MQTT Version 5.0

Working Draft 11

23rd February 2017

Technical Committee:
OASIS Message Queuing Telemetry Transport (MQTT) TC

Chairs:
Brian Raymor (brian.raymor@microsoft.com), Microsoft
Richard Coppen (coppen@uk.ibm.com), IBM

Editors:
Andrew Banks (andrew_banks@uk.ibm.com), IBM
Ed Briggs (edbriggs@microsoft.com), Microsoft
Ken Borgendale (kwb@us.ibm.com), IBM
Rahul Gupta (rahul.gupta@us.ibm.com), IBM

Additional artifacts:

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 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 messages 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 Working Draft (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or approved as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document Approval Process begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

URI patterns:

Initial publication URI:
http://docs.oasis-open.org/mqtt/mqtt/v5.0/csd01/mqtt-v5.0-csd01.docx
Permanent “Latest version” URI:
http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.docx

(Managed by OASIS TC Administration; please don’t modify.)

Table of Contents

1 Introduction

1.1 Organization of MQTT

The specification is split into seven chapters:

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.

Refer to section 4.2 Network Connection for Non-Normative examples.

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 (QoS), Properties, 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.
- Close the Network Connection to 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.
- Closes the Network Connection from the Client.

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.

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.

Shared Subscription:

A Shared Subscription comprises a Topic Filter and a maximum QoS. A Shared Subscription can be associated with more than one Session. An Application Message that matches a Shared Subscription is only sent to the Client associated with one of these Sessions. A Session can subscribe to more than one Shared Subscription and can contain both Shared Subscriptions and Non-Shared Subscriptions.

Wildcard Subscription:

A Wildcard Subscription is a Subscription with a Topic Filter containing one or more wildcard characters. This allows the subscription to receive Application Messages published to multiple Topic Names. Refer to section 4.7 for a description of wildcard characters in a 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.

MQTT Control Packet:

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

Malformed Packet:

A control packet that cannot be parsed according to this specification. See section 4.13 for information about error handling.

Protocol Error:

An error that is detected after the packet has been parsed and found to contain data that is not allowed by the protocol or is inconsistent with the state of the client or server. See section 4.13 for information about error handling.

1.3 Normative references

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,

http://www.rfc-editor.org/info/rfc2119

[RFC3629]

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

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

[RFC5246]

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008,

http://www.rfc-editor.org/info/rfc5246

[RFC6455]

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

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

[Unicode]

The Unicode Consortium. The Unicode Standard,

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

1.4 Non-Normative references

[RFC0793]

Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, http://www.rfc-editor.org/info/rfc793

[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

[MQTTV311]

MQTT V3.1.1 Protocol Specification

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.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, DOI 10.17487/RFC1928, March 1996,

http://www.rfc-editor.org/info/rfc1928

[RFC4511]

Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006,

http://www.rfc-editor.org/info/rfc4511

[RFC5077]

Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008,

http://www.rfc-editor.org/info/rfc5077

[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, DOI 10.17487/RFC5280, May 2008,

http://www.rfc-editor.org/info/rfc5280

[RFC6066]

Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011,

http://www.rfc-editor.org/info/rfc6066

[RFC6749]

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012,

http://www.rfc-editor.org/info/rfc6749

[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, DOI 10.17487/RFC6960, June 2013,

http://www.rfc-editor.org/info/rfc6960

[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

[RFC3986]

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,

http://www.rfc-editor.org/info/rfc3986

[RFC1035]

Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987,

http://www.rfc-editor.org/info/rfc1035

[RFC2782]

Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000,

http://www.rfc-editor.org/info/rfc2782

1.5 Data representation

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 Two Byte Integer

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

1.5.3 Four Byte Integer

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

1.5.4 UTF-8 Encoded String

Text fields in the MQTT 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 Integer length field that gives the number of bytes in a UTF-8 encoded string itself, as illustrated in Figure 1.1 Structure of UTF-8 encoded strings below. Consequently, there is a limit on the size of a string that can be passed in one of these UTF-8 encoded string components; one 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 an MQTT Control Packet containing ill-formed UTF-8 it MUST close the Network Connection. [MQTT-1.5.4-1]

A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000. If a receiver (Server or Client) receives an MQTT Control Packet containing U+0000 it MUST close the Network Connection. [MQTT-1.5.4-2]

The data SHOULD NOT include encodings of the Unicode [Unicode] code points listed below. If a receiver (Server or Client) receives an MQTT Control Packet containing any of them it MAY 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-1.5.4-3]

Non-Normative example

For example, the string A   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

1.5.5 Variable Byte Integer

The Variable Byte Integer is encoded using an 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

Variable Byte Integer field is four. The encoded value MUST use the minimum number of bytes necessary to represent the value. [MQTT-1.5.5-1]

Table 1-1 Size of Variable Byte Integer

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 Byte Integer 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 a Variable Byte Integer type 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 Variable Byte Integer)
    multiplier *= 128
while ((encodedByte AND 128) != 0)
```

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

When this algorithm terminates, value contains the Variable Byte Integer value.

1.5.6 Binary Data

Binary Data is represented by a Two Byte Integer length which indicates the number of data bytes, followed by that number of bytes. Thus, the length of Binary Data is limited to the range of 0 to 65535 Bytes. Where used the data consists only of the data portion of the field, which can take any value and does not include first two length bytes.

1.5.7 UTF-8 String Pair

A UTF-8 String pair consists of two UTF-8 Encoded Strings. This data type is used to hold name-value pairs. The first string serves as the name, and the second string contains the value.

Both strings MUST comply with the requirements for UTF-8 Encoded Strings. If a receiver (Client or Server) receives a string pair which does not meet these requirements, it MUST close the Network Connection. [MQTT-1.5.7-1]

1.6 Security

MQTT Client and Server implementations SHOULD offer Authentication, Authorization and secure communications options, such as those discussed in Chapter 5. Applications concerned with critical infrastructure, personally identifiable information, or other personal or sensitive information are strongly advised to use these security capabilities.

1.7 Editing convention

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

2.1.1 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							

2.1.2 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 MQTT Control Packet types

Table 2-1 MQTT Control Packet types

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

PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS delivery part 3)
SUBSCRIBE	8	Client to Server	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 or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

2.1.3 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-2.1.3-1]. If invalid flags are received, the receiver MUST close the Network Connection [MQTT-2.1.3-2]. Refer to section 0 for details about handling errors.

Table 2-2 Flag Bits

MQTT 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 5.0	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
AUTH	Reserved	0	0	0	0

DUP[1] = Duplicate delivery of a PUBLISH packet
QoS[2] = PUBLISH Quality of Service
RETAIN[3] = PUBLISH RETAIN flag
Refer to section 3.3.1 for a description of the DUP, QoS, and RETAIN flags in the PUBLISH packet.

2.1.4 Remaining Length

Position: starts at byte 2.

The Remaining Length is a Variable Byte Integer that represents 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 packet size is the total number of bytes in an MQTT Control Packet, this is equal to the length of the Fixed Header plus the Remaining Length.

2.2 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.

2.2.1 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 MQTT Control Packet types includes a Two Byte Integer Packet Identifier field. These MQTT Control Packets are PUBLISH (where QoS > 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) MQTT Control Packets MUST contain a non-zero Packet Identifier. Each time a Client sends a new packet of one of these types it MUST assign it a currently unused Packet Identifier [MQTT-2.2.1-1]. If a Client re-sends a particular MQTT Control Packet, then it MUST use the same Packet Identifier in subsequent re-sends of that packet [MQTT-2.2.1-2]. 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. The same conditions apply to a Server when it sends a PUBLISH with QoS > 0.

Packet Identifiers used with PUBLISH, SUBSCRIBE and UNSUBSCRIBE commands form a single, unified set of identifiers. A packet Identifier cannot be used by more than one command at any time.

A PUBLISH packet MUST NOT contain a Packet Identifier if its QoS value is set to 0. [MQTT-2.2.1-3]

A PUBACK, PUBREC or PUBREL packet MUST contain the same Packet Identifier as the PUBLISH packet that was originally sent [MQTT-2.2.1-4]. Similarly, SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE packet respectively [MQTT-2.2.1-5].

MQTT Control Packets that require a Packet Identifier are listed in Table 2.3 – MQTT Control Packets that contain a Packet Identifier.

Table 2-3 MQTT Control Packets that contain a Packet Identifier

MQTT 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
AUTH	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 packet with Packet Identifier 0x1234 from its Server before it receives a PUBACK for the PUBLISH packet that it sent.

```
Client                    Server
PUBLISH Packet Identifier=0x1234---→
←--PUBLISH Packet Identifier=0x1234
PUBACK Packet Identifier=0x1234---→
←--PUBACK Packet Identifier=0x1234
```

2.2.2 Return Code

The Return Code is a one byte unsigned value that indicates the result of an operation. Return Codes less than 128 indicate successful completion. The normal Return Code for success is 0. Return Code values of 128 or greater indicate failure.

The CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, DISCONNECT, and AUTH packets have a Return Code as part of the Variable Header. The SUBACK and UNSUBACK packets have a list of Return Codes in the Payload.

Table 2-4 Return Code List

Value	Hex	Name	Packets
0	0x00	Success	CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UNSUBACK, DISCONNECT, AUTH
0	0x00	Granted QoS 0	SUBACK
1	0x01	Granted QoS 1	SUBACK
2	0x02	Granted QoS 2	SUBACK
4	0x04	Disconnect with Will Message	DISCONNECT
17	0x11	No subscription existed	UNSUBACK
24	0x18	Continue authentication	AUTH
25	0x19	Re-authenticate	AUTH
128	0x80	Unspecified error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
129	0x81	Malformed Packet	CONNACK, DISCONNECT
130	0x82	Protocol Error	CONNACK, DISCONNECT
131	0x83	Implementation specific error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK
132	0x84	Unsupported Protocol Version	CONNACK
133	0x85	Client Identifier not valid	CONNACK
134	0x86	Bad User Name or Password	CONNACK
135	0x87	Not authorized	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT

136	0x88	Server unavailable	CONNACK
137	0x89	Server busy	CONNACK, DISCONNECT
138	0x8A	Banned	CONNACK
139	0x8B	Server shutting down	DISCONNECT
140	0x8C	Bad authentication method	CONNACK
141	0x8D	Keep Alive timeout	DISCONNECT
142	0x8E	Session taken over	DISCONNECT
143	0x8F	Topic Filter invalid	SUBACK, UNSUBACK, DISCONNECT
144	0x90	Topic Name invalid	CONNACK, PUBACK, PUBREC, DISCONNECT
145	0x91	Packet Identifier in use	PUBREC, SUBACK, UNSUBACK
146	0x92	Packet Identifier not found	PUBREL, PUBCOMP
147	0x93	Receive Maximum exceeded	DISCONNECT
149	0x95	Packet too large	CONNACK, DISCONNECT
150	0x96	Message rate too high	DISCONNECT
151	0x97	Quota exceeded	CONNACK, PUBACK, PUBREC, SUBACK, DISCONNECT
152	0x98	Administrative action	DISCONNECT
153	0x99	Payload format invalid	PUBACK, PUBREC, DISCONNECT
154	0x9A	Retain not supported	DISCONNECT
155	0x9B	QoS not supported	DISCONNECT
156	0x9C	Use another server	CONNACK, DISCONNECT
157	0x9D	Server moved	CONNACK, DISCONNECT
158	0x9E	Shared Subscription not supported	SUBACK, DISCONNECT
159	0x9F	Connection rate exceeded	CONNACK, DISCONNECT
161	0xA1	Subscription Identifiers not supported	SUBACK, DISCONNECT
162	0xA2	Wildcard Subscription not supported	SUBACK, DISCONNECT

2.2.3 Properties

The CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBACK, DISCONNECT, and AUTH packet Variable Header ends with a set of Properties. This is composed of a Property Length followed by the Properties. There is no significance in the order of Properties with different Identifiers.

2.2.3.1 Property Length

The Property Length is encoded as a Variable Byte Integer. The Property Length does not include the bytes used to encode itself, but includes the length of the Properties. A length of zero indicates that there are no Properties in the packet.

2.2.3.2 Property

A Property consists of an Identifier which defines the usage and data type, followed by the Value. The Identifier is encoded as a Variable Byte Integer. A packet which contains an Identifier which is not valid for this MQTT Control Packet type, or the following value is not of the specified data type it is a Malformed Packet. If received, use a CONNACK or DISCONNECT packet with Return Code 0x81 (Malformed Packet) as described in section 4.13 Handling errors.

Table 2.6 – Properties

Identifier		**Name**	**Type**	**Packet**
Dec	**Hex**			
1	0x01	Payload Format Indicator	Byte	PUBLISH
2	0x02	Publication Expiry Interval	Four Byte Integer	PUBLISH
3	0x03	Content Type	UTF-8 Encoded String	PUBLISH
8	0x08	Response Topic	UTF-8 Encoded String	PUBLISH
9	0x09	Correlation Data	Binary Data	PUBLISH
11	0x0B	Subscription Identifier	Variable Byte Integer	PUBLISH, SUBSCRIBE
17	0x11	Session Expiry Interval	Four Byte Integer	CONNECT, DISCONNECT
18	0x12	Assigned Client Identifier	UTF-8 Encoded String	CONNACK
19	0x13	Server Keep Alive	Two Byte Integer	CONNACK
21	0x15	Auth Method	UTF-8 Encoded String	CONNECT, CONNACK, AUTH
22	0x16	Auth Data	Binary Data	CONNECT, CONNACK, AUTH
23	0x17	Request Problem Information	Byte	CONNECT
24	0x18	Will Delay Interval	Four Byte Integer	CONNECT
25	0x19	Request Response Information	Byte	CONNECT
26	0x1A	Response Information	UTF-8 Encoded String	CONNACK
28	0x1C	Server Reference	UTF-8 Encoded String	CONNACK, DISCONNECT
31	0x1F	Reason String	UTF-8 Encoded String	CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, AUTH
33	0x21	Receive Maximum	Two Byte Integer	CONNECT, CONNACK
34	0x22	Topic Alias Maximum	Two Byte Integer	CONNECT, CONNACK

35	0x23	Topic Alias	Two Byte Integer	PUBLISH
36	0x24	Maximum QoS	Byte	CONNACK
37	0x25	Retain Available	Byte	CONNACK
38	0x26	User Property	UTF-8 String Pair	CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, AUTH
39	0x27	Maximum Packet Size	Four Byte Integer	CONNECT, CONNACK
40	0x28	Wildcard Subscription Available	Byte	CONNACK
41	0x29	Subscription Identifier Available	Byte	CONNACK
42	0x2A	Shared Subscription Available	Byte	CONNACK

Non-Normative comment
In this specification, only one-byte Identifiers are used.

2.3 Payload

Some MQTT Control Packets contain a Payload as the final part of the packet, as described in Chapter **Error! Reference source not found.**. In the case of the PUBLISH packet this is the Application Message. Table 2.6 – MQTT Control Packets that contain a Payload lists the MQTT Control Packets that require a Payload.

Table 2.6 – MQTT Control Packets that contain a Payload

MQTT Control Packet	Payload
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	Required

PINGREQ	None
PINGRESP	None
DISCONNECT	None
AUTH	None

3 MQTT Control Packets

3.1 CONNECT – Connection Request

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

A Client can only send the CONNECT packet once over a Network Connection. The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection of the Client [MQTT-3.1.0-2]. Refer to section 4.13 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

This is the length of the Variable Header plus the length of the Payload encoded as a Variable Byte Integer.

3.1.2 Variable Header

The Variable Header for the CONNECT Packet contains the following fields in the order: Protocol Name, Protocol Level, Connect Flags, Keep Alive, Property Length, and Properties. The rules for encoding Properties are described in section 2.2.3.

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.

A Server which supports multiple protocols uses the Protocol Name to verify that it has received a CONNECT packet. If the Server does not want to process the data and wishes to reveal that it is an MQTT Server it MAY send a CONNACK packet with Return Code of 0x84 (Unsupported Protocol Version) as described in section 4.13 Handling errors, and then close the Network Connection.

Non-Normative comment

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

3.1.2.2 Protocol Version

Figure 3-3 - Protocol Version byte

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

The one byte unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Level field for the version 5.0 of the protocol is 5 (0x05).

A Server which supports multiple versions of the MQTT protocol uses the Protocol Version to verify that it has received a Version 5 CONNECT packet. If the Protocol Version is not 5 and the Server does not want to process the data, the Server MAY send a CONNACK packet with Return Code 0x84 (Unsupported Protocol Version) as described in section 4.13 Handling errors, and then close the Network Connection.

3.1.2.3 Connect Flags

The Connect Flags byte contains several 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 Start	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT packet is set to 0 [MQTT-3.1.2-1]. If the reserved flag is not zero it is a Malformed Packet. Refer to section 4.13 for information about handling errors.

3.1.2.4 Clean Start

Position: bit 1 of the Connect Flags byte.

This bit specifies whether the Connection starts a new Session or is a continuation of an existing Session. Refer to section 3.1.2.12 for a definition of the Session state.

If Clean Start is set to 1, the Client and Server MUST discard any existing Session and start a new session [MQTT-3.1.2-2]. Consequently, the Session Present flag in CONNACK is always set to 0 if Clean Start is set to 1.

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 packet is accepted, a Will Message MUST be stored on the Server and associated with the Session [MQTT-3.1.2-3]. The Will Message MUST be published after the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Return Code 0x00 (Success) or 0x04 (Disconnect with Will Message) [MQTT-3.1.2-4].

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 with a Return Code 0x00 or 0x04.
- The Server closes the Network Connection without first receiving a DISCONNECT packet with a Return Code 0x00 0x04.

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-3.1.2-5]. The Will Message MUST be removed from the stored Session state in the Server once it has been published or the Server has received a DISCONNECT packet with a Return Code of 0x00 (Success) or 0x04 (Disconnect with Will Message) from the Client [MQTT-3.1.2-6].

If the Will Flag is set to 0, the Will QoS and Will Retain fields in the Connect Flags MUST be set to 0 and the Will Topic and Will Message fields MUST NOT be present in the Payload [MQTT-3.1.2-7]. If the Will Flag is set to 0, the Server MUST NOT publish a Will Message [MQTT-3.1.2-8].

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

Non-Normative comment

The client can arrange for the Will Message to notify that Session Expiry has occurred by setting the Will Delay Interval to be longer than the Session Expiry Interval and sending DISCONNECT with Return Code 0x04 (Disconnect with Will Message).

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-3.1.2-9]

If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). A value of 3 (0x03) is a Malformed Packet and the Server MUST close the Network Connection. [MQTT-3.1.2-10]

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 Will Retain MUST be set to 0. [MQTT-3.1.2-11]

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-3.1.2-12]
- If Will Retain is set to 1, the Server MUST publish the Will Message as a retained message. [MQTT-3.1.2-13]

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-3.1.2-14]

If the User Name Flag is set to 1, a User Name MUST be present in the Payload. [MQTT-3.1.2-15]

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-3.1.2-16]

If the Password Flag is set to 1, a Password MUST be present in the Payload. [MQTT-3.1.2-17]

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 Two Byte Integer which is a time interval measured in seconds. It is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one MQTT Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between MQTT Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other MQTT Control Packets, the Client MUST send a PINGREQ packet [MQTT-3.1.2-18].

If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value instead of the value it sent as the Keep Alive [MQTT-3.1.2-19].

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 an MQTT Control Packet from the Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to the Client as if the network had failed [MQTT-3.1.2-20].

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

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

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

> **Non-Normative comment**
>
> The actual value of the Keep Alive is application specific; typically, this is a few minutes. The maximum value is 18 hours 12 minutes and 15 seconds.

3.1.2.11 Property Length

The length of the Properties in the CONNECT packet Variable Header encoded as a Variable Byte Integer.

3.1.2.12 Session Expiry Interval

17 (0x11) Byte, Identifier of the Session Expiry Interval.

Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a protocol error to include the Session Expiry Interval more than once.

If the Session Expiry Interval is absent, the Session does not expire. If it is set to 0, the Session ends when the Network Connection is closed.

3.1.2.12.1 Session State

The Client and Server are required to store Session state so that reliable messaging can continue across a sequence of Network Connections. After the Network Connection is closed and the Session Expiry Interval has elapsed without a new connection being made, the Client and Server MUST delete the Session state they hold [MQTT-3.1.2-21].

If a new Network Connection is made before the Session has expired, the Server MUST resume communications with the Client based on state from the current Session (as identified by the Client identifier) [MQTT-3.1.2-22]. If there is no Session associated with the Client identifier the Server MUST create a new Session [MQTT-3.1.2-23]. The Client and Server MUST store the Session after the Network Connection is closed [MQTT-3.1.2-24].

After reconnection, the Session lasts as long as the Network Connection plus the new Session Expiry Interval.

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, including any Subscription Identifiers.
- 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.
- If the Session is currently not connected, the time at which the Session state will be deleted.

Retained messages do not form part of the Session state in the Server, they MUST NOT be deleted when the Session ends [MQTT-3.1.2-25].

Refer to section 4.1 for details and limitations of stored state.

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

Non-Normative comment

Setting Clean Start to 1 and a Session Expiry Interval of 0, is equivalent to setting CleanSession to 1 in the MQTT Specification Version 3.1.1.

Setting Clean Start to 0 and no Session Expiry Interval, is equivalent to setting CleanSession to 0 in the MQTT Specification Version 3.1.1.

Non-Normative comment

Typically, a Client will always connect using the same Session Expiry Interval. The choice will depend on the application. A Client that has its Session Expiry Interval always set to 0 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 a non-zero Session Expiry Interval will receive all QoS 1 or QoS 2 messages that were published while it was disconnected provided that its Session has not expired.

Non-Normative comment

A Client might be connecting to a Server using a network that provides intermittent connectivity. This Client can use a short Session Expiry Interval so that it can reconnect when the network is available again and continue reliable message delivery. If the Client does not reconnect, allowing the Session to expire, then Application Messages will be lost.

Non-Normative comment

When a Client connects with a long Session Expiry Interval, or no Session Expiry at all, it is requesting that the Server maintain its MQTT session state after it disconnects for an extended period. Clients should only connect with a long Session Expiry Interval 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 disconnect with a Session Expiry Interval set to 0.

Non-Normative comment

If the Client connects using this protocol, then reconnects using the MQTT V3.1.1 protocol using CleanStart 0 before the Session has expired, the Session state is kept indefinitely.

Non-Normative comment

The Client can avoid implementing its own Session expiry and instead rely on the Session Present flag returned from the Server to determine if the Session had expired. If the Client does implement its own Session expiry, it needs to store the time at which the Session state will be deleted as part of its Session state.

Non-Normative comment

The Client and Server clocks might drift and not measure time intervals accurately. The Client should always rely on the Session Present flag in the CONNACK packet rather than try to calculate whether the Server did keep its Session state.

3.1.2.13 Will Delay Interval

24 (0x18) Byte, Identifier of the Will Delay Interval.

Followed by the Four Byte Integer representing the Will Delay Interval in seconds. It is a Protocol Error to include the Will Delay Interval more than once. If the Will Delay Interval is absent, then there is no delay before the Will Message is published.

The Server delays publishing the Client's Will Message until the Will Delay Interval has passed or the Session ends, whichever happens first. If a new Network Connection to this Session is made before the Will Delay Interval has passed, the Server MUST NOT send the Will Message [MQTT-3.1.2-26].

Non-Normative comment

One use of this is to avoid publishing Will Messages if there is a temporary network disconnection and the Client succeeds in reconnecting and continuing its Session before the Will Message is published.

3.1.2.14 Receive Maximum

33 (0x21) Byte, Identifier of the Receive Maximum.

Followed by the Two Byte Integer representing the Receive Maximum value. It is a Protocol Error to include the Receive Maximum value more than once or for it to have the value 0.

The Client uses this value to limit the number of QoS 1 and QoS 2 publications that it is willing to process concurrently. There is no mechanism to limit the QoS 0 publications that the Server might try to send.

The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK or PUBCOMP from the Client [MQTT-3.1.2-27]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Client uses DISCONNECT with Return Code 0x93 (Receive Maximum exceeded) as described in section 4.13 Handling errors.

Where the Server has sent Receive Maximum PUBLISH packets without receiving acknowledgements, it SHOULD NOT delay the sending of other packet types so as to avoid a possible deadlock.

The value of Receive Maximum applies only to the current Network Connection.

If the Receive Maximum value is absent then its value defaults to 65535.

Refer to section 4.8 Flow Control for details of how the Receive Maximum is used.

> **Non-Normative comment**
>
> The Server might choose to send fewer than Receive Maximum messages to the Client without receiving acknowledgement, even if it has more that this number of messages available to send.
>
> **Non-Normative comment**
>
> The Server might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets.

3.1.2.15 Maximum Packet Size

39 (0x27) Byte, Identifier of the Maximum Packet Size.

Followed by a Four Byte Integer representing the Maximum Packet Size the Client is willing to accept. If the Maximum Packet Size is not present, no limit on the packet size is imposed beyond the limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

> **Non-Normative comment**
>
> It is the responsibility of the application to select a suitable Maximum packet size value if it chooses to restrict the Maximum Packet Size.

The packet size is the total number of bytes in an MQTT Control Packet, as defined in section 2.1.4. The Client uses the Maximum Packet Size to inform the Server that it will not process packets whose size exceeds this limit.

The Server MUST NOT send packets exceeding Maximum Packet Size to the Client [MQTT-3.1.2-28]. If a Client receives a packet whose size exceeds this limit, this is a Protocol Error, the Client uses DISCONNECT with Return Code 0x95 (Packet too large), as described in section 4.13.

Where a Packet is too large to send, the Server MUST discard it without sending it and then behave as if it had completed sending that publication [MQTT-3.1.2-29].

In the case of a Shared Subscription where the message is too large to send to one or more of the Clients but other Clients can receive it, the Server can choose either discard the message without sending the message to any of the Clients, or send the message to one of the Clients that can receive it.

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

3.1.2.16 Topic Alias Maximum

34 (0x22) Byte, Identifier of the Topic Alias Maximum.

Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a Protocol Error to include the Topic Alias Maximum value more than once.

This value indicates the highest value that the Client will accept as a Topic Alias sent by the Server. The Client uses this value to limit the number of Topic Aliases that it is willing to hold on this Connection. The Server MUST NOT send a Topic Alias in a PUBLISH packet to the Client greater than Topic Alias Maximum [MQTT-3.1.2-30]. A value of 0 indicates that the Client does not accept any Topic Aliases on this connection. If Topic Alias Maximum is absent or zero, the Server MUST NOT send any Topic Aliases to the Client [MQTT-3.1.2-31].

3.1.2.17 Request Response Information

25 (0x19) Byte, Identifier of the Request Response Information.

Followed by a Byte with a value of either 0 or 1. It is Protocol Error to include the Request Response Information more than once, or to have a value other than 0 or 1. If the Request Response Information is absent, the value of 0 is used.

The Client uses this value to request the Server to return Response Information in the CONNACK. A value of 0 indicates that the Server MUST NOT return Response Information [MQTT-3.1.2-32]. If the value is 1 the Server MAY return Response Information in the CONNACK packet.

Non-Normative comment

The Server can choose not to include Response Information in the CONNACK, even if the client requested it.

Refer to section 4.10 for more information about how Request / Response works.

3.1.2.18 Request Problem Information

23 (0x17) Byte, Identifier of the Request Problem Information.

Followed by a Byte with a value of either 0 or 1. It is a Protocol Error to include Request Problem Information more than once, or to have a value other than 0 or 1. If the Request Problem Information is absent, the value of 1 is used.

The Client uses this value to indicate whether the Reason String or User Properties are sent in the case of failures.

If the value of Request Problem Information is 0, the Server MAY return a Reason String or User Properties on a CONNACK or DISCONNECT packet, but MUST NOT send a Reason String or User Properties on any other packet [MQTT-3.1.2-33]. If the value is 0 and the Client receives a Reason String or User Properties in a packet other than CONNACK or DISCONNECT, uses a DISCONNECT packet with Return Code 0x82 (Protocol Error) as described in section 4.13 Handling errors.

If this value is 1, the Server MAY return a Reason String or User Properties on any packet where it is allowed.

3.1.2.19 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by a UTF-8 string pair. The first string represents a user defined name. The second string contains the corresponding value. Both strings MUST comply with restrictions for UTF-8 Encoded Strings [MQTT-3.1.2-34].

The User Property MAY appear multiple times if multiple name, value pairs are present. The same name field MAY appear more than once.

Non-Normative comment

User Properties on the CONNECT packet are used to send connection related properties from the Client to the Server. The meaning of these properties is not defined by this specification.

3.1.2.20 Auth Method

21 (0x15) Byte, Identifier of the Auth Method.

Followed by a UTF-8 Encoded String containing the name of the authentication method. Refer to section 4.12 to understand how extended authentication works.

If a Client sets an Auth Method on the CONNECT, the Client MUST NOT send any packets other than AUTH or DISCONNECT packets until it has received a CONNACK packet [MQTT-3.1.2-35].

3.1.2.21 Auth Data

22 (0x16) Byte, Identifier of the Auth Data.

Followed by Binary Data containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. Refer to section 4.12 to understand how extended authentication works.

3.1.2.22 Variable Header Non-Normative example

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 (5)	0	0	0	0	0	1	0	1
Connect Flags									

byte 8	User Name Flag (1) Password Flag (1) Will Retain (0) Will QoS (01) Will Flag (1) Clean Start(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
Properties									
byte 11	Length (5)	0	0	0	0	0	1	0	1
byte 12	Session Expiry Interval identifier (17)	0	0	0	1	0	0	0	1
byte 13	Session Expiry Interval (10)	0	0	0	0	0	0	0	0
byte 14		0	0	0	0	0	0	0	0
byte 15		0	0	0	0	0	0	0	0
byte 16		0	0	0	0	1	0	1	0

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 (ClientID)

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 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 1.5.4 [MQTT-3.1.3-4].

The Server MUST allow ClientID's 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 [MQTT-3.1.3-6]. It MUST then process the CONNECT packet as if the Client had provided that unique ClientID, and MUST return the Assigned Client Identifier in the CONNACK packet [MQTT-3.1.3-7].

If the Server rejects the ClientID it MAY respond to the CONNECT packet with a CONNACK using Return Code 0x85 (Client Identifier not valid) as described in section 4.13 Handling errors, and then it MUST close the Network Connection [MQTT-3.1.3-8].

Non-Normative comment

A Client implementation could provide a convenience method to generate a random ClientID's. When the Session Expiry Interval is long, and the Client or Server generate a ClientID, it is the Clients responsibility to record it reliably.

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 1.5.4 [MQTT-3.1.3-9].

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 Payload that is to be published to the Will Topic as described in Section 3.1.2.5. This field consists of Binary Data.

3.1.3.4 User Name

If the User Name Flag is set to 1, the User Name is the next field in the Payload. The User Name MUST be a UTF-8 Encoded String as defined in Section 1.5.4 [MQTT-3.1.3-10]. It can be used by the Server for authentication and authorization.

3.1.3.5 Password

If the Password Flag is set to 1, the Password is the next field in the Payload. The Password field is Binary Data.   Although this field is called Password, it can be used to carry any credential information.

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 MQTTv5.0 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 Network Connection.
2. The Server MUST validate that the CONNECT packet conforms to section 3.1 and close the Network Connection if it does not conform [MQTT-3.1.4-1]. The Server MAY send a DISCONNECT with a Return Code of 128 or greater as described in section 4.13 before closing the Network Connection.
3. The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST close the Network Connection [MQTT-3.1.4-2]. Before closing the Network Connection, it

MAY send an appropriate CONNACK response with a Return Code of 128 or greater as described in sections 3.2 and 4.13.

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

1. If the ClientID represents a Client already connected to the Server, the Server sends a DISCONNECT packet to the existing Client with Return Code of 0x8E (Session taken over) as described in section 4.13 and MUST close the Network Connection of the existing Client [MQTT-3.1.4-3]. If the existing Client has a Will Message, that Will Message is published as described in section 3.1.2.5.

2. The Server MUST perform the processing of CleanStart that is described in section 3.1.2.4 [MQTT-3.1.4-4].

3. The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Return Code [MQTT-3.1.4-5].

   Non-normative comment

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

4. Start message delivery and Keep Alive monitoring.

Clients are allowed to send further MQTT 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-3.1.4-6].

Non-Normative comment
Clients typically wait for a CONNACK packet, However, if the Client exploits its freedom to send MQTT 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.

Non-Normative comment
Clients that send MQTT Control Packets before they receive CONNACK will be unaware of the Server constraints and whether any existing Session is being used.

3.2 CONNACK – Connect acknowledgement

The CONNACK packet is the packet sent by the Server in response to a CONNECT packet received from a Client. The Server MUST send a CONNACK with a 0x00 (Success) Return Code before sending any Packet other than AUTH [MQTT-3.2.0-1]. The Server MUST send only one CONNACK in a Network Connection [MQTT-3.2.0-2].

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 3-7 – CONNACK packet .

Figure 3-7 – 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 encoded as a Variable Byte Integer.

3.2.2 Variable Header

The Variable Header of the CONNACK Packet contains the following fields in the order: Connect Acknowledge Flags, Connect Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

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 [MQTT-3.2.2-1].

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 Session Expiry Interval set to 0, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a 0x00 (Success) Return Code in the CONNACK packet [MQTT-3.2.2-2].

If the Server accepts a connection with non-zero Session Expiry Interval, the value set in Session Present depends on whether the Server has 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 has not already stored Session state, it MUST set Session Present to 0 in the CONNACK packet [MQTT-3.2.2-4]. This is in addition to setting a 0x00 (Success) Return Code in the CONNACK packet.

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. If 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 close the Network Connection. The Client can discard the Session state on both Client and Server by sending a DISCONNECT packet with Session Expiry set to 0.

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].

Non-Normative comment

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

3.2.2.3 Connect Return Code

Byte 2 in the Variable Header is the Connect Return Code.

The values the Connect Return Code 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 complete the Connection the Server MAY send a CONNACK packet containing the appropriate Connect Return code from this table. If a Server sends a CONNACK packet containing a Return code of 128 or greater it MUST then close the Network Connection [MQTT-3.2.2-6].

Table 3-1 - Connect Return Code values

Value	Hex	Return Code name	Description
0	0x00	Success	The Connection is accepted
128	0x80	Unspecified error	The Server does not wish to reveal the reason for the failure, or none of the other Return Codes apply.
129	0x81	Malformed Packet	Data within the CONNECT packet could not be correctly parsed.
130	0x82	Protocol Error	Data in the CONNECT packet does not conform to this specification
131	0x83	Implementation specific error	The CONNECT is valid but is not accepted by this Server
132	0x84	Unsupported Protocol Version	The Server does not support the level of the MQTT protocol requested by the Client
133	0x85	Client Identifier not valid	The Client Identifier is a valid string but is not allowed by the Server
134	0x86	Bad User Name or Password	The Server does not accept the User Name or Password specified by the Client
135	0x87	Not authorized	The Client is not authorized to connect
136	0x88	Server unavailable	The MQTT Server is not available
137	0x89	Server busy	The Server is busy. Try again later.
138	0x8A	Banned	This Client has been banned by administrative action. Contact the server administrator.

140	0x8C	Bad authentication method	The authentication method is not supported or does not match the authentication method currently in use
144	0x90	Topic Name invalid	The Will Topic Name is not malformed, but is not accepted by this Server
149	0x95	Packet too large	The CONNECT packet exceeded the maximum permissible size
151	0x97	Quota exceeded	An implementation imposed limit has been exceeded
154	0x9A	Retain unavailable	The Server has specified Retain unavailable in the CONNACK
156	0x9C	Use another server	The Client should temporarily use another server
157	0x9D	Server moved	The Client should permanently use another server
159	0x9F	Connection rate exceeded	The connection rate limit has been exceeded.

The Server MUST use one of the Return Code values in Table 3-1 - Connect Return Code values [MQTT-3.2.2-7].

Non-Normative comment

Return Code 0x80 (Unspecified error) may be used where the Server knows the reason for the failure but does not wish to reveal it to the Client, or when none of the other Return Code values applies.

The Server may choose to close the Network Connection without sending a CONNACK to enhance security in the case where an error is found on the CONNECT. For instance, when on a public network and the connection has not been authorized it might not be good to even indicate that this is an MQTT Server.

3.2.2.4 Property Length

This is the length of the Properties in the CONNACK packet Variable Header encoded as a Variable Byte Integer.

3.2.2.5 Receive Maximum

33 (0x21) Byte, Identifier of the Receive Maximum.

Followed by the Two Byte Integer representing the Receive Maximum value. It is a Protocol Error to include the Receive Maximum value more than once or for it to have the value 0.

The Server uses this value to limit the number of publications that it is willing to process concurrently for the Client, it does not provide a mechanism to limit the QoS 0 publications that the Client might try to send.

The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK or PUBCOMP from the Server [MQTT-3.2.2-8]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Server uses a DISCONNECT packet with Return Code 0x93 (Receive Maximum exceeded) as described in section 4.13 Handling errors.

The Client MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them [MQTT-3.2.2-9]. This might result in deadlock. The value of Receive Maximum applies only to the current Network Connection.

If the Receive Maximum value is absent, then its value defaults to 65535.

Refer to section 4.8 Flow Control for details of how the Receive Maximum is used.

> **Non-Normative comment**
>
> The Client might choose to send fewer than Receive Maximum messages to the Client without receiving acknowledgement, even if it has more that this number of messages available to send.

> **Non-Normative comment**
>
> The Client might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets.

> **Non-Normative comment**
>
> If the Client sends QoS 1 or QoS 2 PUBLISH packets before it has received a CONNACK packet, it risks being disconnected because it has sent more than Receive Maximum publications.

3.2.2.6 Maximum QoS

36 (0x24) Byte, Identifier of the Maximum QoS.

Followed by a Byte with a value of either 0 or 1. It is a Protocol Error to include Maximum QoS more than once, or to have a value other than 0 or 1. If the Maximum QoS absent, the Client uses a Maximum QoS of 2.

If a Server does not support QoS 1 or QoS 2 PUBLISH packets it MUST send a Maximum QoS in the CONNACK packet specifying the highest QoS it supports [MQTT-3.2.2-10]. A server that does not support QoS 1 or QoS 2 PUBLISH packets MUST still accept SUBSCRIBE packets containing a Requested QoS of 0, 1 or 2 [MQTT-3.2.2-11].

If a Client receives a Maximum QoS from a Server, it MUST NOT send PUBLISH packets at a QoS level exceeding the Maximum QoS level specified [MQTT-3.2.2-12]. It is a Protocol Error if the Server receives a PUBLISH packet with a QoS greater than the Maximum QoS it specified. In this case use DISCONNECT with Return Code 0x9B (QoS not supported) as described in section 4.13 Handling errors.

If a Server receives a CONNECT packet containing a Will QoS that exceeds its capabilities, it MUST reject the connection. Use a CONNACK packet with Return Code 0x9B (QoS not supported) as described in section 4.13 Handling errors, and MUST close the Network Connection [MQTT-3.2.2-13].

If a Server has not sent a Maximum QoS, and receives a QoS > 0 PUBLISH packet and that QoS level exceeds its capabilities it MUST reply with a PUBACK or PUBREC with the Return Code 0x9B (QoS not supported). In the case of QoS 2 messages, the Server MUST process the subsequent PUBREL packet and issue a PUBCOMP to complete the protocol flow.

> **Non-Normative comment**
>
> A Client is not required to support reception or transmission of QoS 1 or QoS 2 PUBLISH packets. If this is the case, the Client simply restricts the maximum QoS field in any SUBSCRIBE commands it sends to a value it can support.

3.2.2.7 Retain Available

37 (0x25) Byte, Identifier of Retain Available.

Followed by a Byte field. If present, this byte declares whether the Server supports retained messages. A value is 0 means that retained messages are not supported. A value of 1 means retained messages are supported. If not present, then retained messages are supported. It is a Protocol Error to include Retain Available more than once or to use a value other than 0 or 1.

If a Server receives a CONNECT packet containing a Will Message with the Will Retain 1, and it does not support retained publications, the Server MUST reject the connection request. It SHOULD send CONNACK with Return Code 0x9A (Retain unavailable) and then it MUST close the Network Connection [MQTT-3.2.2-14].

A client receiving Retain Available from the Server MUST NOT send a PUBLISH packet with the RETAIN flag set to 1 [MQTT-3.2.2-15]. If the Server receives such a packet, this is a Protocol Error. The Server uses DISCONNECT with Return Code of 0x9A (Retain unavailable) as described in section 4.13.

3.2.2.8 Maximum Packet Size

39 (0x27) Byte, Identifier of the Maximum Packet Size.

Followed by a Four Byte Integer representing the Maximum Packet Size the server is willing to accept. If the Maximum Packet Size is not present, there is no limit on the packet size imposed beyond the limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

It is a Protocol Error to include the Maximum Packet Size more than once, or for the value to be sent to zero, or to be set greater than 2684354565.

The packet size is the total number of bytes in an MQTT Control Packet, as defined in section 2.1.4. The Server uses the Maximum Packet Size to inform the Client that it will not process packets whose size exceeds this limit.

The Client MUST NOT send packets exceeding Maximum Packet Size to the Server [MQTT-3.2.2-16]. If a Server receives a packet whose size exceeds this limit, this is a Protocol Error, the Server uses DISCONNECT with Return Code 0x95 (Packet too large), as described in section 4.13.

3.2.2.9 Assigned Client Identifier

18 (0x12) Byte, Identifier of the Assigned Client Identifier.

Followed by the UTF-8 string which is the Assigned Client Identifier. It is a Protocol Error to include the Assigned Client Identifier more than once.

The Client Identifier which was assigned by the Server because a zero length Client Identifier was found in the CONNECT packet.

If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any Session currently known in the Server [MQTT-3.2.2-17].

3.2.2.10 Topic Alias Maximum

34 (0x22) Byte, Identifier of the Topic Alias Maximum.

Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a Protocol Error to include the Topic Alias Maximum value more than once.

This value indicates the highest value that the Server will accept as a Topic Alias sent by the Client. The Server uses this value to limit the number of Topic Aliases that it is willing to hold on this Connection. The Client MUST NOT send a Topic Alias in a PUBLISH packet to the Server greater than this value [MQTT-3.2.2-18]. A value of 0 indicates that the Server does not accept any Topic Aliases on this connection. If Topic Alias Maximum is absent, the Client MUST NOT send any Topic Aliases on to the Server [MQTT-3.2.2-19].

3.2.2.11 Reason String

31 (0x1F) Byte Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT use this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.2.2-20]. It is a Protocol Error to include the Reason String more than once.

Non-Normative comment

Proper uses for the reason string in the Client would include putting this information into an exception thrown by the client code, or writing this string to a log.

3.2.2.12 User Property

38 (0x26) Byte, Identifier of User Property.
Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the CONNACK beyond the Maximum Packet Size specified by the session partner [MQTT-3.2.2-21]. This property may be included more than once.

The receiver of a CONNACK containing this property is not required to parse or process this property.

3.2.2.13 Wildcard Subscription Available

40 (0x28) Byte, Identifier of Wildcard Subscription Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Wildcard Subscriptions. A value is 0 means that Wildcard Subscriptions are not supported. A value of 1 means Wildcard Subscriptions are supported. If not present, then Wildcard Subscriptions are supported. It is a Protocol Error to include the Wildcard Subscription Available more than once or to send a value other than 0 or 1.

If the Server receives a SUBSCRIBE packet containing a Wild Card Subscription and it does not support Wild Card Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Return Code 0xA2 (Wildcard subscription not supported) as described in section 4.13.

3.2.2.14 Subscription Identifiers Available

41 (0x29) Byte, Identifier of Subscription Identifier Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Subscription Identifiers. A value is 0 means that Subscription Identifiers are not supported. A value of 1 means Subscription Identifiers are supported. If not present, then Subscription Identifiers are supported. It is a

Protocol Error to include the Subscription Identifier Available more than once, or to send a value other than 0 or 1.

If the server receives a SUBSCRIBE packet containing Subscription Identifier and it does not support Subscription Identifiers, this is a Protocol Error. The Server uses DISCONNECT with Return Code of 0xA1 (Subscription Identifiers not supported) as described in section 4.13.

3.2.2.15 Shared Subscription Available

42 (0x2A) Byte, Identifier of Shared Subscription Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Shared Subscriptions. A value is 0 means that Shared Subscriptions are not supported. A value of 1 means Shared Subscriptions are supported. If not present, then Shared Subscriptions are supported. It is a Protocol Error to include the Shared Subscription Available more than once or to send a value other than 0 or 1.

If the server receives a SUBSCRIBE packet containing Shared Subscriptions and it does not support Shared Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Return Code 0x9E (Shared Subscription not supported) as described in section 4.13.

3.2.2.16 Server Keep Alive

19 (0x13) Byte, Identifier of the Server Keep Alive.

Followed by a Two Byte Integer with the Keep Alive time assigned by the server. If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT [MQTT-3.2.2-22]. If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT [MQTT-3.2.2-23]. It is a Protocol Error to include the Server Keep Alive more than once.

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

3.2.2.17 Response Information

26 (0x1A) Byte, Identifier of the Response Information.

Followed by a UTF-8 Encoded String which is used as the basis for creating a Response Topic. The way in which the Client creates a Response Topic form the Response Information is not defined by this specification. It is a Protocol Error to include the Response Information more than once.

If the Client sends a Request Response Information with a value 1, the Server MAY send the Response Information in the CONNACK.

> **Non-Normative comment**
>
> A common use of this is to pass a globally unique portion of the topic tree which is reserved for this Client for at least the lifetime of its Session. This often cannot just be a random name as both the requesting Client and the responding Client need to be authorized to use it. It is normal to use this as the root of a topic tree for a particular client. For the Server to return this information, it normally needs to be correctly configured. Using this mechanism allows this configuration to be done once in the Server rather than in each Client.

Refer to section 4.10 for more information about how Request / Response works.

3.2.2.18 Server Reference

28 (0x1C) Byte, Identifier of the Server Reference.

Followed by a UTF-8 Encoded String which can be used by the Client to identify another Server to use. It is a Protocol Error to include the Server Reference more than once.

The Server uses a Server Reference in either a CONNACK or DISCONNECT packet with Return code of 0x9C (Use another server) or Return Code 0x9D (Server moved) as described in section 4.13.

Refer to section 4.11 Server redirection for information about how Server Reference is used.

3.2.2.19 Auth Method

21 (0x15) Byte, Identifier of the Auth Method.

Followed by a UTF-8 Encoded String containing the name of the authentication method. Refer to section 4.12 to understand how extended authentication works.

3.2.2.20 Auth Data

22 (0x16) Byte, Identifier of the Auth Data.

Followed by Binary Data containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. Refer to section 4.12 to understand how extended authentication works.

3.2.3 Payload

The CONNACK packet has no Payload.

3.3 PUBLISH – Publish message

A PUBLISH 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 3-8 – 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 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 receiver of an MQTT 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 MQTT 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 2.2.1 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

If the Server included a Maximum QoS in its CONNACK response to a Client and it receives a PUBLISH packet with a QoS greater than this, then it uses DISCONNECT with Return Code 0x9B (QoS not supported) as described in section 4.13 Handling errors.

A PUBLISH Packet MUST NOT have both QoS bits set to 1 [MQTT-3.3.1-4]. If a Server or Client receives a PUBLISH packet which has both QoS bits set to 1 it is a Malformed Packet. Use DISCONNECT with Return Code 0x81 (Malformed Packet) as described in section 4.13.

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 replace any existing retained message for this topic and store the Application Message and its QoS [MQTT-3.3.1-5], so that it can be delivered to future subscribers whose subscriptions match its Topic Name. If the

Payload contains zero bytes it is processed normally by the Server but any retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message [MQTT-3.3.1-6]. A zero length retained message MUST NOT be stored as a retained message on the Server [MQTT-3.3.1-7].

If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the message as a retained message and MUST NOT remove or replace any existing retained message [MQTT-3.3.1-8].

If the Server included Retain Available in its CONNACK response to a Client with its value set to 0 and it receives a PUBLISH packet with the RETAIN flag is set to 1, then it uses the DISCONNECT Return Code of 0x9A (Retain not supported) as described in section 4.13.

When a new Non-Shared Subscription is established, the last retained message, if any, on each matching topic name are sent to the Client. These messages are sent with the RETAIN flag set to 1. Which retained messages are sent is controlled by the Retain Handling Subscription Option. Refer to section 3.8.3.1 for a definition of the Subscription Options.

- If Retain Handling is set to 0 the Server MUST send all retrained messages matching the Topic Filter of the subscription to the Client [MQTT-3.3.1-9].
- If Retain Handling is set to 1 and the subscription did not already exist, the Server MUST send all retained message matching the Topic Filter of the subscription to the Client [MQTT-3.3.1-10].
- If Retain Handling is set to 2, the Server MUST NOT send retained messages at the time of the subscribe [MQTT-3.3.1-11].

If the Server receives a PUBLISH packet with the RETAIN flag set to 1, and QoS 0 it SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any time. If this happens there will be no retained message for that topic. Refer to section 4.1 for more information on storing state.

If a retained message expires, it is removed from the store and there will be no retained message for that topic.

The setting of the RETAIN flag in an Application Message forwarded by the Server from an established connection is controlled by the Retain As Published subscription option. Refer to section 3.8.3.1 for a definition of the Subscription Options.

- If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the received PUBLISH packet [MQTT-3.3.1-12].
- If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN flag equal to the RETAIN flag in the received PUBLISH packet [MQTT-3.3.1-13].

Non-Normative comment

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

3.3.1.4 Remaining Length

This is the length of Variable Header plus the length of the Payload encoded as a Variable Byte Integer.

3.3.2 Variable Header

The Variable Header of the PUBLISH Packet contains the following fields in the order: Topic Name, Packet Identifier, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

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 as defined in section 1.5.4 [MQTT-3.3.2-1].

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.

To reduce the size of the PUBLISH packet the sender can use a Topic Alias. The Topic Alias is described in section **Error! Reference source not found.**. It is a Protocol Error if the Topic Name is zero length and there is no Topic Alias.

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 2.2.1 provides more information about Packet Identifiers.

3.3.2.3 Property Length

The length of the Properties in the PUBLISH packet Variable Header encoded as a Variable Byte Integer.

3.3.2.4 Payload Format Indicator

1 (0x01) Byte, Identifier of the Payload Format Indicator.

Followed by the value of the Payload Format Indicator, either of:

- 0 (0x00) Byte Indicates that the Payload is unspecified bytes, which is equivalent to not sending a Payload Format Indicator.
- 1 (0x01) Byte Indicates that the Payload is UTF-8 Encoded Character Data. Note that the UTF-8 Data in the Payload does not include a length prefix, nor is it subject to the restrictions described in section 1.5.3.

A Server MUST send the Payload Format Indicator unaltered to all subscribers receiving the publication [MQTT-3.3.2-4]. The receiver MUST validate that the Payload is of the format indicated, and if it is not send a PUBACK, PUBREC, or DISCONNECT with Return Code of 0x99 (Payload format invalid) as described in section 4.13 [MQTT-3.3.2-5].

3.3.2.5 Publication Expiry Interval

2 (0x02) Byte, Identifier of the Publication Expiry Interval.

Followed by the Four Byte Integer representing the Publication Expiry Interval.

If present, the Four Byte value is the lifetime of the publication in seconds. If the Publication Expiry Interval has passed and the Server has not managed to start onward delivery to a matching subscriber, then it MUST delete the copy of the message for that subscriber [MQTT-3.3.2-6].

If absent, the publication does not expire.

The PUBLISH packet sent to a Client by the Server MUST contain a Publication Expiry Interval set to the received value minus the time that the publication has been waiting in the Server [MQTT-3.3.2-7]. Refer to section 4.1 for details and limitations of stored state.

3.3.2.6 Topic Alias

35 (0x23) Byte, Identifier of the Topic Alias.

Followed by the Two Byte integer representing the Topic Alias value. It is a Protocol Error to include the Topic Alias value more than once.

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

The sender decides whether to use a Topic Alias and chooses the value. It seta a Topic Alias mapping by including a non-zero length Topic Name and a Topic Alias in the PUBLISH packet. The receiver processes the PUBLISH as normal but also sets the specified Topic Alias mapping to this Topic Name.

If a Topic Alias mapping has been set at the receiver, a sender can send a PUBLISH packet that contains that Topic Alias and a zero length Topic Name. The receiver then treats the incoming PUBLISH as if it had contained the Topic Name of the Topic Alias.

A sender can modify the Topic Alias mapping by sending another PUBLISH in the same Network Connection with the same Topic Alias value and a different non-zero length Topic Name.

Topic Alias mappings exist only within a Network Connection and last only for the lifetime of that Network Connection. A receiver MUST NOT carry forward any Topic Alias mappings from one Network Connection to another [MQTT-3.3.2-8].

A Topic Alias of 0 is not permitted. A sender MUST NOT send a PUBLISH packet containing a Topic Alias which has the value 0 [MQTT-3.3.2-9].

A Client MUST NOT send a PUBLISH packet with a Topic Alias whose value exceeds the Topic Alias Maximum value returned by the Server in the CONNACK packet [MQTT-3.3.2-10]. A Client MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it sent in the CONNECT packet [MQTT-3.3.2-11].

A Server MUST NOT send a PUBLISH packet with a Topic Alias whose value exceeds the Topic Alias Maximum value sent by the Client in the CONNECT packet [MQTT-3.3.2-12]. A Server MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it returned in the CONNACK packet [MQTT-3.3.2-13].

The Topic Alias mappings used by the Client and Server are independent from each other. Thus, when a Client sends a PUBLISH containing a Topic Alias value of 1 to a Server and the Server sends a PUBLISH with a Topic Alias value of 1 to that Client they will in general be referring to different Topics.

3.3.2.7 Response Topic

8 (0x08) Byte, Identifier of the Response Topic.

Followed by a UTF-8 Encoded String which is used as the Topic Name for a response message. The Response Topic MUST be a UTF-8 Encoded String as defined in section 1.5.3 [MQTT-3.3.2-14]. The Response Topic MUST NOT contain wildcard characters [MQTT-3.3.2-15]. It is a Protocol Error to include the Response Topic more than once. The presence of a Response Topic identifies the Message as a Request.

Refer to section 4.10 for more information about how Request / Response works.

The Server MUST send the Response Topic unaltered to all subscribers receiving the publication [MQTT-3.3.2-16].

Non-Normative comment:
The receiver of an Application Message with a Response Topic sends a response by using the Response Topic as the Topic Name of a PUBLISH. If the Request Message contains a Correlation Data, the receiver of the Request Message should also include this Correlation Data as a property in the PUBLISH packet of the Response Message.

Non-Normative comment:

The Server cannot forward the Response Topic to a client using MQTT Specification Version 3.1.1, hence such a client cannot take part in the Request / Response dialogue.

3.3.2.8 Correlation Data

9 (0x09) Byte, Identifier of the Correlation Data.

Followed by Binary Data. The Correlation Data is used by the sender of the Request Message to identify which request the Response Message is for when it is received. It is a Protocol Error to include Correlation Data more than once. If the Correlation Data is not present, the Requestor does not require any correlation data.

The Server MUST send the Correlation Data unaltered to all subscribers receiving the publication [MQTT-3.3.2-17]. The value of the Correlation Data only has meaning to the sender of the Request Message and receiver of the Response Message.

Non-Normative comment:

The receiver of an Application Message which contains both a Response Topic and a Correlation Data sends a response by using the Response Topic as the Topic Name of a PUBLISH. The Client should also send the Correlation Data unaltered as part of the PUBLISH of the responses.

If the Correlation Data contains information which can cause application failures if modified by the Client responding to the request, it should be encrypted and/or hashed to allow any alteration to be detected.

Refer to section 4.10 for more information about how Request / Response works.

Non-Normative comment:

The Server cannot forward the Correlation Data to a client using MQTT Specification Version 3.1.1, applications using back level Clients might not behave correctly if they rely on this property.

3.3.2.9 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by a UTF-8 String Pair. The first string represents a name, and the second string contains the corresponding value. Both strings MUST comply with restrictions for UTF-8 Encoded Strings [MQTT-3.3.2-18].

There can be multiple User Properties in a PUBLISH packet, and the same name can occur more than once.

The Server MUST send all User Properties unaltered in a PUBLISH packet when forwarding the Application Message to a Client [MQTT-3.3.2-19]. The Server MUST maintain the order of User Properties when forwarding the Application Message [MQTT-3.3.2-20].

Non-Normative comment

This data type is intended to provide a means of transferring application layer name-value tags whose meaning and interpretation are known only by the application programs responsible for sending and receiving them.

Non-Normative comment:

The Server cannot forward the User Properties to a client using MQTT Specification Version 3.1.1, applications using back level Clients might not behave correctly if they rely on this property

3.3.2.10 Subscription Identifier

11 (0x0B), Identifier of the Subscription Identifier.
Followed by a Variable Byte Integer representing the identifier of the subscription.

The Subscription Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the Subscription Identifier has a value of 0. Several Subscription Identifiers may be included if the publication matches multiple subscriptions, in this case their order is not significant.

3.3.2.11 Content Type

3 (0x03) Identifier of the Content Type.
Followed by a UTF-8 Encoded String describing the content of the Application Message. The Content Type MUST be a UTF-8 Encoded String as defined in section 1.5.3 [MQTT-3.3.2-21].
It is a Protocol Error to include the Content Type more than once. The value of the Content Type is defined by the sending and receiving application.

A Server MUST send the Content Type unaltered to all subscribers receiving the publication [MQTT-3.3.2-22].

Non-Normative comment

The UTF-8 Encoded String may use a MIME content type string to describe the contents of the Application message. However, since the sending and receiving applications are responsible for the definition and interpretation of the string, MQTT performs no validation of the string except to insure it is a valid UTF-8 Encoded String.

Non-Normative comment:

The Server cannot forward the Content Type to a client using MQTT Specification Version 3.1.1, applications using back level Clients might not behave correctly if they rely on this property.

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.

Table 3-3 PUBLISH packet non-normative example

Field	Value
Topic Name	a/b
Packet Identifier	10
Properties	None

Figure 3-9 - 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
byte 8	No Properties	0	0	0	0	0	0	0	0

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. The PUBLISH packet includes the Subscription Identifier carried in the SUBSCRIBE packet, if there was one.

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-3.3.4-2]. 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.

If the Client specified a Subscription Identifier for any of the overlapping subscriptions the Server MUST send those Subscription Identifiers in the message which is published as the result of the subscriptions [MQTT-3.3.4-3]. If the Server sends a single copy of the message it MUST include in the PUBLISH packet the Subscription Identifiers for all matching subscriptions which have a Subscription Identifiers, their order is not significant [MQTT-3.3.4-4]. If the Server sends multiple PUBLISH packets it MUST send in each of them the Subscription Identifier of the matching subscription if it has a Subscription Identifier [MQTT-3.3.4-5].

It is possible that the client made several subscriptions which match a publication and that it used the same identifier for more than one of them. In this case the PUBLISH packet will carry multiple identical subscription identifiers.

It is a Protocol Error for a PUBLISH packet to contain any subscription identifier other than those received in SUBSCRIBE packet which caused it to flow. A PUBLISH packet sent from a Client to a Server MUST NOT contain a subscription identifier [MQTT-3.3.4-6].

If the subscription was shared, then only the subscription identifiers that were in present in the SUBSCRIBE packet from the client which is receiving the message are returned in the PUBLISH packet.

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

If the PUBLISH packet contains a Topic Alias, the Receiver processes it as follows:

1) A Topic Alias value of 0 or greater than the Maximum Topic Alias is a protocol error, the receiver uses DISCONNECT with Return Code of 0x94 (Topic Alias invalid) as described in section 4.13.

2) If the receiver has already established a mapping for the Topic Alias, then
   a) If the packet has a zero length Topic Name, the receiver processes it using the Topic Name that corresponds to the Topic Alias
   b) If the packet contains a non-zero length Topic Name, the receiver processes the packet using that Topic Name and updates its mapping for the Topic Alias to the Topic Name from the incoming packet

3) If the Receiver does not already have a mapping for this Topic Alias
   a) If the packet has a zero length Topic Name field it is a Protocol Error and the receiver uses DISCONNECT with Return Code of 0x82 (Protocol Error) as described in section 4.13.
   b) If the packet contains a Topic Name with a non-zero length, the receiver processes the packet using that Topic Name and sets its mappings for the Topic Alias to Topic Name from the incoming packet.

3.4 PUBACK – Publish acknowledgement

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

3.4.1 Fixed Header

Figure 3-10 - 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							

Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

3.4.2 Variable Header

The Variable Header of the PUBACK Packet contains the following fields in the order: Packet Identifier from the PUBLISH packet that is being acknowledged, PUBACK Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

Figure 3-11 – PUBACK packet Variable Header

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							
byte 3	PUBACK Return Code							

byte 4	Property Length

3.4.2.1 PUBACK Return Code

Byte 3 in the Variable Header is the PACK Return Code. If the Remaining Length is less than 3, there is no Return Code and the value of 0x00 (Success) is used.

The values for the Publish Return code field are listed in Table 3-5 - PUBACK Return Codes

Table 3-5 - PUBACK Return Codes

Value	Hex	Return Code name	Description
0	0x00	Success	The message is accepted. Publication of the QoS 1 message proceeds.
16	0x10	No matching subscribers.	The message is accepted but there are no subscribers. This is sent only by the Server. If the Server does not know if there are any matching subscribers, it MUST use the 0x00 (Success) Return Code [MQTT-3.4.2-1].
128	0x80	Unspecified error	The receiver does not accept the publish but either does not want to reveal the reason, or it does not match one of the other values.
131	0x83	Implementation specific error	The PUBLISH is valid but the receiver is not willing to accept it.
135	0x87	Not authorized	The PUBLISH is not authorized
144	0x90	Topic Name invalid	The Topic Name is not malformed, but is not accepted by this Client or Server
151	0x97	Quota exceeded	An implementation imposed limit has been exceeded.
153	0x99	Payload format invalid	The payload format does not match the specified Payload Format Indicator

The Client or Server sending the PUBACK MUST use one of the PUBACK Return Codes shown in Table 3-5 - PUBACK Return Codes [MQTT-3.4.2-2]. The Return Code 0x00 (Success) may be sent by using a Remaining Length of 2.

3.4.2.2 Property Length

The length of the Properties in the PUBACK packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

3.4.2.3 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.4.2-3]. It is a Protocol Error to include the Reason String more than once.

3.4.2.4 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the PUBACK beyond the Maximum Packet Size specified by the session partner [MQTT-3.4.2-4]. This property may be included more than once.

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 delivery 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 3-12 - 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							

Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

3.5.2 Variable Header

The Variable Header of the PUBREC Packet consists of the following fields in the order: the Packet Identifier from the PUBLISH packet that is being acknowledged, PUBREC Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

Figure 3-13 - PUBREC packet Variable Header

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

byte 2	Packet Identifier LSB
byte 3	PUBREC Return Code
byte 4	Property Length

3.5.2.1 PUBREC Return Code

Byte 3 in the Variable Header is the PUBREC Return Code.

The values for the one byte unsigned Publish Return Code field are listed in Table 3-6 – PUBREC . If the Remaining Length is 2, then the Publish Return Code has the value 0x00 (Success).

Table 3-6 – PUBREC Return Codes

Value	Hex	Return Code name	Description
0	0x00	Success	The message is accepted. Publication of the QoS 2 message proceeds.
16	0x10	No matching subscribers.	The message is accepted but there are no subscribers. This is sent only by the Server. If the Server is does not know if there are any matching subscribers, it MUST use the 0x00 (Success) Return Code [MQTT-3.5.2-1].
128	0x80	Unspecified error	The receiver does not accept the publish but either does not want to reveal the reason, or it does not match one of the other values.
131	0x83	Implementation specific error	The PUBLISH is valid but the receiver is not willing to accept it.
135	0x87	Not authorized	The PUBLISH is not authorized
144	0x90	Topic Name invalid	The Topic Name is not malformed, but is not accepted by this Client or Server
145	0x91	Packet Identifier in use	The Packet Identifier is already in use. This might indicate a mismatch in the session state between the Client and Server.
151	0x97	Quota exceeded	An implementation imposed limit has been exceeded.
153	0x99	Payload format invalid	The payload format does not match the one specified in the Payload Format Indicator.

The Client or Server sending the PUBREC MUST use one of the Return Codes in Table 3-6 – PUBREC Return Codes [MQTT-3.5.2-2]. The Return Code 0x00 (Success) may be sent by using a Remaining Length of 2.

3.5.2.2 Property Length

The length of the Properties in the PUBREC packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

3.5.2.3 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.5.2-3]. It is a Protocol Error to include the Reason String more than once.

3.5.2.4 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the PUBREC beyond the Maximum Packet Size specified by the session partner [MQTT-3.5.2-4]. This property may be included more than once.

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 delivery 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 3-14 – 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							

Bits 3,2,1 and 0 of the Fixed Header in the PUBREL 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, encoded as a Variable Byte Integer.

3.6.2 Variable Header

The Variable Header of the PUBREL Packet contains the following fields in the order: the Packet Identifier from the PUBREC packet that is being acknowledged, PUBREL Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

Figure 3-15 – PUBREL packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							
byte 3	PUBREL Return Code							
byte 4	Property Length							

3.6.2.1 PUBREL Return Code

Byte 3 in the Variable Header is the PUBREL Return Code. If the Remaining Length is less than 3 the value of 0x00 (Success) is used.

The values for the one byte unsigned PUBCOMP Return Code field are listed in Table 3-7 - PUBREL Return Codes.

Table 3-7 - PUBREL Return Codes

Value	Hex	Return Code name	Description
0	0x00	Success	Message released. Publication of QoS 2 message is complete.
146	0x92	Packet Identifier not found	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session state on the Client and Server.

The Client or Server sending the PUBREL MUST use one of the Return Codes in Table 3-7 - PUBREL Return Codes [MQTT-3.6.2-1]. The Return Code 0x00 (Success) may be sent by using a Remaining Length of 2.

3.6.2.2 Property Length

The length of the Properties in the PUBREL packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

3.6.2.3 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the Client [MQTT-3.6.2-2]. It is a Protocol Error to include the Reason String more than once.

3.6.2.4 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information for the PUBREL. The sender MUST NOT send this property if it would increase the size of the PUBREL beyond the Maximum Packet Size specified by the session partner [MQTT-3.6.2-3]. This property may be included more than once.

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 delivery 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 3-16 – 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							

Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

3.7.2 Variable Header

The Variable Header of the PUBCOMP Packet contains the following fields in the order: Packet Identifier from the PUBREL packet that is being acknowledged, PUBCOMP Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

Figure 3-17 - PUBCOMP packet Variable Header

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

byte 2	Packet Identifier LSB
byte 3	PUBCOMP Return Code
byte 4	Property Length

3.7.2.1 PUBCOMP Return Code

Byte 3 in the Variable Header is the PUBCOMP Return Code. If the Remaining Length is less than 3, the value 0x00 (Success) is used.

The values for the one byte unsigned PUBCOMP Return Code field are listed in Table 3-8 – PUBCOMP Return Code.

Table 3-8 – PUBCOMP Return Codes

Value	Hex	Return Code name	Description
0	0x00	Success	Message released. Publication of QoS 2 message is complete.
146	0x92	Packet Identifier not found	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session state on the Client and Server.

The Client or Server sending the PUBCOMP MUST use one of the Return Codes in Table 3-8 – PUBCOMP Return Code [MQTT-3.7.2-1]. The Return Code 0x00 (Success) may be sent by using a Remaining Length of 2.

3.7.2.2 Property Length

The length of the Properties in the PUBCOMP packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

3.7.2.3 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by the Client [MQTT-3.7.2-2]. It is a Protocol Error to include the Reason String more than once.

3.7.2.4 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the PUBCOMP beyond the Maximum Packet Size specified by the session partner [MQTT-3.7.2-3]. This property may be included more than once.

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 request

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 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 3-18 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 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 plus the length of the Payload encoded as a Variable Byte Integer.

3.8.2 Variable Header

The Variable Header of the SUBSCRIBE Packet contains the following fields in the order: Packet Identifier, Property Length, and the Properties. Section 2.2.1 provides more information about Packet Identifiers. The rules for encoding Properties are described in section 2.2.3.

Variable Header non-normative example

Figure 3-19- 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
byte 3	Property Length (0)	0	0	0	0	0	0	0	0

3.8.2.1 Property Length

The length of Properties in the SUBSCIBE packet Variable Header encoded as a Variable Byte Integer.

3.8.2.2 Subscription Identifier

11 (0x0B) Byte, Identifier of the Subscription Identifier.

Followed by a Variable Byte Integer representing the identifier of the subscription. The Subscription Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the Subscription Identifier has a value of 0. It is a Protocol Error to include the Subscription Identifier more than once.

The Subscription Identifier is associated with any subscription created or modified as the result of this SUBSCRIBE packet. If there is a Subscription Identifier, it is stored with the subscription. If this property is not specified, then the absence of a Subscription Identifier is stored with the subscription.

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 MUST be a UTF-8 Encoded String [MQTT-3.8.3-1]. Each Topic Filter is followed by a Subscription Options byte.

The Payload MUST contain at least one Topic Filter and Subscription Options pair [MQTT-3.8.3-2]. A SUBSCRIBE packet with no Payload is a Protocol Error. Refer to section 0 for information about handling errors.

3.8.3.1 Subscription Options

Bits 0 and 1 of the Subscription Options represent Maximum QoS field. This gives the maximum QoS level at which the Server can send Application Messages to the Client. It is a Protocol Error if the Maximum QoS field has the value 3.

Bit 2 of the Subscription Options represent No Local option. If the value is 1, Application Messages MUST NOT be forwarded to a connection with a ClientId equal to the ClientId of the publishing connection [MQTT-3.8.3-3]. It is a Protocol Error to set the No Local bit to 1 on a Shared Subscription and the Server MUST close the Network Connection [MQTT-3.8.3-4].

Bit 3 of the Subscription Options represent Retain As Published option. If 0, Application Messages forwarded using this subscription keep the RETAIN flag they were published with. If 1, the RETAIN flag in the PUBLISH packet indicates whether it came from a retained source or is a new publication.

Bits 4 and 5 of the Subscription Options represent the Retain Handling option. This options specifies whether retained messages are sent when the subscription is established. This does not affect the sending of retained messages at any point after the subscribe. If there are no retained messages matching the Topic Filter all of these values act the same. The values are:

0 = Send retained messages at the time of the subscribe

1 = Send retained messages at subscribe only if the subscription does not currently exist
2 = Do not send retained messages at the time of the subscribe
It is a Protocol Error to send a Retain Handling value of 3.

Bits 6 and 7 of the Subscription Options byte 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 [MQTT-3.8.3-5].

Non-Normative comment

The No Local and Retain As Published subscription options can be used to implement bridging where the Client is sending the message on to another Server.

Not sending retained messages for an existing subscription is useful when a reconnect is done and the Client is not certain whether the subscriptions were completed in the previous connection to the Session.

Not sending stored retained messages because of a new subscription is useful where a Client wishes to receive change notifications and does not need to know the initial state.

Figure 3-20– 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							
Subscription Options								
	Reserved		Retain Handling		RAP	NL	QoS	
byte N+1	0	0	X	X	X	X	X	X

3.8.3.2 Payload Non-Normative example

Figure 3.21 - Payload byte format non-normative example shows the Payload for the SUBSCRIBE Packet

Table 3-9 - Payload non-normative example

Topic Name	“a/b”
Subscription Options	0x01
Topic Name	“c/d”
Subscription Options	0x02

Figure 3-21 - 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
Subscription Options									
byte 6	Subscription Options(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
Subscription Options									
byte 12	Subscription Options(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 a Non-Shared Subscription’s Topic Filter for the current Session then it MUST completely replace that existing Subscription with a new Subscription [MQTT-3.8.4-3]. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its maximum QoS value could be different. If the Retain Handling option is 0, 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 Non-Shared Topic Filter that is not identical to any Topic Filter for the current Session, a new Non-Shared Subscription is created. If the Retain Handling option is not 2, all matching retained messages are sent to the Client.

If a Server receives a Topic Filter that is identical to the Topic Filter for a Shared Subscription that already exists on the Server, the Session is added as a subscriber to that Shared Subscription. No retained messages are sent.

If a Server receives a Shared Subscription Topic Filter that is not identical to any existing Shared Subscription's Topic Filter, a new Shared Subscription is created. The Session is added as a subscriber to that Shared Subscription. No retained messages are sent.

Refer to section 4.8 for more details on Shared Subscriptions.

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/Subscription Option pair [MQTT-3.8.4-6]. This Return Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed [MQTT-3.8.4-7]. 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 [MQTT-3.8.4-8]. 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.

Non-Normative comment

If a subscribing Client has been granted maximum QoS 1 for a particular Topic Filter, then a QoS 0 Application Message matching the filter is delivered to the Client at QoS 0. This means that at most one copy of the 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.

The Subscription Identifiers are part of the session state in the server and return to the client receiving a matching PUBLISH packet. They are removed from the Session state in the Server when the Server receives an UNSUBSCRIBE packet, when the Server receives a SUBSCRIBE packet from the client for the same Topic Filter but with a different Subscription Identifier or with no Subscription Identifier, or when the Server sends Session Present 0 in a CONNACK packet.

The Subscription Identifiers do not form part of the session state in the client. In a useful implementation, a client will associate the Subscription Identifiers with other client side state, this state is typically removed when the client unsubscribes, when the client subscribes for the same Topic Filter with a different identifier or no identifier, or when the client receives Session Present 0 in a CONNACK packet.

The server is not obliged to use the same set of Subscription Identifiers in are transmitted PUBLISH packet. If the client remade a subscription after the initial transmission of a PUBLISH packet and used a different Subscription Identifier, then the server is allowed to use the identifiers from the first transmission in any retransmission. Alternatively, the Server is allowed to use the new identifiers during a retransmission. The Server is not allowed to revert to the old identifier if it has sent a PUBLISH packet containing the new one.

Non-Normative comment

Usage scenarios, for illustration if Subscription Identifiers.

- The Client implementation indicates via its programming interface that a publication matched more than one subscription. The client implementation generates a new identifier each time a subscription is made. If the returned publication carries more than one Subscription Identifier, then the publication matched more than one subscription.

- The Client implementation allows the subscriber to direct messages to a callback associated with the subscription. The Client implementation generates an identifier which uniquely maps the identifier to the callback. When a publication is received it uses the Subscription Identifier to determine which callback is driven.

- The Client implementation returns the topic string used to make the subscription to the application when it delivers the published message. To achieve this the client generates an identifier which uniquely identifies the Topic Filter. When a publication is received the Client implementation uses the identifiers to look up the original Topic Filters and return them to the client application.

- A gateway forwards publications received from a Server to Clients that have made subscriptions to the gateway. The gateway implementation maintains a map of each unique Topic Filter it receives to the set of clientId, subscriptionId pairs that it also received. It generates a unique identifier for each Topic Filter that it forwards to the server. When a publication is received the gateway uses the Subscription Identifiers it received from the Server to look up the Client Identifier, Subscription Identifier pairs associated with them. It adds these to the PUBLISH packets it sends to the Clients. If the upstream Server sent multiple PUBLISH packets because the message matched multiple subscriptions, then this behavior is mirrored to the Clients.

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 or the error which was found for each Subscription that was requested by the SUBSCRIBE.

3.9.1 Fixed Header

Figure 3-22 - 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 plus the length of the Payload encoded as a Variable Byte Integer.

3.9.2 Variable Header

The Variable Header of the SUBACK Packet contains the following fields in the order: the Packet Identifier from the SUBSCRIBE Packet that is being acknowledged, Property Length, and the Properties.

3.9.2.1 Property Length

The length of Properties in the SUBACK packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4, there is no Property Length and the value of 0 is used.

3.9.2.2 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT use this Property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.9.2-1]. It is a Protocol Error to include the Reason String more than once.

3.9.2.3 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the SUBACK beyond the Maximum Packet Size specified by the session partner [MQTT-3.9.2-2]. This property may be included more than once.

Figure 3-23 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].

Table 3-10 - Subscribe Return Codes

Value	Hex	Return Code name	Description
0	0x00	Granted QoS 0	The subscription is accepted and the maximum QoS sent will be QoS 0. This might be a lower QoS than was requested.
1	0x01	Granted QoS 1	The subscription is accepted and the maximum QoS sent will be QoS 1. This might be a lower QoS than was requested.
2	0x02	Granted QoS 2	The subscription is accepted and any received QoS will be sent to this subscription
128	0x80	Unspecified error	The subscription is not accepted and the Server either does not wish to reveal the reason or none of the other Return Codes apply.
131	0x83	Implementation specific error	The SUBSCRIBE is valid but the Server does not accept it.
135	0x87	Not authorized	The Client is not authorized to make this subscription
143	0x8F	Topic Filter invalid	The Topic Filter is correctly formed but is not allowed for this client.
145	0x91	Packet Identifier in use	The specified Packet Identifier is already in use
151	0x97	Quota exceeded	An implementation imposed limit has been exceeded.
158	0x9E	Shared Subscription not supported	The Server does not support Shared Subscriptions for this Client
161	0xA1	Subscription Identifiers not supported	The Server does not support Subscription Identifiers; the subscription is not accepted
162	0xA2	Wildcard subscriptions not supported	The Server does not support Wildcard subscription; the subscription is not accepted

The Server MUST send one of the Return Codes listed in Table 3-10 - Subscribe Return Codes for each subscription received [MQTT-3.9.3-2].

3.9.4 Payload Non-Normative example

Figure 3.27 - Payload byte format non-normative example shows the Payload for the SUBACK packet.

Table 3-11 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 request

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 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, encoded as a Variable Byte Integer.

3.10.2 Variable Header

The Variable Header of the UNSUBSCRIBE Packet contains the Packet Identifier. Section 2.2.1 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 [MQTT-3.10.3-1] as defined in section **Error! Reference source not found.**, packed contiguously.

The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter [MQTT-3.10.3-2]. An UNSUBSCRIBE packet with no Payload is a Protocol Error. Refer to section 0 for information about handling errors.

Payload Non-Normative example

Figure 3.30 - Payload byte format non-normative example show the Payload for the UNSUBSCRIBE packet briefly described in Table 3.7 - Payload non-normative example.

Table 3.7 - 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 MUST be deleted [MQTT-3.10.4-1], otherwise no additional processing occurs.

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

If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST process 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							

Remaining Length field

This is the length of the Variable Header plus the length of the Payload encoded as a Variable Byte Integer.

3.11.2 Variable Header

The Variable Header of the UNSUBACK Packet the following fields in the order: the Packet Identifier from the UNSUBSCRIBE Packet that is being acknowledged, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

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							

3.11.2.1 Property Length

The length of the Properties in the UNSUBACK packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

3.11.2.2 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT use this Property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.11.2-1]. It is a Protocol Error to include the Reason String more than once.

3.11.2.3 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the UNSUBACK beyond the Maximum Packet Size specified by the session partner [MQTT-3.11.2-2]. This property may be included more than once.

3.11.3 Payload

The Payload contains a list of Return Codes. Each Return Code corresponds to a Topic Filter in the UNSUBSCRIBE packet being acknowledged. The order of Return Codes in the UNSUBACK packet MUST match the order of Topic Filters in the UNSUBSCRIBE packet [MQTT-3.11.3-1].

The values for the one byte unsigned Unsubscribe Return Codes are listed in Table 3-12 - Unsubscribe Return Codes. The Server MUST use one of the Return Code values from this table [MQTT-3.11.3-2].

Table 3-12 - Unsubscribe Return Codes

Value	Hex	Return Code name	Description
0	0x00	Success	The subscription is deleted
17	0x11	No subscription existed	No matching subscription existed
128	0x80	Unspecified error	The unsubscribe could not be completed and the Server either does not wish to reveal the reason or none of the other Return Codes apply.
131	0x83	Implementation specific error	The UNSUBSCRIBE is valid but the Server does not accept it.
135	0x87	Not authorized	The client is not authorized to unsubscribe
143	0x8F	Topic Filter invalid	The Topic Filter is correctly formed but is not allowed for this client.
145	0x91	Packet Identifier in use	The specified Packet Identifier is already in use

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 MQTT 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. Refer to section 0 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. Refer to section 0 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

3.13.2 Variable Header

The PINGRESP packet has no Variable Header.

3.13.3 Payload

The PINGRESP packet has no Payload.

3.13.4 Actions

The Client takes no action on receiving this packet

3.14 DISCONNECT – Disconnect notification

The DISCONNECT packet is the final MQTT Control Packet sent from the Client or the Server. It indicates the reason why the Network Connection is being closed. The Client or Server MAY send a DISCONNECT packet before closing the Network Connection. If the Client closes the Network Connection without first sending a DISCONNECT and the Connection has a Will Message, the Will Message is published.

A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Return Code of less than 128 [MQTT-3.14.0-1].

3.14.1 Fixed Header

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							

The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a DISCONNECT packet with a Return code of 0x81 (Malformed Packet) as described in section 4.13 and MUST close the Network Connection [MQTT-3.14.1-1].

Remaining Length field

This is the length of the Variable Header encoded as a Variable Byte Integer.

3.14.2 Variable Header

The Variable Header of the DISCONNECT Packet contains the following fields in the order: Disconnect Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

3.14.2.1 Disconnect Return Code

Byte 1 in the Variable Header is the Disconnect Return Code. If the Remaining Length is less than 1 the value of 0x00 (Success) is used.

The values for the one byte unsigned Disconnect Return Code field are listed in Table 3-13 – Disconnect Return Code values.

Table 3-13 – Disconnect Return Code values

Value	Hex	Return Code name	Sent by	Description
0	0x00	Success	Client	Close the connection normally. Do not send the Will Message.
4	0x04	Disconnect with Will Message	Client	The client wishes to disconnect but requires that the Server also publishes its Will Message.
128	0x80	Unspecified error	Client or Server	The Connection is closed but the sender either does not wish to reveal the reason, or none of the other Return Codes apply.
129	0x81	Malformed Packet	Client or Server	The received packet does not conform to this specification.
130	0x82	Protocol Error	Client or Server	An unexpected or out of order packet was received.
131	0x83	Implementation specific error	Client or Server	The packet received is valid but cannot be processed by this implementation
135	0x87	Not authorized	Server	The request is not authorized
137	0x89	Server busy	Server	The Server is busy and cannot continue processing this Client.
139	0x8B	Server shutting down	Server	The Server is shutting down.
141	0x8D	Keep Alive timeout	Server	The Connection is closed because no packet has been received for 1.5 times the Keepalive time.
142	0x8E	Session taken over	Server	Another Connection using the same ClientId has connected causing this Connection to be closed.
143	0x8F	Topic Filter invalid	Server	The Topic Filter is correctly formed, but is not accepted by this Sever.
144	0x90	Topic Name invalid	Client or Server	The Topic Name is correctly formed, but s not accepted by this Client or Server

147	0x93	Receive Maximum exceeded	Client or Server	The Client or Server has received more than Receive Maximum publication for which it has not sent PUBACK or PUBCOMP.
148	0x94	Topic Alias invalid	Client or Server	The Client or Server has received a PUBLISH packet containing a Topic Alias which is greater than the Maximum Topic Alias it sent in the CONNECT or CONNACK packet.
149	0x95	Packet too large	Client or Server	The packet size is greater than Maximum Packet Size for this Client or Server
150	0x96	Message rate too high	Client or Server	The rate of publish is too high
151	0x97	Quota exceeded	Client or Server	An implementation imposed limit has been exceeded
152	0x98	Administrative action	Client or Server	The Connection is closed due to an administrative action.
153	0x99	Payload format invalid	Client or Server	The payload format does not match the one specified by the Payload Format Indicator.
154	0x9A	Retain unavailable	Server	The Server has specified Retain unavailable in the CONNACK
155	0x9B	Maximum QoS	Server	The Client specified a QoS greater then the QoS specified in a Maximum QoS in the CONNACK.
156	0x9C	Use another server	Server	The Client should temporarily change its Server
157	0x9D	Server moved	Server	The Server is moved and the client should permanently change its server location.
158	0x9E	Shared Subscription not supported	Server	This Server does not support Shared Subscriptions
159	0x9F	Connection rate exceeded	Server	This connection is closed because the connection rate is too high
160	0xA0	Maximum connect time	Server	The maximum connection time authorized for this connection has been exceeded.
161	0xA1	Subscription Identifiers not supported	Server	The Server does not support Subscription Identifiers; the subscription is not accepted
162	0xA2	Wildcard subscriptions not supported	Server	The Server does not support Wildcard subscription; the subscription is not accepted

The Client or Server sending the DISCONNECT MUST use one of the Return Codes in Table 3-13 – Disconnect Return Code s [MQTT-3.14.2-1]. The Return Code 0x00 (Success) may be sent by using a Remaining Length of 0.

Non-Normative comment

The DISCONNECT packet is used to indicate the reason for a disconnect for cases where there is no acknowledge packet (such as a QoS 0 publish) or when the Client or Server is unable to continue processing the Connection.

The information can be used by the Client to decide whether to retry the connection, and how long it should wait before retrying the connection.

3.14.2.2 Property Length

The length of Properties in the DISCONNECT packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 2, a value of 0 is used.

3.14.2.3 Session Expiry Interval

17 (0x11) Byte, Identifier of the Session Expiry Interval.

Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol Error to include the Session Expiry Interval more than once.

If the Session Expiry Interval is absent, the Session Expiry Interval in the CONNECT packet is used.

If the Session Expiry Interval in the CONNECT packet was zero, then it is a Protocol Error to set a non-zero Session Expiry Interval in the DISCONNECT packet. If such a non-zero Session Expiry Interval is received by the Server, it does not treat it as a valid DISCONNECT packet. The Server uses DISCONNECT with Return Code 0x82 (Protocol Error) as described in section 4.13. The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server [MQTT-3.14.2-2]. If a non-zero Session Expiry Interval is received by the use DISCONNECT with Return Code 0x82 (Protocol Error) as described in section 4.13.

3.14.2.4 Reason String

31 (0x1F) Byte, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason for the disconnect. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender MUST NOT use this Property if it would increase the size of the DISCONNECT packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.14.2-3]. It is a Protocol Error to include the Reason String more than once.

3.14.2.5 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the DISCONNECT beyond the Maximum Packet Size specified by the session partner [MQTT-3.14.2-4]. This property may be included more than once.

3.14.2.6 Server Reference

28 (0x1C) Byte, Identifier of the Server Reference.

Followed by a UTF-8 Encoded String which can be used by the Client to identify another Server to use. It is a Protocol Error to include the Server Reference more than once.

The Server sends DISCONNECT including a Server Reference and Return Code 0x9C (Use another server) or 0x9D (Server moved) as described in section 4.13.

Refer to section 4.11 Server Redirection for information about how Server Reference is used.

Figure 3-24 DISCONNECT packet Variable Header non-normative example

	Description	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Disconnect Return Code									
byte 1		0	0	0	0	0	0	0	0
Properties									
byte 2	Length (5)	0	0	0	0	0	1	1	1
byte 3	Session Expiry Interval identifier (17)	0	0	0	1	0	0	0	1
byte 4	Session Expiry Interval (0)	0	0	0	0	0	0	0	0
byte 14		0	0	0	0	0	0	0	0
byte 15		0	0	0	0	0	0	0	0
byte 16		0	0	0	0	0	0	0	0

3.14.3 Actions

After sending a DISCONNECT packet the sender:

- MUST NOT send any more MQTT Control Packets on that Network Connection [MQTT-3.14.3-1].
- MUST close the Network Connection [MQTT-3.14.3-2].

On receipt of DISCONNECT with a Return Code of 0x00 (Success) or 0x04 (Disconnect with Will Message) the Server:

- MUST discard any Will Message associated with the current Connection without publishing it [MQTT-3.14.3-3], as described in section 3.1.2.5.

On receipt of DISCONNECT, the receiver:

- SHOULD close the Network Connection.

3.14.4 Payload

The DISCONNECT packet has no Payload.

3.15 AUTH – Authentication exchange

An AUTH packet is sent from Client to Server or Server to Client as part of an extended authentication exchange, such as challenge / response authentication. It is a Protocol Error for the Client or Server to send an AUTH packet if the CONNECT packet did not contain an Auth Method.

3.15.1 Fixed Header

Figure 3.35 – AUTH packet Fixed Header

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	MQTT Control Packet type (15)				Reserved			
	1	1	1	1	0	0	0	1
byte 2	Remaining Length							

Bits 3,2,1 and 0 of the Fixed Header of the AUTH packet are reserved and MUST be set to 0, 0, 0 and 1 respectively. The Client or Server MUST treat any other value as malformed and close the Network Connection [MQTT-3.15.1-1].

Remaining Length field

This is the length of the Variable Header encoded as a Variable Byte Integer.

3.15.2 Variable Header

The Variable Header of the AUTH Packet contains the following fields in the order: Authenticate Return Code, Property Length, and the Properties. The rules for encoding Properties are described in section 2.2.3.

3.15.2.1 Authenticate Return Code

Byte 0 in the Variable Header is the Authenticate Return Code. The values for the one byte unsigned Authenticate Return Code field are listed in Table 3-14 Authentication Return Codes. The sender of the AUTH Packet MUST use one of these Return Codes [MQTT-3.15.2-1].

Table 3-14 Authentication Return Codes

Value	**Hex**	**Return Code name**	**Sent by**	**Description**
0	0x00	Success	Server	Authentication is successful
24	0x18	Continue authentication	Client or Server	Continue the authentication with another step
25	0x19	Re-authenticate	Client	Initiate a re-authentication

3.15.2.2 Property Length

The length of Properties in the AUTH packet Variable Header encoded as a Variable Byte Integer.

3.15.2.3 Auth Method

21 (0x15) Byte, Identifier of the Auth Method.

Followed by a UTF-8 Encoded String containing the name of the authentication method. Refer to section 4.12 to understand how extended authentication works.

3.15.2.4 Auth Data

22 (0x16) Byte, Identifier of the Auth Data.

Followed by Binary Data containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. Refer to section 4.12 to understand how extended authentication works.

3.15.2.5 User Property

38 (0x26) Byte, Identifier of the User Property.

Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic information. The sender MUST NOT send this property if it would increase the size of the AUTH beyond the Maximum Packet Size specified by the session partner [MQTT-3.15.2-2]. This property may be included more than once.

3.15.3 Payload

The AUTH packet has no Payload.

3.15.4 Actions

Refer to section 4.12 for a description of how extended authentication works.

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 replaced by a Client or the retained message expires. A Server MAY discard QoS=0 retained messages at any time. Refer to section 3.3.1.3 concerning the handling of retained messages.

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 [RFC0793]. TCP/IP can be used for MQTT 3.1.1 and MQTT 5. 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 and DUP flag set to 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 2.2.1 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 [MQTT-4.3.2-1].
- MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to 0 [MQTT-4.3.2-2].
- MUST treat the PUBLISH packet as "unacknowledged" until it has received the corresponding PUBACK packet from the receiver. Refer to section **Error! Reference source not found.** for a discussion of unacknowledged messages [MQTT-4.3.2-3].

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 [MQTT-4.3.2-4].
- 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-4.3.2-5].

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

Sender Action	MQTT 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 2.2.1 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 [MQTT-4.3.3-1].
- MUST send a PUBLISH packet containing this Packet Identifier with QoS 2 and DUP flag set to 0 [MQTT-4.3.3-2].
- MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver [MQTT-4.3.3-3]. Refer to section **Error! Reference source not found.** for a discussion of unacknowledged messages.
- MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Return Code value less than 128. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet [MQTT-4.3.3-4].
- MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver [MQTT-4.3.3-5].
- MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet [MQTT-4.3.3-6].
- MUST NOT apply Publication expiry if a PUBLISH packet has been sent [MQTT-4.3.3-7].

The Packet Identifier becomes available for reuse once the Sender has received the PUBCOMP packet or a PUBREL with a Return Code of 128 or greater.

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 [MQTT-4.3.3-8].
- After it has sent a PUBREC with a Return Code of 128 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication [MQTT-4.3.3-9].
- 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 [MQTT-4.3.3-10].
- MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL [MQTT-4.3.3-11].

- 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-4.3.3-12].
- MUST NOT apply Publication expiry until after PUBCOMP has been sent [MQTT-4.3.3-13].

4.4 Message delivery retry

When a Client reconnects with CleanStart set to 0 and a session is present, both the Client and Server MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their original Packet Identifiers. This is the only circumstance where a Client or Server is REQUIRED to resend messages. Clients and Servers MUST NOT resend messages at any other time [MQTT-4.4.0-1].

If PUBACK or PUBREC is received containing a failure code (> 0x80) the corresponding PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted [MQTT-4.4.0-2].

Non-Normative comment
Historically, retransmission of MQTT Control Packets was required to overcome data loss on some older TCP networks. This might remain a concern where MQTT 5 implementations are to be deployed in such environments.

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

Sender Action	MQTT Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP=0 <Packet Identifier>		
	---------->	
		Store <Packet Identifier> then Initiate onward delivery of the Application Message[1]
		PUBREC <Packet Identifier><Return Code>
	<----------	
Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	---------->	
		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. However, the receiver needs to perform all

checks for conditions which may result in a forwarding failure (e.g. quota exceeded, authorization, etc.) and return the appropriate error response code in the PUBREC because forwarding is NOT postponed until the arrival of the PUBREL.

Figure 4.3 shows the sequence of QoS 2 receive message handling. The receiver needs to perform all checks when the PUBLISH arrives and return the appropriate success for failure code in the PUBREC. .

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 [MQTT-4.5.0-1]. Matching rules are defined in section 4.7.

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 [MQTT-4.6.0-6]. 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-4.6.0-7].

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 a Client 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-4.7.0-1].

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-4.7.1-1].

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-4.7.1-2]. 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 or trailing '/' creates a distinct Topic Name or Topic Filter
- A Topic Name or Topic Filter consisting only of the '/' character is valid
- Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode] [MQTT-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]. Refer to section **Error! Reference source not found.**

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 does 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 Subscriptions

MQTT provides two kinds of Subscription, Shared and Non-Shared.

4.8.1 Non-Shared Subscriptions

A Non-Shared Subscription is associated only with the MQTT Session that created it. Each Subscription includes a Topic Filter, indicating the topic(s) for which messages are to be delivered on that Session, and Subscription Options. The Server is responsible for collecting messages that match the filter and transmitting them on the Session's MQTT connection if and when that connection is active.

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

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

4.8.2 Shared Subscriptions

A Shared Subscription can be associated with multiple subscribing MQTT Sessions. Like a Non-Shared Subscription, it has a Topic Filter and a Subscription Options; however, a publication that matches its Topic Filter is only sent to one of its subscribing Sessions. Shared Subscriptions are useful where several consuming clients share in the processing of the publications in parallel.

A Shared Subscription is identified using a special style of Topic Filter. The format of this filter is:

```
$share/{ShareName}/{filter}
```

- $share is a literal string that marks the Topic Filter as being a Shared Subscription Topic Filter.
- {ShareName} is a character string that does not include "/", "+" or "#"
- {filter} The remainder of the string has the same syntax and semantics as a Topic Filter in a non-shared subscription. Refer to section **4.7.**

A Shared Subscription's Topic Filter MUST start with $share/ and MUST contain a ShareName that is at least one character long [MQTT-4.8.2-1]. The ShareName MUST NOT contain the characters "/", "+" or "#", but MUST be followed by a "/" character. This "/" character MUST be followed by a Topic Filter [MQTT-4.8.2-2] as described in section 4.7.

Non-Normative comment

Shared Subscriptions are defined at the scope of the MQTT Server, rather than of a Session. A ShareName is included in the Shared Subscription's Topic Filter so that there can be more than one Shared Subscription on a Server that has the same {filter} component. An application could choose to use the ShareName to represent the group of subscribing Sessions that are sharing the subscription, but it is not obliged to do this.

Examples:

- Shared subscriptions "$share/consumer1/sport/tennis/+" and "$share/consumer2/sport/tennis/+" are distinct shared subscriptions and so can be associated with different groups of Sessions. Both of them match the same topics as a non-shared subscription to sport/tennis/+ .

  If a message were to be published that matches sport/tennis/+ then a copy would be sent to exactly one of the Sessions subscribed to $share/consumer1/sport/tennis/+ , a separate copy of the message would be sent to exactly one of the Sessions subscribed to $share/consumer2/sport/tennis/+ and further copies would be sent to any clients with non-shared subscriptions to sport/tennis/+

- Shared subscription "$share/consumer1//finance" matches the same topics as a non-shared subscription to /finance.

  Note that "$share/consumer1//finance" and "$share/consumer1/sport/tennis/+" are distinct shared subscriptions, even though they have the same ShareName. While they might be related in some way, no specific relationship between them is implied them having the same ShareName.

A Shared Subscription is created by using a Shared Subscription Topic Filter in a SUBSCRIBE request. So long as only one Session subscribes to a particular Shared Subscription, the shared subscription behaves like a non-shared subscription, except that:

- The $share and {ShareName}portions of the Topic Filter are not taken into account when matching against publications.

- No Retained Messages are sent to the Session when it first subscribes. It will be sent other matching messages as they are published.

Once a Shared Subscription exists, it is possible for other Sessions to subscribe with the same Shared Subscription Topic Filter. The new Session is associated with the Shared Subscription as an additional subscriber. Retained messages are not sent to this new subscriber. Each subsequent Application Message that matches the Shared Subscription is now sent to one and only one of the Sessions that are subscribed to the Shared Subscription.

A Session can explicitly detach itself from a Shared Subscription by sending an UNSUBSCRIBE Packet that contains the full Shared Subscription Topic Filter. Sessions are also detached from the Shared Subscription when they terminate.

A Shared Subscription lasts for as long as it is associated with at least one Session (i.e. a Session that has issued a successful SUBSCRIBE request to its Topic Filter and that has not completed a corresponding UNSUBSCRIBE). A Shared Subscription survives when the Session that originally created it unsubscribes, unless there are no other Sessions left when this happens. A Shared Subscription ends, and any undelivered messages associated with it are deleted, when there are no longer any Sessions subscribed to it.

Notes on Shared Subscriptions

- If there's more than one Session subscribed to the Shared Subscription, the Server implementation is free to choose, on a message by message basis, which Session to use and what criteria it uses to make this selection.

- Different subscribing Clients are permitted to ask for different Requested QoS levels in their SUBSCRIBE packets. The server decides which Maximum QoS to grant to each Client, and it is permitted to grant different Maximum QoS levels to different subscribers. When sending an Application Message to a Client, the Server MUST respect the granted QoS for the Client's subscription [MQTT-4.8.2-3], in the same that it does when sending a message to a Non-Shared Subscriber.

- If the Server is in the process of sending a QoS 2 message to its chosen subscribing Client and the connection to that client breaks before delivery is complete, the Server MUST complete the delivery of the message to that client when it reconnects [MQTT-4.8.2-4] as described in section 4.3.3. If the Client's Session terminates before the client reconnects, the Server MUST NOT send the Application Message to any other subscribed Client [MQTT-4.8.2-5].

- If the Server is in the process of sending a QoS 1 message to its chosen subscribing client and the connection to that client breaks before the Server has received an acknowledgement from the client, the Server MAY wait for the client to reconnect and retransmit the message to that Client. If the Client's Session terminates before the Client reconnects, the Server SHOULD send the Application Message to another Client that is subscribed to the same Shared Subscription. It MAY attempt to send the message to another Client as soon as it loses its connection to the first Client.

- If a Client responds with a negative acknowledgement to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber [MQTT-4.8.2-6].

- A Client is permitted to submit a second SUBSCRIBE request to a Shared Subscription on a Session that's already subscribed to that Shared Subscription. For example, it might for example do this to change the Requested QoS for its subscription or because it was uncertain that the previous subscribe completed before the previous connection was closed. This does not increase the number of times that the Session is associated with the Shared Subscription, so the Session will leave the Shared Subscription on its first UNSUBSCRIBE.

- Each Shared Subscription is independent from any other. It is possible to have two Shared Subscriptions with overlapping filters. In such cases a message that matches both Shared Subscriptions will be processed separately by both of them. If a Client has a Shared Subscription and a Non-Shared subscription and a message matches both of them, the Client will receive a copy of the message by virtue of it having the Non-Shared Subscription. A second copy of the message will be delivered to one of the subscribers to the Shared Subscription, and this could result in a second copy being sent to this Client.

4.9 Flow Control

Clients and Servers may control the number of unacknowledged PUBLISH packets they receive by sending a Receive Maximum value as described in section **Error! Reference source not found.** and 3.2.2.5. The Receive Maximum establishes a quota which is used to limit the number of PUBLISH QOS > 0 packets which can be sent without receiving an PUBACK (for QoS 1) or PUBCOMP (for QoS 2). The PUBACK and PUBCOMP replenish the quota in the manner described below.

When a Client or Server receives a Receive Maximum value, it sets its send quota to that value. It may use a smaller value, but the value chosen MUST be in range of [1...Receive Maximum] [MQTT-4.9.0-1]. If the Receive Maximum value is missing from the CONNECT or CONNACK, the maximum value of 65535 is used. It is a Protocol Error to send a Receive Maximum value of zero.

Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the quota. If the quota reaches zero, the Client or Server MUST NOT send any more PUBLISH packets with QoS > 0 [MQTT-4.9.0-2]. It MAY continue to send PUBLISH packets with QoS 0, or it MAY choose to suspend sending these as well. The client and server MUST continue to process and respond to all other command packets even if the quota is zero [MQTT-4.9.0-3].

Each time a PUBACK or PUBCOMP packet is received, the quota is incremented by 1. The increment is applied regardless of whether the PUBACK or PUBCOMP carried an error code. The quota is not incremented if it is already equal to Receive Maximum. In the event of QoS 2 retransmission, it is possible for a PUBREL to be sent, causing a duplicate PUBCOMP to be received. If applying the quota update would cause the quota to exceed the original value, it is not applied. In this way, the maximum level is 'clamped'.

See sections **Error! Reference source not found.** and 3.2.2.5 for a description of how Clients and Servers react if they are sent more PUBLISH packets than the quota allows.

The quota and Receive Maximum value are not preserved across Network Connections, and are re-initialized with each new Network Connection as described above. They are not part of the session state.

4.10 Request / Response

A Client sends a Request Message by publishing an Application Message which has a Response Topic set as described in section 3.3.2.7. The Request optionally includes a Correlation Data as described in section3.3.2.8.

4.10.1 Basic Request Response (non-normative)

In the Request/Response interaction, one MQTT Client, the Requester, publishes a Request Message to a topic. An Application Message with a Response Topic is a Request Message. Another MQTT Client (the Responder) has subscribed to this topic and receives the Request Message. There may be multiple Responders subscribed to the topic or there may be none. The Responder takes the appropriate action based on the Request Message, and then publishes a Response Message to the Response Topic. The Requestor commonly subscribes to the Response Topic and thereby receives the Response Message. However, some other Client might be subscribed to the Response Topic and the Response Message will be received and processed by that Client. As with the Request Message, the topic on which the Response Message is sent may be subscribed by multiple Clients, or by none.

If the Request Message contains a Correlation Data, the Responder copies it to the Response Message and this is used by the receiver of the Response Message to associate the Response Message with the Request Message. The Response Message does not include the Response Topic.

The Server forwards the Response Topic and Correlation Data Properties in the Request Message, and the Correlation Data in the Response Message. The Server treats the Request Message and the Response Message like any other Application Message.

The requestor normally subscribes to the Response Topic before publishing a Request Message. If there are no subscribers to the Response Topic when the Response Message is sent, the Response Message will not be delivered to any Client.

The Request Message and Response Message can be of any QoS, and the subscription can be in a Session with a non-zero Session Expiry Interval. It is common to send Request Messages at QoS 0 and only when the Responder is expected to be connected. However, this is not required.

The Responder can use a Shared Subscription to allow for a pool of responding Clients. Note however that when using Shared Subscriptions that the order of message delivery is not guaranteed between multiple Clients.

It is the responsibility of the Requestor to make sure it has the necessary authority to publish to the request topic, and to subscribe to the Response Topic. It is the responsibility of the Responder to make sure it has the authority to subscribe to the request topic and publish to the Response Topic. While topic authorization is outside of this specification, it is recommended that Servers implement such authorization.

4.10.2 Determining a Response Topic Value (Non-Normative)

Requesters can determine a Response Topic to use in any manner they choose including local configuration. In many cases, it is desirable that the Response Topic used by a Client be unique to that Client. As the Requestor and Responder commonly need to be authorized to these topics, it can be an authorization problem to use randomized Topic Name.

This specification defines a mechanism to aid the Client to determine the Response Topic to use. This mechanism is optional for both the Client and the Server. At connect time, the Client requests that the Server send a Response Information by setting the Request Response Information in the CONNECT packet. The Response Information is a UTF-8 Encoded String sent in the CONNACK packet. This specification does not define the contents of that string.

A common use of this is to pass a globally unique portion of the topic tree which is reserved for this Client for at least the lifetime of its Session. It is normal to use this as the root of a topic tree for a particular client. For the Server to return this information, it normally needs to be correctly configured. Using this mechanism allows this configuration to be done once in the Server rather than in each Client.

Refer to section 3.2.2.17**Error! Reference source not found.** for the definition of the Response Information.

4.11 Server redirection

A Server can request that the Client uses another Server by sending CONNACK or DISCONNECT with Return Codes 0x9C (Use another server), or 0x9D (Server moved) as described in section 4.13. When sending one of these Return Codes, the Server MAY also send a Server Reference to indicate the location of the Server or Servers the Client SHOULD use.

The Return Code 0x9C (Use another server) specifies that the Client SHOULD temporarily change to using another server. The other Server is either already known to the Client, or is specified using a Server Reference.

The Return Code 0x9D (Server moved) specifies that the Client SHOULD permanently change to using another server. The other Server is either already known to the Client, or is specified using a Server Reference.

The Server Reference is a UTF-8 Encoded String. The value of this string is a space separated list of references. The format of references is not specified here.

Non-Normative comment

It is recommended that each reference consists of a name optionally followed by a colon and a port number.

If the name contains a colon the name string can be enclosed within square brackets (“[“ and ‘]”). A name enclosed by square brackets must not contain the right square bracket (“]”) character. This is used to represent an IPv6 literal address which uses colon separators.

This is a simplified version of an URI authority as described in [RFC3986].

The name within a Server Reference commonly represents a host name, DNS name [RFC1035], SRV name [RFC2782] , or literal IP address.

The value following the colon separator is commonly a port number in decimal. This is not needed where this information comes from the name resolution (such as with SRV) or is defaulted.

Non-Normative comment

Examples of the Server Reference are:

```
    myserver.xyz.org
    myserver.xyz.org:8883
    10.10.151.22:8883 [fe80::9610:3eff:fe1c]:1883
```

The Server is not required to give a Server Reference, and the Client is not required to follow a Server Reference. This feature can be used to allow for load balancing, server relocation, and client provisioning to a server.

4.12 Enhanced authentication

The MQTT CONNECT packet provides data for basic authentication of a Network Connection using the User Name and Password fields. While these fields are named for a simple password authentication, they can be used to carry other forms of authentication such as passing a token as the Password.

Enhanced authentication extends this basic authentication to include challenge / response style authentication. This enhanced authentication is done by exchanging AUTH packets between the Client and the Server between the CONNECT and CONNACK.

The implementation of enhanced authentication is optional for both Clients and Servers. If the Client does not include an Auth Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Auth Method in the CONNACK packet [MQTT-4.12.0-1]. If the Server does not send an Auth Method on an AUTH or CONNACK packet, the Client MUST NOT send an AUTH packet to the Server [MQTT-4.12.0-2].

To begin an enhanced authentication, the Client includes an Auth Method in the CONNECT packet. This specifies the authentication method to use. If the Server does not support the Auth Method supplied by the Client, it MAY send a CONNACK with a Return Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in section 4.13, and MUST close the Network Connection [MQTT-4.12.0-3].
If the Client does not include an Auth Method in the CONNECT packet, the Server SHOULD authenticate using the information in the CONNECT packet, TLS session, and Network Connection.

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

Non-Normative comment

The Auth Method is commonly a SASL Mechanism, and using such a registered name aids interchange. However, the Auth Method is not constrained to using registered SASL Mechanisms.

If the Auth Method selected by the Client specifies that the Client sends data first, the Client SHOULD include an Auth Data field in the CONNECT packet. The contents of the Auth Data is defined by the authentication method.

If the Server requires additional information to complete the authorization, it sends an AUTH packet with a Return Code of 0x18 (Continue authentication). It MUST also set the Auth Method to the same value sent by the Client [MQTT-4.12.0-4]. If the authentication method requires the Server to send authentication data to the Client, it is sent in the Auth Data. The Server can fail the authentication at any time. It MAY send a CONNACK with a Return Code of 128 or above as described in section 4.13, and MUST close the Network Connection [MQTT-4.12.0-5].

The Client responds to the Server by sending an AUTH packet with a Return Code of 0x18 (Continue authentication). The Client MUST set the Auth Method to the same value sent by the Server [MQTT-4.12.0-6]. If the authentication method requires the Client to send authentication data for the Server, it is sent in the Auth Data. The Client can close the connection at any point. It MAY send a DISCONNECT packet before doing so.

The Client and Server exchange AUTH packets as required until the Server accepts the authentication by sending a CONNACK with a Return Code of 0. The Server MUST set the Auth Method in the CONNACK to the Auth Method that the Client set on the CONNECT [MQTT-4.12.0-7]. If the acceptance of the authentication requires data to be sent to the Client, it is sent in the Auth Data.

Non-Normative example showing a SCRAM challenge

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

Non-Normative example showing a Kerberos challenge

- Client to Server CONNECT Auth Method="GS2-KRB5"
- Server to Client AUTH rc=0x18 Auth Method="GS2-KRB5"
- Client to Server AUTH rc=0x18 Auth Method="GS2-KRB5" Auth Data=initial context token
- Server to Client AUTH rc=0x18 Auth Method="GS2-KRB5" Auth Data=reply context token
- Client to Server AUTH rc=0x18 Auth Method="GS2-KRB5"
- Server to Client CONNACK rc=0 Auth Method="GS2-KRB5" Auth Data=outcome of authentication

4.12.1 Re-authentication

After receiving a CONNACK which contains an Auth Method, the Client can initiate a re-authentication by sending an AUTH packet with a Return Code of 0x19 (Re-authentication). The Client MUST set the Auth Method to the same value as the Auth Method originally used to authenticate the Network Connection [MQTT-4.12.1-1]. If the authentication method requires client data first, this AUTH packet contains the first piece of authentication data as the Auth Data.

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

If the re-authentication fails, the Server SHOULD send DISCONNECT with an appropriate Return Code as described in section 4.13, and MUST close the Network Connection [MQTT-4.12.1-2].

During this re-authentication sequence, the flow of other packets between the Client and Server can continues using the previous authentication.

> **Non-Normative comment:**
>
> The Server may require that the Client not change some of the authentication related fields by failing the re-authentication. For instance, if the Server does not allow the User Name to be changed it can fail any re-authentication which changes the User Name.

4.13 Handling errors

4.13.1 Malformed Packet and Protocol Errors

Definitions of Malformed Packet and Protocol Errors are contained in section 1.2 Terminology, some but not all, of these error cases are noted throughout the specification. The rigour with which a Client or Server checks an MQTT Control Packet it has received will be a compromise between:

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

If the sender is compliant with this specification it will not send Malformed Packets or cause Protocol Errors. However, if a Client sends MQTT Control Packets before it receives CONNACK, it might cause a Protocol Error because it made an incorrect assumption about the Server capabilities. See section 3.1.4 Response.

The Return Codes used for Malformed Packet and Protocol Errors are:

• 0x81	Malformed Packet
• 0x82	Protocol Error
• 0x93	Receive Maximum exceeded
• 0x95	Packet too large
• 0x9A	Retain not supported
• 0x9B	QoS not supported
• 0x9E	Shared Subscription not supported
• 0xA1	Subscription Identifiers not supported
• 0xA2	Wildcard Subscription not supported

Where Client receives a Malformed Packet or detects a Protocol Error, and a Return Code is mentioned in the specification, it SHOULD close the Network Connection. In the case of an error in a AUTH packet it MAY send a DISCONNECT packet containing the return code, before closing the Network Connection. If the case of an error in any other packet it SHOULD send a DISCONNECT packet containing the return code before closing the Network Connection.

Where Server receives a Malformed Packet or detects a Protocol Error, and a return code is mentioned in the specification, it MUST close the Network Connection [MQTT-4.13.1-1]. In the case of an error in a CONNECT packet it MAY send a CONNACK packet containing the return code, before closing the Network Connection. In the case of an error in any other packet it SHOULD send a DISCONNECT packet containing the return code before closing the Network Connection.

If either the Server or Client omits to check some feature of an MQTT Control Packet, it might fail to detect an error, consequently it might allow data to be damaged. If the Client or Server encounters a Malformed Packet or Protocol Error. it MUST close the Network Connection [MQTT-4.13.1-2]. Before closing the Network Connection, it SHOULD send a DISCONNECT packet containing return code 0x81 (Malformed Packet) or 0x82 (Protocol Error). There are no consequences for other Sessions.

4.13.2 Other errors

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

Acknowledgment packets PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK allow a Return Code of 128 or greater to indicate that the received packet, identified by a Packet Identifier, was in error. There are no consequences for other Sessions or other Packets flowing on the same session.

The CONNACK and DISCONNECT packets allow a Return Code of 128 or greater to indicate that the Network Connection will be closed. If a return code of 128 or greater is specified then the Network Connection MUST be closed whether or not the CONNACK or DISCONNECT is sent [MQTT-4.13.2-1]. Some of these return codes are sent for reasons other than in response to an inbound packet. Sending of one of these return codes does not have consequence for any other Session.

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

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 MQTT 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 [MQTTNIST]. 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 User Name 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

If a client has been successfully authenticated, a server implementation should check that it is authorized before accepting its connection.

Authorization may be based on information provided by the Client such as User Name, the hostname/IP address of the Client, or the outcome of authentication mechanisms.

In particular, the implementation should check that the client is authorized to use the Client Identifier as this gives access to the MQTT Session state (described in section **Error! Reference source not found.**). This authorization check is to protect against the case where one client, accidentally or maliciously, provides a Client Identifier that is already being used by some other client.

An implementation should provide access controls that take place after CONNECT to restrict the client's ability to publish to particular Topics or to subscribe using particular Topic Filters. In particular, an implementation should consider limiting access to Topic Filters that have broad scope, such as the # Topic Filter.

5.4.3 Authentication of the Server by the Client

The MQTT 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 MQTT Control Packets

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

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 MQTT 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. [IEEE8021AR] is a standard for implementing mechanisms to authenticate a device's identity using a cryptographically bound identifier.

5.4.8 Detecting abnormal behaviors

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

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

Server implementations might close the Network Connection of Clients that breach its security rules.

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

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

5.4.9 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 close the Network Connection of Clients and require them to re-authenticate with new credentials.
- Servers may require the Client to re-authenticate using the re-authentication support of enhanced authentication.

Constrained devices and Clients on constrained networks can make use of TLS [RFC5246] session resumption, 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 User Name 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 Subprotocol 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 6.6-1 - 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 MQTT 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.

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 MQTT 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)
- Chapter 7 - Conformance Targets

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.

Appendix A. Acknowledgments

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:

Participants:

- Alex Kritikos (Software AG, Inc.)
- Allan Stockdill-Mander (IBM)
- Andrew Banks (IBM)
- Andrew Schofield (IBM)
- Brian Raymor (Microsoft)
- Christian Götz (dc-square GmbH)
- Christopher Kelley (Cisco Systems)
- Dave Sheehan (Infiswift)
- David Horton (Solace Systems)
- Derek Fu (IBM)
- Dominik Obermaier (dc-square GmbH)
- Dr. Andy Stanford-Clark (IBM)
- Ed Briggs (Microsoft)
- Ian Craggs (IBM)
- James Amsden (IBM)
- Jonathan Levell (IBM)
- Ken Borgendale (IBM)
- Konstantin Dotchkoff (Microsoft)
- Mickael Remond (ProcessOne)
- Nicholas O'Leary (IBM)
- Peter Niblett (IBM)
- Rahul Gupta (IBM)
- Raphael Cohn (Individual)
- Richard Coppen (IBM)
- Senthil Balasubramaniam (Infiswift)
- Shawn McAllister (Solace Systems)
- Stefan Hagen (Individual)
- Stefan Vaillant (Cumulocity)
- Tobias Sommer (Cumulocity)
- William Cox (Individual)

Secretary:

- Ian Craggs (icraggs@uk.ibm.com), IBM

Appendix B. Mandatory Normative Statement (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.

3817 # Appendix C. Revision History

Revision	Date	Editor	Changes Made
[1]	[18th July 2016]	[Andrew Banks]	• [MQTT-249] Add expiry capabilities to MQTT • [MQTT-256] Message Format indication and message metadata in general.TC accepted proposal • [MQTT-269] MQTT-SN Feature: Topic Registration • [MQTT-270] SN Feature: server initiated disconnects • Rename Remaining length datatype to Variable Byte Integer • Introduce two and four-byte integer data types
[2]	[10th August 2016]	[Andrew Banks] [Rahul Gupta]	• [MQTT-249] Add expiry capabilities to MQTT. • [MQTT-263] Simplified State Management. TC accepted proposals. • [MQTT-289] Update the working draft to the new template for MQTT v5 from OASIS
[3]	[25th August 2016]	[Rahul Gupta] [Ken Borgendale]	• [MQTT-236] Consolidate acknowledgements, enable negative acknowledgements • [MQTT-270] Server initiated disconnects • [MQTT-294] Incorrect version number in section 3.1.2.2 Protocol Level
[4]	[6th September 2016]	[Andrew Banks]	• [MQTT-257] Flow Control
[5]	[22nd September 2016]	[Andrew Banks]	• [MQTT-249] Session Expiry • [MQTT-302] WD4: Minor suggestions in sections 2.3.3.X
[6]	[23nd September 2016]	[Andrew Banks]	• Accept all changes, remove markup.
[7]	[26th September 2016]	[Ed Briggs]	• [MQTT-295] Modified 4.4 to prohibit retransmission during a transport connection • [MQTT-257] Flow Control algorithm added
[7]	[28th September 2016]	[Andrew Banks]	• [MQTT-251] Return server assigned client id to client • [MQTT-303] Missing reference to Receive Maximum in Appendix B • [MQTT-290] Session Expiry Will message • [MQTT-269] MQTT-SN Feature: Topic Registration
[7]	[3rd October 2016]	[Ed Briggs]	• [MQTT-236] Added CONNACK Banned

			Error Code • Added QoS Not Supported to PUBACK and PUBREC. • Added Invalid Topic to CONNACK to signify invalid Will Topic • Changed ‘Message Too Long” to “Packet too long based on TC agreement to use packet size, not payload size.
[7]	[3rd October 2016]	[Ken Borgendale]	• [MQTT-197] Request / response (mechanism, section 4.9 not complete) • [MQTT-235] NoLocal • [MQTT-278] Server Keep Alive • [MQTT-284] Enhanced problem determination
[7]	[4th October 2016]	[Ed Briggs]	• [MQTT-301] Added Identifier definition for Retain Unavailable Advertisement • [MQTT-300] Added Identifier definition for Maximum QoS • [MQTT-296] Added sentence requiring minimum size encoded value for variable length integer in section 1.5.5. • [MQTT-287] Added text for single unified packet identifier space
[7]	[5th October 2016]	[Ken Borgendale]	• [MQTT-234] Shared Subscriptions • [MQTT-293] Recommendations for securing an MQTT server
[7]	[6th October 2016]	[Ed Briggs]	• [MQTT-304] User Defined CONNECT Tags • [MQTT-305] User Defined PUBLISH Tags • Defined new UTF-8 String Pair Data Type • Added Identifier 38 (0x26) for User Defined Name-Value Pair
[8]	[7th October 2016]	[Andrew Banks]	• [MQTT-310] Treat invalid topic alias as a Protocol Error
[8]	[8th October 2016]	[Ken Borgendale]	• Fix section numbering and TOC issues, along with other formatting issues.
[8]	[18th October 2016]	[Ken Borgendale}	• [MQTT-260] Try another server • [MQTT-255] Alternate authentication • [MQTT-285] Subscribe options • [MQTT-309] Enhanced Problem determination for ACKS • Editorial changes
[8]	[18th October 2016]	[Ed Briggs]	• [MQTT-314] Simplified String Pair Type
[9]	[25th October 2016]	[Ed Briggs]	• [MQTT-318] ACK Code for quota exceeded • Changed SUBACK code for Shared Subscription Not Supported to 0x9e to remove conflict with 0x97 (Quota

			exceeded) • [MQTT-317] CONNACK Connect rate limit exceeded Return Code added. • [MQTT-286] QoS 2 Delivery now uses what was called Method B. All references to Method A and B are removed and non-normative text regarding checking of errors before returning PUBREC has been added.
[9]	[31st October 2016]	[Ken Borgendale]	• [MQTT-316] Rename Identifier/Value pairs to Properties • [MQTT-319] Re-authentication
[9]	[15th November 2016]	[Andrew Banks]	• [MQTT-323] Comments from Andrew Schofield.
[9]	[24th November 2016]	[Andrew Banks]	• [MQTT-253] Subscription Identifiers. • [MQTT-300] Maximum QoS • Comments from Konstantin Dotchkoff review of WD08
[9]	[30th November 2016]	[Andrew Banks]	• [MQTT-301],[MQTT-300],[MQTT-299] updates in light of [MQTT-311] Common method for handling limits violations.
[10]	[13th December 2016]	[Andrew Banks]	• [MQTT-307] Clarify Handling of DISCONNECT Expiry interval error in WD04 • [MQTT-306] Clarify Handling of Malformed DISCONNECT command in WD04
10	[16th December 2016]	[Ed Briggs]	• [MQTT-299] Maximum Packet Size added to CONNECT and CONNACT with updated text on Protocol Errors and error handling • [MQTT-326] Updated error handling of Maximum QoS and Retain Unavailable to treat violations of advertisements as Protocol Errors. • [MQTT-322] Add Content Type property to PUBLISH. Added definitions.
10	[31st December 2016]	[Andrew Banks]	• [MQTT-327] Words like Malformed Control Packet and Protocol Errors are used randomly.
10	[4th January 2016]	[Andrew Banks]	• [MQTT-328] WD9: Inconsistencies overlaps in Return codes.
10	[4th January 2016]	[Ed Briggs]	• [MQTT-322] Content Type Property. Added non-normative text regarding the use of a MIME string and it (non) interpretation.
10	[4th January 2016]	[Ed Briggs]	• [MQTT-321] User Properties on ACKs. Added user properties to the acks.

10	[4th January 2016]	[Ed Briggs]	• Added text stating that reception of a PUBACK or PUBREC with a failure code removes the corresponding PUBLISH from the retransmission process, and the packet will not be retransmitted.
10	[4th January 2017]	[Rahul Gupta]	• [MQTT-324] Consolidate list of optional server capabilities and review how they are signaled to the client • Appendix A • Hyperlink to Normative and Non-Normative References • Section indexes and other editorial cleanup
11	[6th January 2017]	[Ken Borgendale]	• [MQTT-328] Additional changes • Add a global table of return codes • Normalize text referring to return codes throughout the document • Fix references to User Properties • Review comments and make changes • Normalize indentation of H4
11	[9th January 2017]	[Andrew Banks]	• [MQTT-311] Clearing comments from the V5 specification.
11	[10th January 2017]	[Rahul Gupta]	• [MQTT-332] Consistency 8 bit and one byte • [MQTT-333] Zero, 0 and non-zero • [MQTT-365] Paragraph indentation issue
11	[11th January 2017]	[Ed Briggs]	• [MQTT-337] UNSUBACK no UNSUBACK • [MQTT-338] Typographical: In -> If • [MQTT-346] Misplaced comma • [MQTT-350] Protocol violation -> Protocol Error. • [MQTT-355] Receive Maximum value less than 1 fixed. • [MQTT-360] Variable Header non-normative, bytes 8 and 12 (13) fixed
11	[12th January 2017]	[Ed Briggs]	• [MQTT-361] Client Identifier (ClientID) • [MQTT-375] 4 Byte Integer • [MQTT-391] Typo
11	[12th January 2017]	[Rahul Gupta]	• [MQTT-381] RETAIN Flag Consistency • [MQTT-336] Retain As Published Consistency • [MQTT-389] Minor fix in Request Response • [MQTT-384] Payload Format Indicator corrections • [MQTT-382] Consistency of QoS terminology
11	[13th January 2017]	[Andrew Banks]	• [MQTT-368] Consistency: Session Expiry [Interval]

			• [MQTT-366] Line 1243 : "the is new" and repeating the details of Will Message firing
11	[16th January 2017]	[Andrew Banks]	• [MQTT-363] Repeating the Binary Data representation. • [MQTT-364] byes of Binary Data. • [MQTT-362] Consistency: "this" is the next field.
11	[16th January 2017]	[Ed Briggs]	• [MQTT-383] Payload Format -> Payload Format Indicator in Table 2.6
11	[17th January 2017]	[Andrew Banks]	• [MQTT-354] Special terms are not consistently capitalized
11	[17 January 2017]	[Ken Borgendale]	• MQTT-357 and MQTT-358 Receive Maximum cleanup • MQTT-387 Topic Alias cleanup • MQTT-394 Fix figure and table references • MQTT-395 Fix following fields (variable header) for consistency • MQTT-401 Fix property length consistency • MQTT-404 Reason string consistency
11	[18th January 2017]	[Rahul Gupta]	• [MQTT-345] re connection and re transmission • [MQTT-351] Correction of usage “you” in the document
11	[18th January 2017]	[Ed Briggs]	• [MQTT-339] Changed case of multiple must not • [MQTT-343] Fixed Inconsistencies in Topic Alias • [MQTT-341] Shared subscription edits. • [MQTT-370] Consistency Variable Byte Integer • [MQTT-393] Incorrect UNSUBACK variable length header value
11	[19th January 2017]	[Ed Briggs]	• [MQTT-397] Return Code v. • [MQTT-403] AUTH command Flag 0 set to zero.
11	[20th January 2017]	[Andrew Banks]	• [MQTT-335] using this version of MQTT
11	[24th January 2017]	[Ken Borgendale]	• [MQTT-352] Normalize packet names • [MQTT-411] Return code for Payload format invalid.
11	[26th January 2017]	[Rahul Gupta]	• [MQTT-377] Inconsistent use of properties • Changed Request Problem Info to Request Problem Information • Changed Request Reply Info to Request Reply Information • Change Reply Info to Reply Information
11	[27th January 2017]	[Andrew Banks]	• [MQTT-340] Continuing to process incorrect protocol names or protocol

			versions
11	30th January 2017	[Andrew Banks]	• [MQTT-347] Comments and Questions about Will*
11	1st February 2017	[Andrew Banks]	• Incorporate peter Niblett review comments.
11	6th February 2017	[Andrew Banks]	• [MQTT-409] Publications which are undeliverable because the packet is too large.
11	8th February 2017	[Rahul Gupta]	• [MQTT-353] Long discussion on sessions in Sessions Expiry interval • Citations validated and changed
11	14th February 2017	[Andrew Banks]	• [MQTT-412] Consolidated error handling
11	14th February 2017	[Ed Briggs]	• [MQTT-402] PUBREC Received – Removed erroneous text • [MQTT-342] Maximum Packet Size – Removed limits and added non-normative text.
11	22nd February 2017	[Rahul Gupta]	• [MQTT-376] Consistency of describing properties • [MQTT-372] Maximum QoS • Normative Statements Indexed