-3.1.2-1].

Non normative comment

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

3.1.2.2 Protocol Level

Figure 3.3 - Protocol Level byte

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

The 8 bit unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Level field for the version 3.1.1 of the protocol is 4 (0x04). 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-3.1.2-2].

3.1.2.3 Connect Flags

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

Figure 3.4 - Connect Flag bits

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

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-3.1.2-3].

3.1.2.4 Clean Session

Position: bit 1 of the Connect Flags byte.

This bit specifies the handling of the Session state.

The Client and Server can store Session state to enable reliable messaging to continue across a sequence of Network Connections. This bit is used to control the lifetime of the Session state.

If CleanSession is set to 0, the Server MUST resume communications with the Client based on state from the current Session (as identified by the Client identifier). If there is no Session associated with the Client identifier the Server MUST create a new Session. The Client and Server MUST store the Session after the Client and Server are disconnected [MQTT-3.1.2-4]. After the disconnection of a Session that had CleanSession set to 0, 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-3.1.2-5]. It MAY also store QoS 0 messages that meet the same criteria.

If CleanSession is 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-3.1.2-6].

The Session state in the Client consists of:

- QoS 1 and QoS 2 messages which have been sent to the Server, but have not been completely acknowledged.
- QoS 2 messages which have been received from the Server, but have not been completely acknowledged.

The Session state in the Server consists of:

- The existence of a Session, even if the rest of the Session state is empty.
- The Client's subscriptions.
- QoS 1 and QoS 2 messages which have been sent to the Client, but have not been completely acknowledged.
- QoS 1 and QoS 2 messages pending transmission to the Client.
- QoS 2 messages which have been received from the Client, but have not been completely acknowledged.
- Optionally, QoS 0 messages pending transmission to the Client.

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

See Section 4.1 for details and limitations of stored state.

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

Non normative comment

To ensure consistent state in the event of a failure, the Client should repeat its attempts to connect with CleanSession set to 1, until it connects successfully.

Non normative comment

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

Non normative comment

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

3.1.2.5 Will Flag

Position: bit 2 of the Connect Flags.

If the Will Flag is set to 1 this indicates that, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Network Connection. The Will Message MUST be published when the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of a DISCONNECT Packet [MQTT-3.1.2-8].

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

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

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-3.1.2-9].

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 [MQTT-3.1.2-10].

If the Will Flag is set to 0 the Will QoS and Will Retain fields in the Connect Flags MUST be set to zero and the Will Topic and Will Message fields MUST NOT be present in the payload [MQTT-3.1.2-11].

If the Will Flag is set to 0, a Will Message MUST NOT be published when this Network Connection ends [MQTT-3.1.2-12].

The Server SHOULD publish Will Messages promptly. In the case of a Server shutdown or failure the server MAY defer publication of Will Messages until a subsequent restart. If this happens there might be a delay between the time the server experienced failure and a Will Message being published.

3.1.2.6 Will QoS

Position: bits 4 and 3 of the Connect Flags.

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

If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00) [MQTT-3.1.2-13].

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-3.1.2-14].

3.1.2.7 Will Retain

Position: bit 5 of the Connect Flags.

This bit specifies if the Will Message is to be Retained when it is published.

If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0 [MQTT-3.1.2-15].

If the Will Flag is set to 1:

- If Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message [MQTT-3.1.2-16].
- If Will Retain is set to 1, the Server MUST publish the Will Message as a retained message [MQTT-3.1.2-17].

3.1.2.8 User Name Flag

Position: bit 7 of the Connect Flags.

If the User Name Flag is set to 0, a user name MUST NOT be present in the payload [MQTT-3.1.2-18].

If the User Name Flag is set to 1, a user name MUST be present in the payload [MQTT-3.1.2-19].

3.1.2.9 Password Flag

Position: bit 6 of the Connect Flags byte.

If the Password Flag is set to 0, a password MUST NOT be present in the payload [MQTT-3.1.2-20].

If the Password Flag is set to 1, a password MUST be present in the payload [MQTT-3.1.2-21].

If the User Name Flag is set to 0, the Password Flag MUST be set to 0 [MQTT-3.1.2-22].

3.1.2.10 Keep Alive

Figure 3.5 Keep Alive bytes

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

The Keep Alive is a time interval measured in seconds. Expressed as a 16-bit word, it is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one Control Packet and the point it starts sending the next. 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-3.1.2-23].

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

If the Keep Alive value is non-zero and 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-3.1.2-24].

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

A Keep Alive value of zero (0) has the effect of turning off the keep alive mechanism. This means that, in this case, the Server is not required to disconnect the Client on the grounds of inactivity.

556 Note that a Server is permitted to disconnect a Client that it determines to be inactive or non-responsive
557 at any time, regardless of the Keep Alive value provided by that Client.
558
559 **Non normative comment**
560 The actual value of the Keep Alive is application specific; typically this is a few minutes. The
561 maximum value is 18 hours 12 minutes and 15 seconds.

562 ### 3.1.2.11 Variable header non normative example

563 **Figure 3.6 - Variable header non normative example**

	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) Will Flag (1) Clean Session (1) *Reserved* (0)	1	1	0	0	1	1	1	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

564

3.1.3 Payload

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

3.1.3.1 Client Identifier

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

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

The ClientId MUST be a UTF-8 encoded string as defined in Section 1.5.3 [MQTT-3.1.3-4].

The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters

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

The Server MAY allow ClientId's that contain more than 23 encoded bytes. The Server MAY allow ClientId's that contain characters not included in the list given above.

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-3.1.3-6].

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

If the Client supplies a zero-byte ClientId with CleanSession 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-3.1.3-8].

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-3.1.3-9].

Non normative comment

A Client implementation could provide a convenience method to generate a random ClientId. Use of such a method should be actively discouraged when the CleanSession is set to 0.

3.1.3.2 Will Topic

If the Will Flag is set to 1, the Will Topic is the next field in the payload. The Will Topic MUST be a UTF-8 encoded string as defined in Section 1.5.3 [MQTT-3.1.3-10].

3.1.3.3 Will Message

If the Will Flag is set to 1 the Will Message is the next field in the payload. The Will Message defines the Application Message that is to be published to the Will Topic as described in Section 3.1.2.5. This field consists of a two byte length followed by the payload for the Will Message expressed as a sequence of zero or more bytes. The length gives the number of bytes in the data that follows and does not include the 2 bytes taken up by the length itself.

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

3.1.3.4 User Name

If the User Name Flag is set to 1, this is the next field in the payload. The User Name MUST be a UTF-8 encoded string as defined in Section 1.5.3 [MQTT-3.1.3-11]. It can be used by the Server for authentication and authorization.

3.1.3.5 Password

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

Figure 3.7 - Password bytes

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.							

3.1.4 Response

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

1. If the Server does not receive a CONNECT Packet within a reasonable amount of time after the Network Connection is established, the Server SHOULD close the connection.

2. 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-3.1.4-1].

3. The Server MAY check that the contents of the CONNECT Packet meet any further restrictions and MAY perform authentication and authorization checks. If any of these checks fail, it SHOULD send an appropriate CONNACK response with a non-zero return code as described in section 3.2 and it MUST close the Network Connection.

If validation is successful the Server performs the following steps.

1. If the ClientId represents a Client already connected to the Server then the Server MUST disconnect the existing Client [MQTT-3.1.4-2].

2. The Server MUST perform the processing of CleanSession that is described in section 3.1.2.4 [MQTT-3.1.4-3].

3. The Server MUST acknowledge the CONNECT Packet with a CONNACK Packet containing a zero return code [MQTT-3.1.4-4].

4. Start message delivery and keep alive monitoring.

Clients are allowed to send further Control Packets immediately after sending a CONNECT Packet; Clients need not wait for a CONNACK Packet to arrive from the Server. If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT Packet [MQTT-3.1.4-5].

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

3.2 CONNACK – Acknowledge connection request

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

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

3.2.1 Fixed header

The fixed header format is illustrated in Figure 3.8 – CONNACK Packet fixed header.

Figure 3.8 – CONNACK Packet fixed header

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

Remaining Length field

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

3.2.2 Variable header

The variable header format is illustrated in Figure 3.9 – CONNACK Packet variable header.

Figure 3.9 – CONNACK Packet variable header

	Description	7	6	5	4	3	2	1	0
Connect Acknowledge Flags		Reserved							SP[1]
byte 1		0	0	0	0	0	0	0	X

Connect Return code									
byte 2		X	X	X	X	X	X	X	X

3.2.2.1 Connect Acknowledge Flags

Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0.

Bit 0 (SP[1]) is the Session Present Flag.

3.2.2.2 Session Present

Position: bit 0 of the Connect Acknowledge Flags.

If the Server accepts a connection with CleanSession set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a zero return code in the CONNACK packet [MQTT-3.2.2-1].

If the Server accepts a connection with CleanSession set to 0, the value set in Session Present depends on whether the Server already has stored Session state for the supplied client ID. If the Server has stored Session state, it MUST set Session Present to 1 in the CONNACK packet [MQTT-3.2.2-2]. If the Server does not have stored Session state, it MUST set Session Present to 0 in the CONNACK packet. This is in addition to setting a zero return code in the CONNACK packet [MQTT-3.2.2-3].

The Session Present flag enables a Client to establish whether the Client and Server have a consistent view about whether there is already stored Session state.

Once the initial setup of a Session is complete, a Client with stored Session state will expect the Server to maintain its stored Session state. In the event that the value of Session Present received by the Client from the Server is not as expected, the Client can choose whether to proceed with the Session or to disconnect. The Client can discard the Session state on both Client and Server by disconnecting, connecting with Clean Session set to 1 and then disconnecting again.

If a server sends a CONNACK packet containing a non-zero return code it MUST set Session Present to 0 [MQTT-3.2.2-4].

3.2.2.3 Connect Return code

Byte 2 in the Variable header.

The values for the one byte unsigned Connect Return code field are listed in Table 3.1 – Connect Return code values. If a well formed CONNECT Packet is received by the Server, but the Server is unable to process it for some reason, then the Server SHOULD attempt to send a CONNACK packet containing the appropriate non-zero Connect return code from this table. If a server sends a CONNACK packet containing a non-zero return code it MUST then close the Network Connection [MQTT-3.2.2-5].

Table 3.1 – Connect Return code values

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

If none of the return codes listed in Table 3.1 – Connect Return code values are deemed applicable, then the Server MUST close the Network Connection without sending a CONNACK [MQTT-3.2.2-6].

3.2.3 Payload

The CONNACK Packet has no payload.

3.3 PUBLISH – Publish message

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

3.3.1 Fixed header

Figure 3.10 – PUBLISH Packet fixed header illustrates the fixed header format:

Figure 3.10 – PUBLISH Packet fixed header

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							

3.3.1.1 DUP

Position: byte 1, bit 3.

If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted to send this MQTT PUBLISH Packet. If the DUP flag is set to 1, it indicates that this might be re-delivery of an earlier attempt to send the Packet.

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet [MQTT-3.3.1.-1]. The DUP flag MUST be set to 0 for all QoS 0 messages [MQTT-3.3.1-2].

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 is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission [MQTT-3.3.1-3].

Non normative comment

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

Non normative comment

It is important to note that the DUP flag refers to the Control Packet itself and not to the Application Message that it contains. When using QoS 1, it is possible for a Client to receive a PUBLISH Packet with DUP flag set to 0 that contains a repetition of an Application Message that it received earlier, but with a different Packet Identifier. Section 2.3.1 provides more information about Packet Identifiers.

3.3.1.2 QoS

Position: byte 1, bits 2-1.

This field indicates the level of assurance for delivery of an Application Message. The QoS levels are listed in the Table 3.2 - QoS definitions, below.

Table 3.2 - QoS definitions

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
-	1	1	Reserved – must not be used

A PUBLISH Packet MUST NOT have both QoS bits set to 1. If a Server or Client receives a PUBLISH Packet which has both QoS bits set to 1 it MUST close the Network Connection [MQTT-3.3.1-4].

3.3.1.3 RETAIN

Position: byte 1, bit 0.

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-3.3.1-5]. When a new subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber [MQTT-3.3.1-6]. 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 choose to discard it at any time - if this happens there will be no retained message for that topic [MQTT-3.3.1-7]. See Section 4.1 for more information on storing state.

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-3.3.1-8]. 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-3.3.1-9].

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-3.3.1-10]. “As normal” means that the RETAIN flag is

not set in the message received by existing Clients. A zero byte retained message MUST NOT be stored as a retained message on the Server [MQTT-3.3.1-11].

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-3.3.1-12].

Non normative comment

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

Remaining Length field

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

3.3.2 Variable header

The variable header contains the following fields in the order: Topic Name, Packet Identifier.

3.3.2.1 Topic Name

The Topic Name identifies the information channel to which payload data is published.

The Topic Name MUST be present as the first field in the PUBLISH Packet Variable header. It MUST be a UTF-8 encoded string [MQTT-3.3.2-1] as defined in section 1.5.3.

The Topic Name in the PUBLISH Packet MUST NOT contain wildcard characters [MQTT-3.3.2-2].

The Topic Name in a PUBLISH Packet sent by a Server to a subscribing Client MUST match the Subscription's Topic Filter according to the matching process defined in Section 4.7 [MQTT-3.3.2-3]. However, since the Server is permitted to override the Topic Name, it might not be the same as the Topic Name in the original PUBLISH Packet.

3.3.2.2 Packet Identifier

The Packet Identifier field is only present in PUBLISH Packets where the QoS level is 1 or 2. Section 2.3.1 provides more information about Packet Identifiers.

3.3.2.3 Variable header non normative example

Figure 3.11 - Publish Packet variable header non normative example illustrates an example variable header for the PUBLISH Packet briefly described in Table 3.3 - Publish Packet non normative example.

Table 3.3 - Publish Packet non normative example

Field	Value
Topic Name	a/b
Packet Identifier	10

Figure 3.11 - Publish Packet variable header non normative example

	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

3.3.3 Payload

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

3.3.4 Response

The receiver of a PUBLISH Packet MUST respond according to Table 3.4 - Expected Publish Packet response as determined by the QoS in the PUBLISH Packet [MQTT-3.3.4-1].

Table 3.4 - Expected Publish Packet response

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

3.3.5 Actions

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

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

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

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

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 close the Network Connection [MQTT-3.3.5-2].

3.4 PUBACK – Publish acknowledgement

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

3.4.1 Fixed header

Figure 3.12 - PUBACK Packet fixed header

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

Remaining Length field

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

3.4.2 Variable header

This contains the Packet Identifier from the PUBLISH Packet that is being acknowledged.

Figure 3.13 – PUBACK Packet variable header

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

3.4.3 Payload

The PUBACK Packet has no payload.

3.4.4 Actions

This is fully described in Section 4.3.2.

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

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

3.5.1 Fixed header

Figure 3.14 – PUBREC Packet fixed header

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

Remaining Length field

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

3.5.2 Variable header

The variable header contains the Packet Identifier from the PUBLISH Packet that is being acknowledged.

Figure 3.15 – PUBREC Packet variable header

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

3.5.3 Payload

The PUBREC Packet has no payload.

3.5.4 Actions

This is fully described in Section 4.3.3.

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

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

3.6.1 Fixed header

Figure 3.16 – PUBREL Packet fixed header

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

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-3.6.1-1].

Remaining Length field

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

3.6.2 Variable header

The variable header contains the same Packet Identifier as the PUBREC Packet that is being acknowledged.

Figure 3.17 – PUBREL Packet variable header

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

3.6.3 Payload

The PUBREL Packet has no payload.

3.6.4 Actions

This is fully described in Section 4.3.3.

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

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

3.7.1 Fixed header

Figure 3.18 – PUBCOMP Packet fixed header

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

Remaining Length field

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

3.7.2 Variable header

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

Figure 3.19 – PUBCOMP Packet variable header

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

3.7.3 Payload

The PUBCOMP Packet has no payload.

3.7.4 Actions

This is fully described in Section 4.3.3.

3.8 SUBSCRIBE - Subscribe to topics

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

3.8.1 Fixed header

Figure 3.20 – SUBSCRIBE Packet fixed header

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							

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-3.8.1-1].

Remaining Length field

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

3.8.2 Variable header

The variable header contains a Packet Identifier. Section 2.3.1 provides more information about Packet Identifiers.

3.8.2.1 Variable header non normative example

Figure 3.21 shows a variable header with Packet Identifier set to 10.

Figure 3.21 - Variable header with a Packet Identifier of 10, Non normative example

	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

3.8.3 Payload

The payload of a SUBSCRIBE Packet contains a list of Topic Filters indicating the Topics to which the Client wants to subscribe. The Topic Filters in a SUBSCRIBE packet payload MUST be UTF-8 encoded strings as defined in Section 1.5.3 [MQTT-3.8.3-1]. A Server SHOULD support Topic filters that contain the wildcard characters defined in Section 4.7.1. If it chooses not to support topic filters that contain wildcard characters it MUST reject any Subscription request whose filter contains them [MQTT-3.8.3-2]. Each filter is followed by a byte called the Requested QoS. This gives the maximum QoS level at which the Server can send Application Messages to the Client.

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-3.8.3-3]. See section 4.8 for information about handling errors.

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

Figure 3.22 – SUBSCRIBE Packet payload format

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

The upper 6 bits of the Requested QoS byte are not used in the current version of the protocol. They are reserved for future use. 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, or QoS is not 0,1 or 2 [MQTT-3-8.3-4].

3.8.3.1 Payload non normative example

Figure 3.23 - Payload byte format non normative example shows the payload for the SUBSCRIBE Packet briefly described in Table 3.5 - Payload non normative example.

Table 3.5 - Payload non normative example

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

Figure 3.23 - Payload byte format non normative example

	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 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
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 LSB (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

3.8.4 Response

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

The Server is permitted to start sending PUBLISH packets matching the Subscription before the Server sends the SUBACK Packet.

If a Server receives a SUBSCRIBE Packet containing a Topic Filter that is identical to an existing Subscription's Topic Filter then it MUST completely replace 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 messages matching the Topic Filter MUST be re-sent, but the flow of publications MUST NOT be interrupted [MQTT-3.8.4-3].

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

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-3.8.4-4].

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-3.8.4-5]. 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-3.8.4-6].

Non normative examples

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

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

Non normative comment

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

3.9 SUBACK – Subscribe acknowledgement

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

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

3.9.1 Fixed header

Figure 3.24 – SUBACK Packet fixed header

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							

Remaining Length field

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

3.9.2 Variable header

The variable header contains the Packet Identifier from the SUBSCRIBE Packet that is being acknowledged. Figure 3.25 - variable header format below illustrates the format of the variable header.

Figure 3.25 – SUBACK Packet variable header

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

3.9.3 Payload

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

Figure 3.26 - Payload format below illustrates the Return Code field encoded in a byte in the Payload.

Figure 3.26 – SUBACK Packet payload format

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

Allowed return codes:

0x00 - Success - Maximum QoS 0
0x01 - Success - Maximum QoS 1
0x02 - Success - Maximum QoS 2
0x80 - Failure

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

3.9.3.1 Payload non normative example

Figure 3.27 - Payload byte format non normative example shows the payload for the SUBACK Packet briefly described in Table 3.6 - Payload non normative example.

Table 3.6 - Payload non normative example

Success - Maximum QoS 0	0
Success - Maximum QoS 2	2
Failure	128

Figure 3.27 - Payload byte format non normative example

	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

3.10 UNSUBSCRIBE – Unsubscribe from topics

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

3.10.1 Fixed header

Figure 3.28 – UNSUBSCRIBE Packet Fixed header

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							

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

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

3.10.2 Variable header

The variable header contains a Packet Identifier. Section 2.3.1 provides more information about Packet Identifiers.

Figure 3.29 – UNSUBSCRIBE Packet variable header

Bit	7	6	5	4	3	2	1	0

byte 1	Packet Identifier MSB
byte 2	Packet Identifier LSB

3.10.3 Payload

The payload for the UNSUBSCRIBE Packet contains the list of Topic Filters that the Client wishes to unsubscribe from. The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 encoded strings as defined in Section 1.5.3, packed contiguously [MQTT-3.10.3-1].

The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter. An UNSUBSCRIBE packet with no payload is a protocol violation [MQTT-3.10.3-2]. See section 4.8 for information about handling errors.

3.10.3.1 Payload non normative example

Figure 3.30 - Payload byte format non normative example show the payload for the UNSUBSCRIBE Packet briefly described in Table3.7 - Payload non normative example.

Table3.7 - Payload non normative example

Topic Filter	“a/b”
Topic Filter	“c/d”

Figure 3.30 - Payload byte format non normative example

	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 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
Topic Filter									
byte 6	Length MSB (0)	0	0	0	0	0	0	0	0
byte 7	Length LSB (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

3.10.4 Response

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

occurs [MQTT-3.10.4-1].

If a Server deletes a Subscription:

- It MUST stop adding any new messages for delivery to the Client [MQTT-3.10.4-2].
- It MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client [MQTT-3.10.4-3].
- It MAY continue to deliver any existing messages buffered for delivery to the Client.

The Server MUST respond to an UNSUBSUBCRIBE request by sending an UNSUBACK packet. The UNSUBACK Packet MUST have the same Packet Identifier as the UNSUBSCRIBE Packet [MQTT-3.10.4-4]. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-3.10.4-5].

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-3.10.4-6].

3.11 UNSUBACK – Unsubscribe acknowledgement

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

3.11.1 Fixed header

Figure 3.31 – UNSUBACK Packet fixed header

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

Remaining Length field

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

3.11.2 Variable header

The variable header contains the Packet Identifier of the UNSUBSCRIBE Packet that is being acknowledged.

Figure 3.32 – UNSUBACK Packet variable header

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

3.11.3 Payload

The UNSUBACK Packet has no payload.

3.12 PINGREQ – PING request

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

1. Indicate to the Server that the Client is alive in the absence of any other Control Packets being sent from the Client to the Server.
2. Request that the Server responds to confirm that it is alive.
3. Exercise the network to indicate that the Network Connection is active.

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

3.12.1 Fixed header

Figure 3.33 – PINGREQ Packet fixed header

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

3.12.2 Variable header

The PINGREQ Packet has no variable header.

3.12.3 Payload

The PINGREQ Packet has no payload.

3.12.4 Response

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

3.13 PINGRESP – PING response

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

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

3.13.1 Fixed header

Figure 3.34 – PINGRESP Packet fixed header

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

1142

1143 ### 3.13.2 Variable header

1144 The PINGRESP Packet has no variable header.

1145 ### 3.13.3 Payload

1146 The PINGRESP Packet has no payload.

1147 ## 3.14 DISCONNECT – Disconnect notification

1148 The DISCONNECT Packet is the final Control Packet sent from the Client to the Server. It indicates that
1149 the Client is disconnecting cleanly.

1150 ### 3.14.1 Fixed header

1151 **Figure 3.35 – DISCONNECT Packet fixed header**

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

1152 The Server MUST validate that reserved bits are set to zero and disconnect the Client if they are not zero
1153 [MQTT-3.14.1-1].

1154 ### 3.14.2 Variable header

1155 The DISCONNECT Packet has no variable header.

1156 ### 3.14.3 Payload

1157 The DISCONNECT Packet has no payload.

1158 ### 3.14.4 Response

1159 After sending a DISCONNECT Packet the Client:

1160 • MUST close the Network Connection [MQTT-3.14.4-1].

1161 • MUST NOT send any more Control Packets on that Network Connection [MQTT-3.14.4-2].

1162

1163 On receipt of DISCONNECT the Server:

1164 • MUST discard any Will Message associated with the current connection without publishing it, as
1165 described in Section 3.1.2.5 [MQTT-3.14.4-3].

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

4 Operational behavior

4.1 Storing state

It is necessary for the Client and Server to store Session state in order to provide Quality of Service guarantees. The Client and Server MUST store Session state for the entire duration of the Session [MQTT-4.1.0-1]. A Session MUST last at least as long it has an active Network Connection [MQTT-4.1.0-2].

Retained messages do not form part of the Session state in the Server. The Server SHOULD retain such messages until deleted by a Client.

Non normative comment

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

Non normative comment

It is possible that hardware or software failures may result in loss or corruption of Session state stored by the Client or Server.

Non normative comment

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

Non normative comment

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

4.1.1 Non normative example

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

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

4.2 Network Connections

The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client.

Non normative comment

The transport protocol used to carry MQTT 3.1 was TCP/IP as defined in [RFC793]. TCP/IP can be used for MQTT 3.1.1. The following are also suitable:

- TLS [RFC5246]
- WebSocket [RFC6455]

Non normative comment

TCP ports 8883 and 1883 are registered with IANA for MQTT TLS and non TLS communication respectively.

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

4.3 Quality of Service levels and protocol flows

MQTT delivers Application Messages according to the Quality of Service (QoS) levels defined here. The delivery protocol is symmetric, in the description below the Client and Server can each take the role of either Sender or Receiver. The delivery protocol is concerned solely with the delivery of an application message from a single Sender to a single Receiver. When the Server is delivering an Application Message to more than one Client, each Client is treated independently. The QoS level used to deliver an Application Message outbound to the Client could differ from that of the inbound Application Message.

The non-normative flow diagrams in the following sections are intended to show possible implementation approaches.

4.3.1 QoS 0: At most once delivery

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

In the QoS 0 delivery protocol, the Sender

- MUST send a PUBLISH packet with QoS=0, DUP=0 [MQTT-4.3.1-1].

In the QoS 0 delivery protocol, the Receiver

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

Figure 4.1 – QoS 0 protocol flow diagram, non normative example

Sender Action	Control Packet	Receiver Action
PUBLISH QoS 0, DUP=0		
	---------->	
		Deliver Application Message to appropriate onward recipient(s)

4.3.2 QoS 1: At least once delivery

This quality of service ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH Packet has a Packet Identifier in its variable header and is acknowledged by a PUBACK Packet. Section 2.3.1 provides more information about Packet Identifiers.

In the QoS 1 delivery protocol, the Sender

- MUST assign an unused Packet Identifier each time it has a new Application Message to publish.
- MUST send a PUBLISH Packet containing this Packet Identifier with QoS=1, DUP=0.
- MUST treat the PUBLISH Packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver. See Section 4.4 for a discussion of unacknowledged messages.

[MQTT-4.3.2-1].

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

Note that a Sender is permitted to send further PUBLISH Packets with different Packet Identifiers while it is waiting to receive acknowledgements.

In the QoS 1 delivery protocol, the Receiver

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

[MQTT-4.3.2-2].

Figure 4.2 – QoS 1 protocol flow diagram, non normative example

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		

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

4.3.3 QoS 2: Exactly once delivery

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

A QoS 2 message has a Packet Identifier in its variable header. Section 2.3.1 provides more information about Packet Identifiers. The receiver of a QoS 2 PUBLISH Packet acknowledges receipt with a two-step acknowledgement process.

In the QoS 2 delivery protocol, the Sender

- MUST assign an unused Packet Identifier when it has a new Application Message to publish.
- MUST send a PUBLISH packet containing this Packet Identifier with QoS=2, DUP=0.
- MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver. See Section 4.4 for a discussion of unacknowledged messages.
- MUST send a PUBREL packet when it receives a PUBREC packet from the receiver. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet.
- MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver.
- MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet.

[MQTT-4.3.3-1].

The Packet Identifier becomes available for reuse once the Sender has received the PUBCOMP Packet.

Note that a Sender is permitted to send further PUBLISH Packets with different Packet Identifiers while it is waiting to receive acknowledgements.

In the QoS 2 delivery protocol, the Receiver

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

[MQTT-4.3.3-2].

Figure 4.3 – QoS 2 protocol flow diagram, non normative example

Sender Action	Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP 0 <Packet Identifier>		
	---------->	

		Method A, Store message or Method B, 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>		
	---------->	
		Method A, Initiate onward delivery of the Application Message[1] then discard message or Method B, Discard <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<----------	
Discard stored state		

[1] The receiver is not required to complete delivery of the Application Message before sending the PUBREC or PUBCOMP. When its original sender receives the PUBREC packet, ownership of the Application Message is transferred to the receiver.

Figure 4.3 shows that there are two methods by which QoS 2 can be handled by the receiver. They differ in the point within the flow at which the message is made available for onward delivery. The choice of Method A or Method B is implementation specific. As long as an implementation chooses exactly one of these approaches, this does not affect the guarantees of a QoS 2 flow.

4.4 Message delivery retry

When a Client reconnects with CleanSession set to 0, both the Client and Server MUST re-send any unacknowledged PUBLISH Packets (where QoS > 0) and PUBREL Packets using their original Packet Identifiers [MQTT-4.4.0-1]. This is the only circumstance where a Client or Server is REQUIRED to redeliver messages.

Non normative comment

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

4.5 Message receipt

When a Server takes ownership of an incoming Application Message it MUST add it to the Session state of those clients that have matching Subscriptions. Matching rules are defined in Section 4.7 [MQTT-4.5.0-1].

Under normal circumstances Clients receive messages in response to Subscriptions they have created. A Client could also receive messages that do not match any of its explicit Subscriptions. This can happen if the Server automatically assigned a subscription to the Client. A Client could also receive messages while an UNSUBSCRIBE operation is in progress. The Client MUST acknowledge any Publish Packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains [MQTT-4.5.0-2].

4.6 Message ordering

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

- When it re-sends any PUBLISH packets, it MUST re-send them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages) [MQTT-4.6.0-1]
- It MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages) [MQTT-4.6.0-2]
- It MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages) [MQTT-4.6.0-3]
- It MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages) [MQTT-4.6.0-4]

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-4.6.0-5].

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-4.6.0-6].

Non normative comment

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

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

4.7 Topic Names and Topic Filters

4.7.1 Topic wildcards

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

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

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

4.7.1.1 Topic level separator

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

4.7.1.2 Multi-level wildcard

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

Non normative comment

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

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

Non normative comment

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

4.7.1.3 Single level wildcard

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

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

Non normative comment

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

Non normative comment

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

4.7.2 Topics beginning with $

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

Non normative comment

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

Non normative comment

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

4.7.3 Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

- All Topic Names and Topic Filters MUST be at least one character long [MQTT-4.7.3-1]
- Topic Names and Topic Filters are case sensitive
- Topic Names and Topic Filters can include the space character
- A leading ‘/’ creates a distinct Topic Name or Topic Filter
- A Topic Name or Topic Filter consisting only of the ‘/’ character is valid
- Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode] [MQTT-4.7.3-2]
- Topic Names and Topic Filters are UTF-8 encoded strings, they MUST NOT encode to more than 65535 bytes [MQTT-4.7.3-3]. See Section 1.5.3

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

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

Non normative comment

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

Non normative comment

- "ACCOUNTS" and "Accounts" are two different topic names
- "Accounts payable" is a valid topic name
- "/finance" is different from "finance"

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

4.8 Handling errors

Unless stated otherwise, if either the Server or Client encounters a protocol violation, it MUST close the Network Connection on which it received that Control Packet which caused the protocol violation [MQTT-4.8.0-1].

A Client or Server implementation might encounter a Transient Error (for example an internal buffer full condition) that prevents successful processing of an MQTT packet.

If the Client or Server encounters a Transient Error while processing an inbound Control Packet it MUST close the Network Connection on which it received that Control Packet [MQTT-4.8.0-2]. If a Server detects a Transient Error it SHOULD NOT disconnect or have any other effect on its interactions with any other Client.

5 Security

5.1 Introduction

This Chapter is provided for guidance only and is **Non Normative**. However, it is strongly recommended that Server implementations that offer TLS [RFC5246] SHOULD use TCP port 8883 (IANA service name: secure-mqtt).

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

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

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

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

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

In addition to technical security issues there could also be geographic (e.g. U.S.-EU SafeHarbor [USEUSAFEHARB]), industry specific (e.g. PCI DSS [PCIDSS]) and regulatory considerations (e.g. Sarbanes-Oxley [SARBANES]).

5.2 MQTT solutions: security and certification

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

Guidance on using MQTT within the NIST Cyber Security Framework [NISTCSF] can be found in the MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure Cybersecurity [MQTT NIST]. The use of industry proven, independently verified and certified technologies will help meet compliance requirements.

5.3 Lightweight cryptography and constrained devices

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

ISO 29192 [ISO29192] makes recommendations for cryptographic primitives specifically tuned to perform on constrained “low end” devices.

5.4 Implementation notes

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

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

5.4.1 Authentication of Clients by the Server

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

Implementations passing authentication data in clear text, obfuscating such data elements or requiring no authentication data should be aware this can give rise to Man-in-the-Middle and replay attacks. Section 5.4.5 introduces approaches to ensure data privacy.

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

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

An implementation might allow for authentication where the credentials are sent in an Application Message from the Client to the Server.

5.4.2 Authorization of Clients by the Server

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

5.4.3 Authentication of the Server by the Client

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

Where TLS [RFC5246] is used, SSL Certificates sent from the Server can be used by the Client to authenticate the Server. Implementations providing MQTT service for multiple hostnames from a single IP address should be aware of the Server Name Indication extension to TLS defined in section 3 of RFC

6066 [RFC6066].This allows a Client to tell the Server the hostname of the Server it is trying to connect to.

An implementation might allow for authentication where the credentials are sent in an Application Message from the Server to the Client.

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

5.4.4 Integrity of Application Messages and Control Packets

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

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

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

5.4.5 Privacy of Application Messages and Control Packets

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

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

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

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

5.4.6 Non-repudiation of message transmission

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

5.4.7 Detecting compromise of Clients and Servers

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

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

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

5.4.8 Detecting abnormal behaviors

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

- Repeated connection attempts
- Repeated authentication attempts
- Abnormal termination of connections
- Topic scanning (attempts to send or subscribe to many topics)
- Sending undeliverable messages (no subscribers to the topics)
- Clients that connect but do not send data

Server implementations might disconnect Clients that breach its security rules.

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

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

5.4.9 Other security considerations

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

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

In the case of long lasting connections:

- Client and Server implementations using TLS [RFC5246] should allow for session renegotiation to establish new cryptographic parameters (replace session keys, change cipher suites, change authentication credentials).
- Servers may disconnect Clients and require them to re-authenticate with new credentials.

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

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

5.4.10 Use of SOCKS

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

5.4.11 Security profiles

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

5.4.11.1 Clear communication profile

When using the clear communication profile, the MQTT protocol runs over an open network with no additional secure communication mechanisms in place.

5.4.11.2 Secured network communication profile

When using the secured network communication profile, the MQTT protocol runs over a physical or virtual network which has security controls e.g., VPNs or physically secure network.

5.4.11.3 Secured transport profile

When using the secured transport profile, the MQTT protocol runs over a physical or virtual network and using TLS [RFC5246] which provides authentication, integrity and privacy.

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

5.4.11.4 Industry specific security profiles

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

[NISTCSF] NIST Cyber Security Framework

[NIST7628] NISTIR 7628 Guidelines for Smart Grid Cyber Security

[FIPS1402] Security Requirements for Cryptographic Modules (FIPS PUB 140-2)

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

[NSAB] NSA Suite B Cryptography

6 Using WebSocket as a network transport

If MQTT is transported over a WebSocket [RFC6455] connection, the following conditions apply:

- MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection [MQTT-6.0.0-1].
- A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries [MQTT-6.0.0-2].
- The client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers [MQTT-6.0.0-3].
- The WebSocket Sub Protocol name selected and returned by the server MUST be “mqtt” [MQTT-6.0.0-4].
- The WebSocket URI used to connect the client and server has no impact on the MQTT protocol.

6.1 IANA Considerations

This specification requests IANA to register the WebSocket MQTT sub-protocol under the “WebSocket Subprotocol Name” registry with the following data:

Figure 6.1 - IANA WebSocket Identifier

Subprotocol Identifier	mqtt
Subprotocol Common Name	mqtt
Subprotocol Definition	http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

7 Conformance

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

An MQTT implementation MAY conform as both an MQTT Client and MQTT Server implementation. A Server that both accepts inbound connections and establishes outbound connections to other Servers MUST conform as both an MQTT Client and MQTT Server [MQTT-7.0.0-1].

Conformant implementations MUST NOT require the use of any extensions defined outside of this specification in order to interoperate with any other conformant implementation [MQTT-7.0.0-2].

7.1 Conformance Targets

7.1.1 MQTT Server

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

1. The format of all Control Packets that the Server sends matches the format described in Chapter 2 and Chapter 3.

2. It follows the Topic matching rules described in Section 4.7.

3. It satisfies all of the MUST level requirements in the following chapters that are identified except for those that only apply to the Client:

    - Chapter 1 - Introduction
    - Chapter 2 - MQTT Control Packet format
    - Chapter 3 - MQTT Control Packets
    - Chapter 4 - Operational behavior
    - Chapter 6 - (if MQTT is transported over a WebSocket connection)
    - Chapter 7 - Conformance Targets

A conformant Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client [MQTT-7.1.1-1]. However conformance does not depend on it supporting any specific transport protocols. A Server MAY support any of the transport protocols listed in Section 4.2, or any other transport protocol that meets the requirements of [MQTT-7.1.1-1].

7.1.2 MQTT Client

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

1. The format of all Control Packets that the Client sends matches the format described in Chapter 2 and Chapter 3.

2. It satisfies all of the MUST level requirements in the following chapters that are identified except for those that only apply to the Server:

    - Chapter 1 - Introduction
    - Chapter 2 - MQTT Control Packet format
    - Chapter 3 - MQTT Control Packets
    - Chapter 4 - Operational behavior
    - Chapter 6 - (if MQTT is transported over a WebSocket connection)

A conformant Client MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client [MQTT-7.1.2-1]. However conformance does not depend on it supporting any specific transport protocols. A Client MAY support any of the transport protocols listed in Section 4.2, or any other transport protocol that meets the requirements of [MQTT-7.1.2-1].

Appendix A. Acknowledgements (non normative)

The TC owes special thanks to Dr Andy Stanford-Clark and Arlen Nipper as the original inventors of the MQTT protocol and for their continued support with the standardization process.

The following individuals were members of the OASIS Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

- Sanjay Aiyagari (VMware, Inc.)
- Ben Bakowski (IBM)
- Andrew Banks (IBM)
- Arthur Barr (IBM)
- William Bathurst (Machine-to-Machine Intelligence (M2MI) Corporation)
- Ken Borgendale (IBM)
- Geoff Brown (Machine-to-Machine Intelligence (M2MI) Corporation)
- James Butler (Cimetrics Inc.)
- Marco Carrer (Eurotech S.p.A.)
- Raphael Cohn (Individual)
- Sarah Cooper (Machine-to-Machine Intelligence (M2MI) Corporation)
- Richard Coppen (IBM)
- AJ Dalola (Telit Communications S.p.A.)
- Mark Darbyshire (TIBCO Software Inc.)
- Scott deDeugd (IBM)
- Paul Duffy (Cisco Systems)
- Phili DesAutels (LogMeIn Inc.)
- John Fallows (Kaazing)
- Pradeep Fernando (WSO2)
- Paul Fremantle (WSO2)
- Thomas Glover (Cognizant Technology Solutions)
- Rahul Gupta (IBM)
- Steve Huston (Individual)
- Wes Johnson (Eurotech S.p.A.)
- Christopher Kelley (Cisco Systems)
- David Kemper (TIBCO Software Inc.)
- James Kirkland (Red Hat)
- Alex Kritikos (Software AG, Inc.)
- Louis-P. Lamoureux (Machine-to-Machine Intelligence (M2MI) Corporation)
- David Locke (IBM)
- Shawn McAllister (Solace Systems)
- Dale Moberg (Axway Software)
- Manu Namboodiri (Machine-to-Machine Intelligence (M2MI) Corporation)

- Peter Niblett (IBM)
- Arlen Nipper (Individual)
- Julien Niset (Machine-to-Machine Intelligence (M2MI) Corporation)
- Mark Nixon (Emerson Process Management)
- Nicholas O'Leary (IBM)
- Sandor Palfy (LogMeIn Inc.)
- Dominik Obermaier (dc-square GmbH)
- Pavan Reddy (Cisco Systems)
- Andrew Schofield (IBM)
- Wadih Shaib (BlackBerry)
- Ian Skerrett (Eclipse Foundation)
- Joe Speed (IBM)
- Allan Stockdill-Mander (IBM)
- Gary Stuebing (Cisco Systems)
- Steve Upton (IBM)
- James Wert jr. (Telit Communications S.p.A.)
- T. Wyatt (Individual)
- SHAWN XIE (Machine-to-Machine Intelligence (M2MI) Corporation)
- Dominik Zajac (dc-square GmbH)
- Ed Briggs (Microsoft)

Secretary:

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

Appendix B. Mandatory normative statements (non normative)

This Appendix is non-normative and is provided as a convenient summary of the numbered conformance statements found in the main body of this document. See Chapter 7 for a definitive list of conformance requirements.

Normative Statement Number	Normative Statement
[MQTT-1.5.3-1]	The character data in a UTF-8 encoded string MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular this data MUST NOT include encodings of code points between U+D800 and U+DFFF. If a Server or Client receives a Control Packet containing ill-formed UTF-8 it MUST close the Network Connection.
[MQTT-1.5.3-2]	A 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-1.5.3-3]	A 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-2.2.2-1]	Where a flag bit is marked as “Reserved” in Table 2.2 - Flag Bits, it is reserved for future use and MUST be set to the value listed in that table.
[MQTT-2.2.2-2]	If invalid flags are received, the receiver MUST close the Network Connection.
[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 re-sends a particular Control Packet, then it MUST use the same Packet Identifier in subsequent re-sends 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 QO2 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 or PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH Packet that was originally sent.
[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 Packet sent 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]	If CleanSession is set to 0, the Server MUST resume communications with the Client based on state from the current Session (as identified by the Client identifier). If there is no Session associated with the Client identifier the Server MUST create a new Session. The Client and Server MUST store the Session after the Client and Server are disconnected.
[MQTT-3.1.2-5]	After the disconnection of a Session that had CleanSession set to 0, 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 CleanSession is 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 messages 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, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Network Connection. The Will Message MUST be published when the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of 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.
[MQTT-3.1.2-11]	If the Will Flag is set to 0 the Will QoS and Will Retain fields in the Connect Flags MUST be set to zero and the Will Topic and Will Message fields MUST NOT be present in the payload.
[MQTT-3.1.2-12]	If the Will Flag is set to 0, a Will Message MUST NOT be published when this Network Connection ends.
[MQTT-3.1.2-13]	If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00).
[MQTT-3.1.2-14]	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-3.1.2-15]	If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0.

[MQTT-3.1.2-16]	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 message.
[MQTT-3.1.2-17]	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 message.
[MQTT-3.1.2-18]	If the User Name Flag is set to 0, a user name MUST NOT be present in the payload.
[MQTT-3.1.2-19]	If the User Name Flag is set to 1, a user name MUST be present in the payload.
[MQTT-3.1.2-20]	If the Password Flag is set to 0, a password MUST NOT be present in the payload.
[MQTT-3.1.2-21]	If the Password Flag is set to 1, a password MUST be present in the payload.
[MQTT-3.1.2-22]	If the User Name Flag is set to 0, the Password Flag MUST be set to 0.
[MQTT-3.1.2-23]	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-3.1.2-24]	If the Keep Alive value is non-zero and 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]	Each Client connecting to the Server has a unique ClientId. The ClientId MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session 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 CONNECT packet payload.
[MQTT-3.1.3-4]	The ClientId MUST be a UTF-8 encoded string as defined in Section 1.5.3.
[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 "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".
[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 CleanSession to 1.
[MQTT-3.1.3-8]	If the Client supplies a zero-byte ClientId with CleanSession 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-3.1.3-10]	The Will Topic MUST be a UTF-8 encoded string as defined in Section 1.5.3.
[MQTT-3.1.3-11]	The User Name MUST be a UTF-8 encoded string as defined in Section 1.5.3.
[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 CONNECT validation is successful the Server MUST perform the processing of CleanSession that is described in section 3.1.2.4.
[MQTT-3.1.4-4]	If CONNECT validation is successful the Server MUST acknowledge the CONNECT Packet with a CONNACK Packet containing a zero return code.
[MQTT-3.1.4-5]	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 the Server accepts a connection with CleanSession set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a zero return code in the CONNACK packet.
[MQTT-3.2.2-2]	If the Server accepts a connection with CleanSession set to 0, the value set in Session Present depends on whether the Server already has stored Session state for the supplied client ID. If the Server has stored Session state, it MUST set Session Present to 1 in the CONNACK packet.
[MQTT-3.2.2-3]	If the Server does not have stored Session state, it MUST set Session Present to 0 in the CONNACK packet. This is in addition to setting a zero return code in the CONNACK packet.
[MQTT-3.2.2-4]	If a server sends a CONNACK packet containing a non-zero return code it MUST set Session Present to 0.
[MQTT-3.2.2-5]	If a server sends a CONNACK packet containing a non-zero return code it MUST then close the Network Connection.
[MQTT-3.2.2-6]	If none of the return codes listed in Table 3.1 – Connect Return code values are deemed applicable, then the Server MUST close the Network Connection without sending a CONNACK.
[MQTT-3.3.1-1]	The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet.
[MQTT-3.3.1-2]	The DUP flag MUST be set to 0 for all QoS 0 messages.
[MQTT-3.3.1-3]	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 is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission.
[MQTT-3.3.1-4]	A PUBLISH Packet MUST NOT have both QoS bits set to 1. If a Server or Client receives a PUBLISH Packet which has both QoS bits set to 1 it MUST close the Network Connection.

[MQTT-3.3.1-5]	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-3.3.1-6]	When a new subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber.
[MQTT-3.3.1-7]	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 choose to discard it at any time - if this happens there will be no retained message for that topic.
[MQTT-3.3.1-8]	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-3.3.1-9]	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-3.3.1-10]	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-3.3.1-11]	A zero byte retained message MUST NOT be stored as a retained message on the Server.
[MQTT-3.3.1-12]	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-3.3.2-1]	The Topic Name MUST be present as the first field in the PUBLISH Packet Variable header. It 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 in a PUBLISH Packet sent by a Server to a subscribing Client MUST match the Subscription's Topic Filter according to the matching process defined in Section 4.7.
[MQTT-3.3.4-1]	The receiver of a PUBLISH Packet MUST respond according to Table 3.4 - Expected Publish Packet response as determined by the QoS in the PUBLISH Packet.
[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 close the Network Connection.
[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-3.8.1-1]	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 Topic Filters in a SUBSCRIBE packet payload MUST be UTF-8 encoded strings as defined in Section 1.5.3.
[MQTT-3.8.3-2]	If the Server chooses not to support topic filters that contain wildcard characters it MUST reject any Subscription request whose filter contains them.
[MQTT-3.8.3-3]	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-3-8.3-4]	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, or QoS is not 0,1 or 2.
[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 that it is acknowledging.
[MQTT-3.8.4-3]	If a Server receives a SUBSCRIBE Packet containing a Topic Filter that is identical to an existing Subscription's Topic Filter then it MUST completely replace 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 messages matching the Topic Filter MUST be re-sent, but the flow of publications MUST NOT be 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 Filters in an UNSUBSCRIBE packet MUST be UTF-8 encoded strings as defined in Section 1.5.3, packed contiguously.

[MQTT-3.10.3-2]	The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter. An UNSUBSCRIBE packet with no payload is a protocol violation.
[MQTT-3.10.4-1]	The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription is deleted, otherwise no additional processing occurs.
[MQTT-3.10.4-2]	If a Server deletes a Subscription It MUST stop adding any new messages for delivery to the Client.
[MQTT-3.10.4-3]	If a Server deletes a Subscription It MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client.
[MQTT-3.10.4-4]	The Server MUST respond to an UNSUBSUBCRIBE request by sending an UNSUBACK packet. The UNSUBACK Packet MUST have the same Packet Identifier as the UNSUBSCRIBE Packet.
[MQTT-3.10.4-5]	Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK.
[MQTT-3.10.4-6]	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 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 any Will Message associated with the current connection without publishing it, as described in Section 3.1.2.5.
[MQTT-4.1.0-1]	The Client and Server MUST store Session state for the entire duration of the Session.
[MQTT-4.1.0-2]	A Session MUST last at least as long it has an active Network Connection.
[MQTT-4.3.1-1]	In the QoS 0 delivery protocol, the Sender • MUST send a PUBLISH packet with QoS=0, DUP=0.
[MQTT-4.3.2-1]	In the QoS 1 delivery protocol, the Sender • MUST assign an unused Packet Identifier each time it has a new Application Message to publish. • MUST send a PUBLISH Packet containing this Packet Identifier with QoS=1, DUP=0. • MUST treat the PUBLISH Packet as "unacknowledged" until it has received the corresponding PUBACK packet from the receiver. See Section 4.4 for a discussion of unacknowledged messages.
[MQTT-4.3.2-2]	In the QoS 1 delivery protocol, the Receiver

	• MUST respond with a PUBACK Packet containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message. • After it has sent a PUBACK Packet the Receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new publication, irrespective of the setting of its DUP flag.
[MQTT-4.3.3-1]	In the QoS 2 delivery protocol, the Sender • MUST assign an unused Packet Identifier when it has a new Application Message to publish. • MUST send a PUBLISH packet containing this Packet Identifier with QoS=2, DUP=0. • MUST treat the PUBLISH packet as "unacknowledged" until it has received the corresponding PUBREC packet from the receiver. See Section 4.4 for a discussion of unacknowledged messages. • MUST send a PUBREL packet when it receives a PUBREC packet from the receiver. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet. • MUST treat the PUBREL packet as "unacknowledged" until it has received the corresponding PUBCOMP packet from the receiver. • MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet.
[MQTT-4.3.3-2]	In the QoS 2 delivery protocol, the Receiver • MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message. • Until it has received the corresponding PUBREL packet, the Receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case. • MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL. • After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication.
[MQTT-4.4.0-1]	When a Client reconnects with CleanSession set to 0, both the Client and Server MUST re-send any unacknowledged PUBLISH Packets (where QoS > 0) and PUBREL Packets using their original Packet Identifiers.
[MQTT-4.5.0-1]	When a Server takes ownership of an incoming Application Message it MUST add it to the Session state of those clients that have matching Subscriptions. Matching rules are defined in Section 4.7.
[MQTT-4.5.0-2]	The Client MUST acknowledge any Publish Packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains.
[MQTT-4.6.0-1]	When it re-sends any PUBLISH packets, it MUST re-send 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-4.7.2-1]	The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character.
[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-4.7.3-4]	When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters.
[MQTT-4.8.0-1]	Unless stated otherwise, if either the Server or Client encounters a protocol violation, it MUST close the Network Connection on which it received that Control Packet which caused the protocol violation.
[MQTT-4.8.0-2]	If the Client or Server encounters a Transient Error while processing an inbound Control Packet it MUST close the Network Connection on which it received that Control Packet.
[MQTT-6.0.0-1]	MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection.
[MQTT-6.0.0-2]	A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries.
[MQTT-6.0.0-3]	The client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers.
[MQTT-6.0.0-4]	The WebSocket Sub Protocol name selected and returned by the server MUST

	be “mqtt”.
[MQTT-7.0.0-1]	A Server that both accepts inbound connections and establishes outbound connections to other Servers MUST conform as both an MQTT Client and MQTT Server.
[MQTT-7.0.0-2]	Conformant implementations MUST NOT require the use of any extensions defined outside of this specification in order to interoperate with any other conformant implementation.
[MQTT-7.1.1-1]	A conformant Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client.
[MQTT-7.1.2-1]	A conformant Client MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client.

1814

Appendix C. Revision history (non normative)

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]	[20th 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
[CSD04]	[28 January 2014]	[Rahul Gupta]	Resolved Issues – 112, 114, 115, 120, 117, 134, 132, 133, 130, 131, 129
[18]	[20 February 2014]	[A Banks] [Rahul Gupta]	Resolved Issues – 175, 139, 176, 166, 149, 164, 140, 154, 178, 188, 181, 155, 170, 196, 173, 157, 195, 191, 150, 179, 185, 174, 163 Resolved Issues – 135, 136, 147, 161, 169, 180, 182, 184, 189, 187
[19]	[28 February 2014]	[A Banks] [Rahul Gupta]	Resolved Issues – 167, 192, 141, 138, 137, 198, 165 Resolved Issues – 199, 144, 159,
[20]	[07 March 2014]	[A Banks] [Rahul Gupta]	Resolved Issues – 113, 162, 158, 146 Resolved Issues – 172, 190, 202, 201
[21]	[17 March 2014]	[A Banks] [Rahul Gupta]	Resolved Issues – 151, 194, 160, 168 Resolved Issues – 205,
[22]	[27 March 2014]	[Rahul Gupta] [A Banks]	Resolved Issues – 145, 186, 142 Resolved Issues – 152, 193
[23]	[28 March 2014]	[A Banks]	Resolved Issues – 204, 148, 210, 208, 209, 171, 183, 117, 212
[24]	[7 April 2014]	[Rahul Gupta] [A Banks]	Added Table of figures Corrected Issue 209
[25]	[8 May 2014]	[Rahul Gupta]	Resolved Issues – 213, 214
[25]	[3 September 2014]	[A Banks]	Resolved Issues – 240, 242, 246
[26]	[17 September 2014]	[Rahul Gupta]	Resolved Issues – 247
[27]	[18 November 2015]	[Rahul Gupta] [A Banks]	Updated with Errata 01 for Resolved Issue - 275

1817