MQTT for Sensor Networks (MQTT-SN) Version 2.0

Committee Specification Draft 01

01 August 2024

This stage:

…

Previous stage:

N/A

Latest stage:

…

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

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

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

Related work:
This specification is related to:

- *MQTT Version 5.0*. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. OASIS Standard. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
- *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta. OASIS Standard. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.
- *MQTT-SN Version 1.2* by Andy Stanford-Clark and Hong Linh Truong. Link: https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf.

Abstract:
This specification defines the MQTT for Sensor Networks protocol (MQTT-SN). It is closely related to the MQTT v3.1.1 and MQTT v5.0 standards. MQTT-SN is optimized for implementation on low-cost, battery-operated devices with limited processing and storage resources. It is designed so that it will work over a variety of networking technologies and bridge to an MQTT network.

Status:
This document was last revised or approved by the OASIS Message Queuing Telemetry Transport (MQTT) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee

(TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt#technical .

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

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

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

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

[MQTT-SN-v2.0]

MQTT for Sensor Networks Version 2.0. Edited by Andrew Banks, Davide Lenzarini, Ian Craggs, Rahul Gupta, Simon Johnson, Stefan Hagen, and Tara E. Walker. 01 August 2024. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/mqtt/mqtt-sn/v2.0/csd01/mqtt-sn-v2.0-csd01.docx. Latest stage: https://docs.oasis-open.org/mqtt/mqtt-sn/v2.0/mqtt-sn-v2.0.docx
(**Note:** Publication URIs are managed by OASIS TC Administration; please don't modify. The OASIS TC Process requires that Work Products at any level of approval must use the OASIS file naming scheme, and must include the OASIS copyright notice. The URIs above have been constructed according to the file naming scheme. Remove this note before submitting for publication.)

Notices

Table of Contents

1 Introduction

[All text is normative unless otherwise labeled]

1.0 Intellectual property rights policy

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

1.1 Changes from earlier Versions

[Optional section.]

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

1.1.1 MQTT-SN v1.2

Text describing the changes/differences

1.2 Organization of the MQTT-SN specification

The specification is split into five chapters:

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

1.3 Terminology

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

Datagram:

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

Underlying Network:

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

The arrival of a datagram is not guaranteed, but the contents of any datagram which does arrive at its destination must be correct.

Network Address:

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

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

Unicast Address:

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

Multicast Address:

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

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

Network Identity:

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

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

Virtual Connection:

An MQTT-SN construct corresponding to the network connection in MQTT. It associates a Network Identity with an MQTT-SN endpoint.

Application Message:

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

Client:

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

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

and/or:

- publishes Application Messages to a Multicast Address.

and/or:

- accepts Application Messages from a Multicast Address.

Server:

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

Also known as a **Gateway**.

A Server:

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

and/or

- accepts packets from a Multicast Address.

and/or

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

and/or

- opens an MQTT Network Connection to an MQTT Server for every MQTT-SN CONNECT request
- forwards equivalent MQTT packets to the MQTT Server for each MQTT-SN packet received

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

MQTT Server:

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

Also known as a **Broker**.

An MQTT Server:

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

Session:

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

Session State:

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

Subscription:

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

Wildcard Subscription:

A Wildcard Subscription is a Subscription with a Topic Filter containing one or more wildcard characters. This allows the subscription to match more than one Topic Name. Refer to section 4.7.1 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.

Topic Alias:

A Topic Alias is an integer value that is used to identify the Topic instead of using the Topic Name, to reduce packet size.

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-SN Control Packet:

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

Malformed Packet:

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

Protocol Error:

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

Will Message:

An Application Message which is published by the Server after the Virtual Connection is deleted in cases where the Virtual Connection is not deleted normally. Refer to section 3.1.4.9 for information about Will Messages.

Retained Message:

An Application Message which is stored by the Server for a topic on the receipt of a Publish Packet with the retained flag set. When a Client subscribes to a topic with a Retained Message set, the Server sends the Retained Message to the Client, depending on the setting of the Retain Handling Subscribe Flags. Refer to section 3.1.17.2 and section 4.26 for more information about Retained Messages.

Disallowed Unicode code point:

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

Wireless Sensor Network:

Spatially dispersed and dedicated sensors that monitor and record the physical conditions of the environment and forward the collected data to a central location.

Also known as **WSN**, for short.

1.4 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

[RFC6455]

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

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

[Unicode]

The Unicode Consortium. The Unicode Standard,

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

1.5 Informative References

1.6 MQTT For Sensor Networks (MQTT-SN)

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

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

Consider an example of a medicine tracking application. The application needs to know the location and temperature of the medicine, but it does not want to concern itself with the network details of the devices providing the data. It may be that the number and types of the devices changes over time.

There may also be other applications using the same sensor data for other purposes. The model is that the devices and applications produce and consume data addressed by the Topics rather than the other devices and applications.

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

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

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

1.6.1 MQTT-SN and MQTT Differences

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

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

1 In addition to Topic Alias and long Topic Names MQTT-SN allows predefined and short two-byte Topic Names.

2 If the network supports Multicast, Gateway discovery can be implemented, otherwise the Gateway addresses must be configured in the nodes.

3 Support for sleeping clients allows battery operated devices to enter a low power mode. In this state, Application Messages for the Client are buffered by the Gateway and delivered when the client wakes.

4 A new Quality of Service level (WITHOUT SESSION) is introduced in MQTT-SN, allowing devices to publish without a session having been established.

5 MQTT-SN has fewer requirements on the underlying transport and it can use connectionless network transports such as User Datagram Protocol (UDP).

6 MQTT-SN introduces the PROTECTION packet for packet-based security based on symmetric-key cryptography.

1.7 Data representation

1.7.1 Bits (Byte)

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

1.7.2 Two Byte Integer

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

1.7.3 Four Byte Integer

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

1.7.4 UTF-8 Encoded String

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

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

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

Table 1: Structure of UTF-8 Encoded Strings

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

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

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

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

- U+0001..U+001F control characters
- U+007F..U+009F control characters
- Code points defined in the Unicode specification [Unicode] to be non-characters (for example U+0FFFF)

A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always interpreted as U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver [MQTT-SN-1.7.4-3].

Informative example

For example, the string A𪛖 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:

Bit	7	6	5	4	3	2	1	0
byte 1	‘A’ (0x41)							
	0	1	0	0	0	0	0	1
byte 2	(0xF0)							

	1	1	1	1	0	0	0	0
byte 3	(0xAA)							
	1	0	1	0	1	0	1	0
byte 4	(0x9B)							
	1	0	0	1	1	0	1	1
byte 5	(0x94)							
	1	0	0	1	0	1	0	0

Table 2: Fixed Length UTF-8 Encoded String informative example

2 MQTT-SN Control Packet format

2.1 Structure of an MQTT-SN Control Packet

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

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

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

Table 3: Structure of an MQTT-SN Control Packet

2.1.1 Packet Header

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

Byte	Use
1	Length
2	MQTT-SN Control Packet Type

Table 4: Packet Header Format 1

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

Table 5: Packet Header Format 2

2.1.2 Length

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

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

The 3-byte format allows the encoding of packet lengths up to 65,535 bytes. It is more efficient to use the shorter 1-byte format for packets with lengths up to and including 255 bytes.

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

Informative comment

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

2.1.3 MQTT-SN Control Packet Type

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

Name	Value	Direction of flow	Description
Reserved	0x00	Forbidden	Reserved
ADVERTISE	0x01	Gateway Multicast	Advertise the gateway presence
SEARCHGW	0x02	Client Multicast	Client GWINFO request
GWINFO	0x03	Gateway to Client	Response to a SEARCHGW
AUTH	0x04	Client to Gateway or Gateway to Client	Authentication handshake
CONNECT	0x05	Client to Gateway	Virtual Connection request
CONNACK	0x06	Gateway to Client	Virtual Connection acknowledgement
REGISTER	0x0A	Client to Gateway	Request topic alias
REGACK	0x0B	Gateway to Client	Supply topic alias
PUBLISH	0x0C	Client to Gateway or Gateway to Client	Publish packet
PUBACK	0x0D	Client to Gateway or Gateway to Client	Publish acknowledgment (QoS 1) or Publish error (Any QoS).
PUBCOMP	0x0E	Client to Gateway or	Publish complete (QoS 2 delivery part 3)

		Gateway to Client	
PUBREC	0x0F	Client to Gateway or Gateway to Client	Publish received (QoS 2 delivery part 1)
PUBREL	0x10	Client to Gateway or Gateway to Client	Publish release (QoS 2 delivery part 2)
PUBLISH-WITHOUT-SESSION	0x11	Client to Gateway	Publish packet for out of a session messages which have no session on the gateway or broker
SUBSCRIBE	0x12	Client to Gateway	Subscribe request
SUBACK	0x13	Gateway to Client	Subscribe acknowledgment
UNSUBSCRIBE	0x14	Client to Gateway	Unsubscribe request
UNSUBACK	0x15	Gateway to Client	Unsubscribe acknowledgment
PINGREQ	0x16	Client to Gateway	PING request
PINGRESP	0x17	Gateway to Client	PING response
DISCONNECT	0x18	Client to Gateway or Gateway to Client	Disconnect notification
WAKEUP	0x19		Wake up request
- Reserved -	0x1A-0x1D		Forbidden (Old Will Range)
- Reserved -	0x1E-0xFD		Forbidden
FORWARDER ENCAPSULATION	0xFE	Forwarder to Client or Forwarder to Gateway	Encapsulated MQTT-SN packet

PROTECTION ENCAPSULATION	0xFF	Client to Gateway or Gateway to Client	A protection envelope that can encapsulate any MQTT-SN packet with the exception of Forwarder-Encapsulation packet (0xFE)

Table 6: Packet type listing

Name	Value	Direction of flow	Description
Reserved	0x00	Forbidden	Reserved
CONNECT	0x01		
CONNACK	0x02		
PUBLISH	0x03		
PUBACK	0x04		
PUBREC	0x05		
PUBREL	0x06		
PUBCOMP	0x07		
SUBSCRIBE	0x08		
SUBACK	0x09		
UNSUBSCRIBE	0x0A		
UNSUBACK	0x0B		
PINGREQ	0x0C		
PINGRESP	0x0D		
DISCONNECT	0x0E		
AUTH	0x0F		
Reserved			
REGISTER			
REGACK			
PUBLISHWOS			
ADVERTISE			

SEARCHGW			
GWINFO			
FORWARDER ENCAPSULATION			
PROTECTION ENCAPSULATION			

2.2 Packet Identifier

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

MQTT-SN Control Packet	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)
PUBWOS	NO
PUBACK	YES
PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES

SUBACK	YES
UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	YES
PINGRESP	YES
DISCONNECT	NO
AUTH	NO
ADVERTISE	NO
SEARCHGW	NO
GWINFO	NO
REGISTER	YES
REGACK	YES
FORWARDER ENCAPSULATION	NO
PROTECTION ENCAPSULATION	NO

Table 8 Packets with Packet Identifier

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

Each time a Client sends a new SUBSCRIBE, UNSUBSCRIBE, or PUBLISH (where QoS > 0) MQTT-SN Control Packet it MUST assign it a non-zero Packet Identifier that is currently unused.

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

The Packet Identifier becomes available for reuse after the sender has processed the corresponding acknowledgement packet, defined as follows. In the case of a QoS 1 PUBLISH, this is the corresponding PUBACK; in the case of QoS 2 PUBLISH it is PUBCOMP or a PUBREC with a Reason Code of 128 or greater. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK.

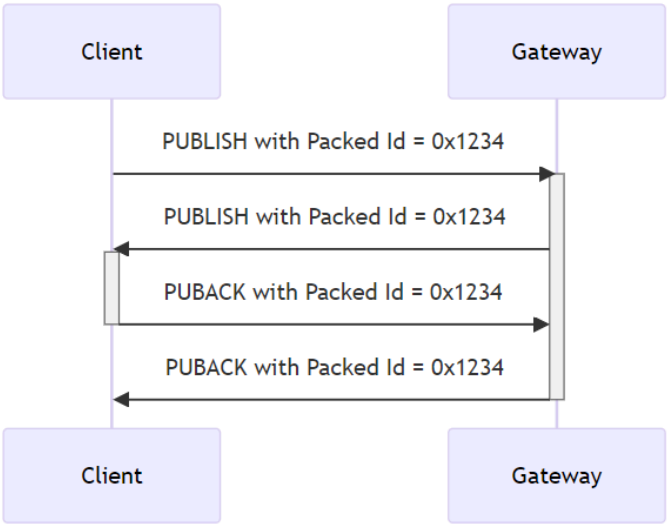
Packet Identifiers used with PUBLISH, SUBSCRIBE and UNSUBSCRIBE packets form a single, unified set of identifiers separately for the Client and the Gateway in a Session. A Packet Identifier cannot be used by more than one command at any time.

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

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

Informative comment

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



2.3 MQTT-SN Packet Fields

2.3.1 Protocol Id

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

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

2.3.2 Radius

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

2.3.3 Reason Code

A Reason Code is a one-byte unsigned value that indicates the result of an operation. Reason Codes share a common set of values across the various Control Packet types. Reason Code values of 0x80 or greater indicate failure.

Each value and meaning of each *Reason Code* field is shown below.

Identifier		Name	Packets	Description
Dec	Hex			
0	0x00	Success	CONNACK, SUBACK, UNSUBACK, REGACK, PUBACK, PUBREC, PUBREL, PUBCOMP, AUTH (server only)	The operation was successful.

0	0x00	Normal disconnection	DISCONNECT	Delete the Virtual Connection normally. Do not send the Will Message.
0	0x00	Granted QoS 0	SUBACK	The subscription is accepted and the maximum QoS sent will be QoS 0. This might be a lower QoS than was requested.
1	0x01	Granted QoS 1	SUBACK	The subscription is accepted and the maximum QoS sent will be QoS 1. This might be a lower QoS than was requested.
2	0x02	Granted QoS 2	SUBACK	The subscription is accepted and any received QoS will be sent to this subscription.
4	0x04	Disconnect with will message	DISCONNECT (client only)	The Client wishes to disconnect but requires that the Server also publishes its Will Message.
16	0x10	No matching subscribers	PUBACK, PUBREC	The Application Message is accepted but there are no subscribers. If the Server knows that there are no matching subscribers, it MAY use this Reason Code instead of 0x00 (Success).
17	0x11	No subscription existed	UNSUBACK	No matching Topic Filter is being used by the Client.
24	0x18	Continue authentication	AUTH	Continue the authentication with another step.
25	0x19	Re-authenticate	AUTH (client only)	Initiate a re-authentication.
128	0x80	Unspecified error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT	The receiver does not accept the request but either does not want to reveal the reason, or it does not match one of the other values.
129	0x81	Malformed packet	CONNACK, DISCONNECT	The received packet does not conform to this specification.

130	0x82	Protocol error	CONNACK, DISCONNECT	An unexpected or out of order packet was received.
131	0x83	Implementation specific error	CONNACK, PUBACK, PUBREC, REGACK, SUBACK, UNSUBACK, DISCONNECT	The packet received is valid but cannot be processed by this implementation.
132	0x84	Unsupported Protocol Version	CONNACK	The Server does not support the version of the MQTT or MQTT-SN protocol requested by the Client.
133	0x85	Client identifier not valid	CONNACK	The Client Identifier is a valid string but is not allowed by the Server.
134	0x86	Bad user name or password	CONNACK	The Server does not accept the User Name or Password specified by the Client
135	0x87	Not authorized	CONNACK, PUBACK, PUBREC, REGACK, SUBACK, UNSUBACK, DISCONNECT (server only)	The request is not authorized.
136	0x88	Server unavailable	CONNACK	The MQTT Server is not available.
137	0x89	Server busy	CONNACK, DISCONNECT (server only)	The Server is busy and cannot continue processing requests from this Client.
138	0x8A	Banned	CONNACK	This Client has been banned by administrative action. Contact the server administrator.
139	0x8B	Server shutting down	DISCONNECT (server only)	The Server is shutting down.
140	0x8C	Bad authentication method	CONNACK, DISCONNECT	The authentication method is not supported or does not match the authentication method currently in use.
141	0x8D	Keep alive timeout	DISCONNECT (server only)	The Connection is closed because no packet has

				been received for 1.5 times the Keepalive time.
142	0x8E	Session taken over	DISCONNECT (server only)	Another Connection using the same ClientID has connected causing this Connection to be closed.
143	0x8F	Topic filter invalid	SUBACK, UNSUBACK, DISCONNECT (server only)	The Topic Filter is correctly formed, but is not accepted by this Server.
144	0x90	Topic name invalid	CONNACK, PUBACK, PUBREC, DISCONNECT (server only)	The Topic Name is correctly formed, but is not accepted by this Client or Server.
145	0x91	Packet identifier in use	PUBACK, PUBREC, SUBACK, UNSUBACK	The specified Packet Identifier is already in use.
146	0x92	Packet identifier not found	PUBREL, PUBCOMP	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session State on the Client and Server.
147	0x93	Receive maximum exceeded	DISCONNECT	The Client or Server has received more than Receive Maximum publication for which it has not sent PUBACK or PUBCOMP.
148	0x94	Topic alias invalid	DISCONNECT (server only)	The Client or Server has received a PUBLISH packet containing a Topic Alias which is greater than the Maximum Topic Alias it sent in the CONNECT or CONNACK packet. (Transparent gateway only)
149	0x95	Packet too large	CONNACK, DISCONNECT	The packet size is greater than Maximum Packet Size for this Client or Server.
150	0x96	Packet rate too high	DISCONNECT	The received data rate is too high.
151	0x97	Quota exceeded	REGACK, SUBACK, DISCONNECT	An implementation or administrative imposed limit has been exceeded.

152	0x98	Administrative action	DISCONNECT	The Virtual Connection is deleted due to an administrative action.
153	0x99	Payload format invalid	PUBACK, PUBREC, DISCONNECT (server only)	The MQTT payload format does not match the one specified by the Payload Format Indicator. (Transparent gateway only)
154	0x9A	Retain not supported	CONNACK, DISCONNECT (server only)	The MQTT Server does not support retained messages. (Transparent gateway only)
155	0x9B	QoS not supported	CONNACK, DISCONNECT (server only)	The Client specified a QoS greater than the QoS specified in a Maximum QoS in the MQTT CONNACK. (Transparent gateway only)
156	0x9C	Use another server	CONNACK, DISCONNECT (server only)	The Client should temporarily change its Server.
157	0x9D	Server moved	CONNACK, DISCONNECT (server only)	The Server is moved and the Client should permanently change its server location.
158	0x9E	Shared subscription not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Shared Subscriptions. (Transparent gateway only)
159	0x9F	Connection rate exceeded	CONNACK, DISCONNECT (server only)	This Virtual Connection is deleted because the connection rate is too high.
160	0xAD	Maximum connect time	DISCONNECT (server only)	The maximum connection time authorized for this Virtual Connection has been exceeded.
161	0xA1	Subscription identifiers not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Subscription Identifiers; the subscription is not accepted. (Transparent gateway only)
162	0xA2	Wildcard subscription not supported	SUBACK, DISCONNECT (server only)	The MQTT Server does not support Wildcard Subscriptions; the

				subscription is not accepted. (Transparent gateway only)
230	0xE6	Only PROTECTION packet supported (Note 1)	Any packet except PROTECTION and Forwarder Encapsulation	Specific to MQTT-SN
231	0xE7	Protection scheme invalid	DISCONNECT	Specific to MQTT-SN
232	0xE8	Unknown Sender Id	DISCONNECT	Specific to MQTT-SN
240	0xF0	Unknown Topic Alias	PUBACK, SUBACK	Specific to MQTT-SN
241	0xF1	Congestion	SUBACK, REGACK, CONNACK, PUBACK	Specific to MQTT-SN
242	0xF2	Protection packet not supported	DISCONNECT	Specific to MQTT-SN
243	0xF3	Forwarder Encapsulation not supported	DISCONNECT	Specific to MQTT-SN
244	0xF4	Unknown topic alias	SUBACK, UNSUBACK, PUBACK, REGACK	Specific to MQTT-SN
245-255	0xF5-0xFF	Reserved for MQTT-SN		Specific to MQTT-SN

Table 9: Reason Code Values

Note(s):

1. It is used by a receiver to indicate that it expected a packet to be protected and it wasn't.

2.3.4 Topic Data

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

2.3.5 Topic Name

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

2.4 Topic Types

Several packets will refer to a topic type in their flags. This is a 2-bit field which determines the format of the topic value.

The allowable values are as follows:

	Topic Type Value	Name	Description
0	0b00	Session Topic Alias	A session topic alias is negotiated between the gateway and client within the scope of a gateway session.
1	0b01	Predefined Topic Alias	A predefined alias is known statically by both the gateway and the client outside the scope of a gateway session. No negotiation is required since both entities have knowledge of the topic alias mapping.
2	0b10	Short Topic Name	A 2-byte topic name which requires no negotiation.
3	0b11	Long Topic Name	A full topic, which requires no session negotiation.

Table 10: Topic types and their description

Refer to section 4 for detailed descriptions of topic names and aliases.

3 MQTT-SN Control Packets

3.1 Format of Individual Packets

This section specifies the format of the individual MQTT-SN packets.

3.1.1 ADVERTISE - Gateway Advertisement

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

Table 11: ADVERTISE Packet

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

Informative comment

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

3.1.1.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.1.2 GwId

The *GwId* field is at least 1-byte identifier and uniquely identifies a gateway which is advertising itself in the network.

The MQTT-SN protocol itself doesn't guarantee the uniqueness of the *GwId* field.

Informative comment

If the Gateway has a MAC address, it can be used as *GwId*.

3.1.1.3 Duration

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

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

3.1.2 SEARCHGW - Search for A Gateway

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

Table 12: SEARCHGW packet

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

Informative comment

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

3.1.2.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.2.2 Radius

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

A Client or a Gateway MUST NOT forward the SEARCHGW received if the Radius value is 0.

If a Client or a Gateway forwards the SEARCHGW received, it MUST reduce the Radius value by 1.

3.1.3 GWINFO - Gateway Information

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

Table 13: GWINFO packet

The GWINFO packet is sent as response to a SEARCHGW packet with the radius as indicated in the SEARCHGW packet. If sent by a Gateway, it contains only the id of the sending Gateway; otherwise, if sent by a client, it also includes the address of the Gateway.

Informative comment

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

3.1.3.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.3.2 GwId

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

3.1.3.3 GwAdd

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

3.1.4 CONNECT - Connection Request

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	CONNECT FLAGS							
	Reserved		Default Awake Messages			Auth	Will	Clean Start
Byte 3	*0*		*X*	*X*	*X*	*X*	*X*	*X*	*X*
	WILL FLAGS (OPTIONAL - ONLY WHEN WILL FLAG SET)							
	Reserved			Will Retain	Will QoS		Will Topic Type	
Byte (3 + 1)	*0*	*0*	*0*	*X*	*X*	*X*	*X*	*X*
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Protocol Version							
Byte 7	Keep Alive MSB							
Byte 8	Keep Alive LSB							
Byte 9	Session Expiry Interval MSB							
Byte 10	Session Expiry Interval							
Byte 11	Session Expiry Interval							
Byte 12	Session Expiry Interval LSB							

Byte 13	Max Packet Size MSB
Byte 14	Max Packet Size LSB
	WILL DATA (OPTIONAL - ONLY WHEN WILL FLAG SET)
Byte (14 + 1)	Will Topic Data MSB
Byte (14 + 2)	Will Topic Data LSB
Byte (14 + 3)	Will Payload Length MSB
Byte (14 + 4)	Will Payload Length LSB
Byte (14 + N)	Will Payload Or (Will Topic Name + Will Payload)
	AUTH DATA (OPTIONAL - ONLY WHEN AUTH FLAG SET)
Byte (14 + 1)	Auth Method Length
Byte (14 + 2)	Auth Method
Byte (14 + 3)	Auth Data Length MSB
Byte (14 + 4)	Auth Data Length LSB
Byte (14 + 5)	Auth Data
Byte 15 … N	Client Identifier

Table 14: CONNECT packet

The CONNECT packet is sent from the Client to the Gateway to create or continue a Session.

3.1.4.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.4.2 Connect Flags

The Connect Flags is 1 byte field which contains several parameters specifying the behavior of the MQTT-SN Virtual Connection.

The Connect *Flags* field includes the following flags:

- **Clean Start:** Stored in Bit 0 and specifies whether the Virtual Connection starts a new Session or is a continuation of an existing Session
- **Will**: Stored in Bit 1 and if set to 1 it indicates that the Will Flags and the Will Data sections are present
- **Auth**: Stored in Bit 2 and indicates if the packet contains authentication data
- **Default Awake Messages**: A value between 0-15 to indicate the maximum number of messages a client shall receive during an AWAKE session. Specifying 0 will mean it is up to the gateway to determine how many messages it will send, which may be unbounded.

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

3.1.4.3 Packet Identifier

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

3.1.4.4 Protocol Version

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

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

Table 15: Protocol version values

The value of the Protocol Version field for MQTT-SN version 2.0 MUST be 2 (0x02).

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

3.1.4.5 Keep Alive Timer

The Keep Alive is a Two Byte Integer greater than 0 (1 - 65,535), which is a time interval measured in seconds. It is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one MQTT-SN Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between MQTT-SN Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other MQTT-SN Control Packets, the Client MUST send a PINGREQ packet.

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

If the Gateway does not receive an MQTT-SN Control Packet from the Client within one and a half times the Keep Alive time period, it MUST consider the session 'LOST' (see state description in table 3.6).

If a Client does not receive a PINGRESP packet within a T_{retry} amount of time after it has sent a PINGREQ, it SHOULD retry the transmission according to section 4.24 up to the maximum number of attempts. If a PINGRESP is still not received it MUST close the Session to the Gateway by way of a DISCONNECT, with the understanding that the GW may no longer be reachable.

A Keep Alive must have a value greater than 0. It is considered a protocol error If a Keep Alive value of 0 is set.

> **Informative comment**
>
> The Gateway may have other reasons to disconnect the Client, for instance because it is shutting down. Setting Keep Alive does not guarantee that the Client will remain connected.
>
> **Informative comment**
>
> The actual value of the Keep Alive is application specific; typically, this is a few minutes. The maximum value of 65,535 is 18 hours 12 minutes and 15 seconds.

3.1.4.6 Session Expiry Interval

The Session Expiry Interval is a four-byte integer time interval measured in seconds. If the Session Expiry Interval is set to 0, the Session ends (and state deleted) when a (non SLEEPING) DISCONNECT packet is sent from either the client or gateway.

If the Session Expiry Interval is 0xFFFFFFFF (UINT_MAX), the Session does not expire.

The Client and Gateway MUST store the Session State after a DISCONNECT is issued if the Session Expiry Interval is greater than 0.

Informative comment

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

Informative comment

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

3.1.4.7 Maximum Packet Size

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

Informative comment

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

The packet size is the total number of bytes in an MQTT-SN Control Packet, as defined in section 2.1. The Client uses the Maximum Packet Size to inform the Server that it will not process packets exceeding this limit.

The Gateway MUST NOT send packets exceeding Maximum Packet Size to the Client. If a Client receives a packet whose size exceeds this limit, this is a Protocol Error, the Client uses DISCONNECT with Reason Code 0x95 (Packet too large).

Where a Packet is too large to send, the Gateway MUST discard it without sending it and then behave as if it had completed sending that Application Message.

Informative comment

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

3.1.4.8 Client Identifier (ClientID)

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

Informative comment

A Client Identifier can be between 0 - 65,521 bytes. We advise for practicality, ClientID's are restricted to a reasonable size (less than 243 bytes to fit within a small CONNECT packet).

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

The Gateway may choose to allow more than 23 bytes.

The Client Identifier MUST be a UTF-8 Encoded String.

3.1.4.9 Connect Will Flags (optional, only with *Will* flag set)

The Connect optional *Will Flags* is 1 byte field which contains several parameters specifying the handling of the Will Message. It is present only if the Will flag in the Connect *Flags* contains the value 1.

The Connect optional *Will Flags* field includes the following flags:

- **Will Topic Type:** This is a 2-bit field in Bit 0 and 1 which determines the format of the topic value. Refer to Table 10 for the definition of the various topic types.
- **Will QoS**: Stored in Bit 2 and 3, these two bits specify the QoS level to be used. The value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). A value of 3 (0x03) is a Malformed Packet.
- **Will Retain**: Stored in Bit 4, this bit specifies if the Will Message is to be retained when it is published.

If the Will Flag is set to 1 this indicates that a Will Message MUST be stored on the Server and associated with the Session [MQTT-SN-3.1.4.9-1]. The Will Message consists of the Will Topic, and Will Payload fields in the CONNECT Packet. The Will Message MUST be published after the Virtual Connection is deleted or the Session ends, unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) [MQTT-SN--3.1.4.9-2].

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 Server deletes the Virtual Connection because of a protocol error.

If the Will Flag is set to 1, the Will Topic, and Will Payload fields MUST be present in the Packet [MQTT-SN-3.1.4.9-3]. The Will Message MUST be removed from the stored Session State in the Server once it has been published or the Server has received a DISCONNECT packet with a Reason Code of 0x00 (Normal disconnection) from the Client [MQTT-SN-3.1.4.9-4].

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

3.1.4.10 Will Topic Short or Will Topic Length (optional, only with *Will* flag set)

In the case of Will Topic Type being b11 this field will refer to the length of data assigned to the “Will Full Topic Name”, in all other cases, this will be the value used as the Will topic alias or Will short topic name.

3.1.4.11 Will Payload Length (optional, only with *Will* flag set)

Contains the length of the Will Payload field.

3.1.4.12 Will Payload (optional, only with *Will* flag set)

It contains the content of the Will Message which is published after the Virtual Connection is deleted.

In the case of Topic Type b11 the payload section will be prefixed with a “Will Full Topic Name” encoded with a UTF-8 encoded string value of length determined by the previously defined length field. Thereafter, the *Will Payload* field corresponds to the MQTT Will Payload and so it defines the Application Message Payload that is to be published to the Will Topic and this field consists of Binary Data. It has a variable length defined by the Will Payload Length fields.

3.1.4.13 Authentication Method Length (optional, only with *Auth* flag set)

Single byte value (max 0-255 bytes), representing the length of field used to specify the authentication method. Refer to section 4.10 for more information about extended authentication.

3.1.4.14 Authentication Method (optional, only with *Auth* flag set)

A UTF-8 Encoded String containing the name of the authentication method. Refer to section 4.10 for more information about extended authentication.

3.1.4.15 Authentication Data Length (optional, only with *Auth* flag set)

Two byte value (max 0-65535 bytes), representing the length of field used to specify the authentication data. Refer to section 4.10 for more information about extended authentication.

3.1.4.16 Authentication Data (optional, only with *Auth* flag set)

Binary Data containing authentication data. The contents of this data are defined by the authentication method. Refer to section 4.10 for more information about extended authentication.

3.1.4.17 CONNECT Actions

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

1. The Server MUST validate that the CONNECT packet matches the format described in section 3.1.4 and MUST NOT create a Virtual Connection for this CONNECT if it does not match. [MQTT-SN-3.1.4.17-1] The Server MAY send a CONNACK with a Reason Code of 0x80 or greater as described in section 4.12.
2. The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST NOT create a Virtual Connection for this CONNECT [MQTT-SN-3.1.4.17-2]. It MAY send an appropriate CONNACK response with a Reason Code of 0x80 or greater as described in section 3.1.5 and section 4.12.

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 Reason Code of 0x8E (Session taken over) as described in section 4.12 and MUST delete the Virtual Connection of the existing Client [MQTT-SN-3.1.4.17-3]. 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 Clean Start that is described in section 4.16 [MQTT-SN-3.1.4.17-4].
3. The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Reason Code [MQTT-SN-3.1.4.17-5].
4. Start Application Message delivery and Keep Alive monitoring.

Informative comment

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

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 except AUTH packets [MQTT-3.1.4-6].

Informative comment

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

Informative comment

Clients that send MQTT-SN Control Packets before they receive CONNACK will be unaware of Server information including whether any existing Session is being used.

Informative comment

The Server can limit reading from the Network if the Client sends too much data before authentication is complete. This is suggested as a way of avoiding denial of service attacks.

3.1.5 CONNACK - Connect Acknowledgement

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	CONNACK FLAGS							
	Reserved						Auth	Session Present
Byte 3	0	0	0	0	0	0	X	X
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Reason Code							
Byte 7	Session Expiry Interval MSB							

Byte 8	Session Expiry Interval
Byte 9	Session Expiry Interval
Byte 10	Session Expiry Interval LSB
	AUTH DATA (OPTIONAL - ONLY WHEN AUTH FLAG SET)
Byte (10+1)	Auth Method Length
Byte (10+2)	Auth Method
Byte (10+3)	Auth Data Length MSB
Byte (10+4)	Auth Data Length LSB
Byte (10+5)	Auth Data
Byte 11…N	Assigned Client Identifier (optional)

Table 16: CONNACK packet

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

3.1.5.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.5.2 Connack Flags

The CONNACK FLAGS is a 1 byte field located at byte 4 which contains flags specifying the behavior of the MQTT-SN Virtual Connection on the gateway. Bits 7-2 of the CONNACK FLAGS are reserved and MUST be set to 0.

The Connack *Flags* field includes the following flags:

- **Session Present:** Stored in Bit 0 and specifies whether an existing session was present on the gateway for the given client identifier. A value of 1 indicates a session was present, a value 0 indicates no session was present.

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

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

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

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

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

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

- **Auth:** Stored in Bit 1 and specifies whether the packet contains Auth material that should be considered.

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

3.1.5.3 Packet Identifier

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

3.1.5.4 Reason Code

Byte 5 in the CONNACK header contains the Connect Reason Code. The values for the Connect Reason Code field are shown in Table 9: Reason Code Values. The Server sending the CONNACK packet MUST use one of the Connect Reason Code values.

If a Server sends a CONNACK packet containing a Reason code of 128 or greater it MUST then delete the Virtual Connection.

3.1.5.5 Session Expiry Interval

If the Session Expiry Interval is 0, the value of Session Expiry Interval in the CONNECT Packet is used. *The server uses this property to inform the Client that it is using a value other than that sent by the Client in the CONNECT*.

3.1.5.6 Authentication Method Length (optional, only with *Auth* flag set)

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

3.1.5.7 Authentication Method (optional, only with *Auth* flag set)

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

3.1.5.8 Authentication Data Length (optional, only with *Auth* flag set)

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

3.1.5.9 Authentication Data (optional, only with *Auth* flag set)

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 4.11 for more information about extended authentication.

3.1.5.10 Assigned Client Identifier

The Client Identifier assigned by the gateway when the associated CONNECT packet contained no Client Identifier. If the Client connects using a zero length Client Identifier, the Server MUST respond

with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Gateway.

The Assigned Client Identifier MUST be a UTF-8 Encoded String.

Informative comment

Assigned Client Identifiers SHOULD be less than 247 bytes so they can be accommodated in a small packet version. This is also to cater for devices which may not support larger Client Identifiers.

Informative comment

Where a transparent gateway obtains an Assigned Client Identifier which is deemed too large for a device, it should maintain a registry to map shorter gateway generated Client Identifiers with their versions returned from the broker.

3.1.6 AUTH - Authentication Exchange

Bit	7	6	5	4	3	2	1	0
byte 1	Length							
byte 2	Packet Type							
byte 3	Packet Identifier MSB							
byte 4	Packet Identifier LSB							
byte 5	Auth Reason Code							
byte 6	Auth Method Length (K)							
byte 7:7+K	Auth Method							
byte 8+K:N	Auth Data (N)							

Table 22: AUTH packet

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

3.1.6.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.7.2 Packet Identifier

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

3.1.6.2 Reason Code

Byte 3 in the Auth packet holds the Authentication Reason Code. The values for the Authentication Reason Code field are shown in Table 9: Reason Code Values. The sender of the AUTH Packet MUST use one of the Authenticate Reason Codes.

3.1.6.3 Auth Method Length

The length of the auth method string.

3.1.6.4 Auth Method

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

3.1.6.5 Auth Data

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

Informative comment

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

3.1.6.6 Auth Actions

Refer to section 4.11 for more information about extended authentication.

3.1.7 REGISTER - Register Topic Alias Request

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
Byte 3	Packet Identifier MSB							
Byte 4	Packet Identifier LSB							
Byte 5	Topic Alias MSB							
Byte 6	Topic Alias LSB							
Byte 7…N	Topic Name							

Table 24: REGISTER packet

The REGISTER packet is sent by a client to a GW for requesting a topic alias value for the included topic name. It is also sent by a GW to inform a client about the topic alias value it has assigned to the included topic name.

3.1.7.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.7.2 Packet Identifier

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

3.1.7.3 Topic Alias

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

3.1.7.4 Topic Name

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

3.1.7.5 Register Actions

As described in section 4.17.

3.1.8 REGACK - Register Topic Alias Response

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	REGACK FLAGS							
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Topic Type	
Byte 3	0	0	0	0	0	0	X	X
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Topic Alias MSB							
Byte 7	Topic Alias LSB							
Byte 8	Reason Code							

Table 25: REGACK packet

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

3.1.8.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.8.2 REGACK Flags

The REGACK Flags is 1 byte field in Byte position 3 of the REGACK packet.

The REGACK Flags field includes the following flag:

- **Topic Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic value. Refer to Table 10 for the definition of the various topic types.

3.1.8.3 Packet Identifier

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

3.1.8.4 Topic Alias

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

3.1.8.5 Reason Code

Byte 8 in the REGACK packet holds the Register Reason Code. The values for the Register Reason Code field are shown in Table 9: Reason Code Values. The sender of the REGACK Packet MUST use one of the Register Reason Codes.

3.1.9 Publish Variants

MQTT-SN is designed to be optimized for packet size. For this reason, the PUBLISH packet has been split into 3 variants; Variant 1 catering for PUBLISH WITHOUT SESSION where no GW session is required, Variant 2 catering for Quality of Service 0 where no response ACK is required and thus no packet identifier is required and Quality of Service 1 and 2 where a response is expected. The table below breaks down the different versions of the PUBLISH packet and their respective type identifiers.

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

3.1.10 PUBWOS - Publish Without Session

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type (0x11)							
	PUBWOS FLAGS							
	Reserved			*Retain*	*Reserved*			*Topic Type*

Byte 3	*0*	*0*	*0*	*X*	*0*	*0*	*X*	*X*
Byte 4	Topic Data MSB							
Byte 5	Topic Data LSB							
Byte (6 + TL) .. N	Data Or (Full Topic Name + Data)							

Table 28: PUBWOS packet

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

The PUBWOS packet does not have a corresponding feature in MQTT. If forwarded to an MQTT connection, PUBWOS packets MUST have their MQTT Quality of Service level set to 0. [MQTT-SN-3.1.10-1]

Informative comment

If the Transport Layer supports multicast, like UDP/IP, the PUBWOS packet is generally sent to a Multicast Address.

Informative comment

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

3.1.10.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.10.2 PUBWOS Flags

The PUBWOS Flags field is 1-byte located in the Byte 3 position of the PUBWOS control packet.

The PUBWOS Flags includes the following flags:

- **Topic Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic data field. Refer to Table 10 for the definition of the various topic types. **NOTE: only predefined topic alias, short topic or full topic types are allowed in PUBWOS packets.**
- **Retain**: 1 bit field stored in Bit 4 and has the same meaning as with MQTT. The field signifies whether the existing Retained Message for this topic is replaced or kept. For a detailed description of Retained Messages see section 4.26.

3.1.10.3 Topic Data

Contains 2 bytes of topic length (if the topic type is Full Topic Name) or the topic alias (predefined), or short topic name as indicated in the *Topic Type* field in flags. Determines the topic which this payload will be published to.

3.1.10.4 Data

In the case of Topic Alias Type b11 the data section will be prefixed with a “Full Topic Name” encoded with a UTF-8 encoded string value of length determined by the previously defined length

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

3.1.12.7 PUBWOS Actions

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

If received by a Client or Server, the PUBWOS packet MUST be treated as if its QoS were 0 [MQTT-SN-3.1.12.7-1] as described in section 3.1.12.7.

3.1.11 PUBLISH - QoS 0

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type (0x0C)							
	PUBLISH QoS 0 FLAGS							
	Reserved	*QoS*		*Retain*	*Reserved*	*Reserved*	*Topic Type*	
Byte 3	*0*	*0*	*0*	*X*	*0*	*0*	*X*	*X*
Byte 4	Topic Data MSB							
Byte 5	Topic Data LSB							
Byte (6 + TL) .. N	Data Or (Full Topic Name + Data)							

Table 29: PUBLISH packet

This packet is used by both clients and gateways to publish data for a certain topic. PUBLISH QoS 0, 1 & 2 packets received by a Gateway MUST be associated with a valid Client Session [MQTT-SN-3.1.11-1].

3.1.11.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.11.2 PUBLISH Flags

The PUBLISH Flags field is 1-byte located in Byte 3 position of the PUBLISH control packet.

The PUBLISH Flags includes the following flags:

- **Topic Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic data field. Refer to Table 10 for the definition of the various topic types.

- **QoS**: This is a 2-bit field stored in Bit 5 and 6. QoS has the same meaning as with MQTT indicating the Quality of Service. This field is set to “0b00” for QoS 0. For a detailed description of the various Quality Of Service levels refer to section 4.3.
- **Retain**: 1 bit field stored in Bit 4 and has the same meaning as with MQTT. The field signifies whether the existing Retained Message for this topic is replaced or kept. For a detailed description of Retained Messages see section 4.26.

3.1.11.3 Topic Data

Contains 2 bytes of topic length (if the topic type is Full Topic Name) or the topic alias (predefined or normal), or short topic name as indicated in the *Topic Type* field in flags. Determines the topic which this payload will be published to.

3.1.11.4 Data

In the case of Topic Type b11 the data section will be prefixed with a “Full Topic Name” encoded with a UTF-8 encoded string value of length determined by the previously defined length field. Thereafter, the *Data* field corresponds to the payload of an MQTT PUBLISH packet. It has a variable length and contains the application data that is being published.

3.1.11.5 PUBLISH - QoS 0 Actions

As described in section 3.1.12.7.

3.1.12 PUBLISH - QoS 1 and 2

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Length							
Byte 2	Packet Type (0x0C)							
	PUBLISH QoS 1&2 FLAGS							
	DUP	*QoS*		*Retain*	*Reserved*	*Reserved*	*Topic Type*	
Byte 3	*X*	*X*	*X*	*X*	*0*	*0*	*X*	*X*
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Topic Data MSB							
Byte 7	Topic Data LSB							
Byte (8 + TL) .. N	Data Or (Full Topic Name + Data)							

Table 30: PUBLISH packet

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

PUBLISH QoS 0, 1 & 2 packets received by a Gateway MUST be associated with a valid Client Session [MQTT-SN-3.1.12-1].

3.1.12.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.12.2 PUBLISH Flags

The PUBLISH Flags field is 1-byte located in Byte 3 position of the PUBLISH control packet.

The PUBLISH Flags includes the following flags:

- **Topic Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic data field. Refer to Table 10 for the definition of the various topic types.
- **QoS**: This is a 2-bit field stored in Bit 5 and 6. QoS has the same meaning as with MQTT indicating the Quality of Service. The QoS levels are shown below:

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

Table 3-2 - QoS definitions

For a detailed description of the various Quality Of Service levels refer to section 4.3.

- **DUP**: 1 bit field stored in Bit 7 and has the same meaning as with MQTT. It notes the duplicate delivery of packets. If the DUP flag is set to “0”, it signifies that the packet is sent for the first time. If the DUP flag is set to “1”, it signifies that the packet was retransmitted or retried.
- **Retain**: 1 bit field stored in Bit 4 and has the same meaning as with MQTT. The field signifies whether an existing Retained Message for this topic is replaced or kept. For a detailed description of Retained Messages see section 4.26.

3.1.12.4 Packet Identifier

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

3.1.12.5 Topic Data

Contains 2 bytes of topic length (if the topic type is Full Topic Name) or the topic alias (predefined or normal), or short topic name as indicated in the *Topic Type* field in flags. Determines the topic which this payload will be published to.

3.1.12.6 Data

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

3.1.12.7 PUBLISH Actions

The receiver of a PUBLISH Packet MUST respond with the packet as determined by the QoS in the PUBLISH Packet. [MQTT-SN-3.1.12.7-1].

Table 3-3 Expected PUBLISH packet response

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

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

The Server uses a PUBLISH packet to send an Application Message to each Client which has a matching subscription. 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 Application Message might match multiple filters. In this case the Server MUST deliver the Application Message to the Client respecting the maximum QoS of all the matching subscriptions [MQTT-SN-3.1.12.7-2]. In addition, the Server MAY deliver further copies of the Application Message, one for each additional matching subscription and respecting the subscription's QoS in each case.

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

Informative Comment

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

The Client MUST NOT send more than one QoS 1 or QoS 2 PUBLISH packet for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Server [MQTT-3.3.4-7]. If it receives more than one QoS 1 or QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Server uses a DISCONNECT packet with Reason Code 0x93 (Receive Maximum exceeded) as described in section 4.12 Handling errors. Refer to section 4.9 for more information about flow control.

Informative 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.

The Server MUST NOT send more than one QoS 1 and QoS 2 PUBLISH packet for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Client [MQTT-3.3.4-9]. If it receives more than one QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Client uses DISCONNECT with Reason Code 0x93 (Receive Maximum exceeded) as described in section 4.12 Handling errors. Refer to section 4.9 for more information about flow control.

Informative 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.

3.1.13 PUBACK – Publish Acknowledgement

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

Table 31: PUBACK packet

A PUBACK packet is the response to a PUBLISH packet with QoS 1. It can also be sent as response to a PUBLISH packet of any QoS (*with the exception of QoS -1, or PUBWOS*) in case of an error; the error reason is then indicated in the *Reason Code* field.

3.1.13.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.13.2 Packet Identifier

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

3.1.13.3 Reason Code

Byte 5 in the PUBACK packet holds the Reason code in response to the PUBLISH packet. The PUBACK Reason Codes are shown in Table 9: Reason Code Values. The Client or Server sending the PUBACK packet MUST use one of the PUBACK Reason Codes.

3.1.13.4 PUBACK Actions

As described in section 4.3.3.

3.1.14 PUBREC (QoS 2 delivery part 1)

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

Table 33: PUBREC packet

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

3.1.14.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.14.2 Packet Identifier

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

3.1.14.3 Reason Code

Byte 5 in the PUBREC packet holds the Reason code in response to the PUBLISH packet. The PUBREC Reason Codes are shown in Table 9: Reason Code Values. The Client or Server sending the PUBREC packet MUST use one of the PUBREC Reason Codes.

3.1.14.4 PUBREC Actions

As described in section 4.3.4.

3.1.15 PUBREL (QoS 2 delivery part 2)

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

Table 34: PUBREL packet

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

3.1.15.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.15.2 Packet Identifier

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

3.1.15.3 Reason Code

Byte 5 in the PUBREL packet holds the Reason code in response to the PUBREC packet. The PUBREL Reason Codes are shown in Table 9: Reason Code Values. The Client or Server sending the PUBREL packet MUST use one of the PUBREL Reason Codes.

3.1.15.4 PUBREL Actions

As described in section 4.3.4.

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

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

Table 35: PUBCOMP packet

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

3.1.16.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.16.2 Packet Identifier

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

3.1.16.3 Reason Code

Byte 5 in the PUBCOMP packet holds the Reason code in response to the PUBREL packet. The PUBCOMP Reason Codes are shown in Table 9: Reason Code Values. The Client or Server sending the PUBCOMP packet MUST use one of the PUBCOMP Reason Codes.

3.1.16.4 PUBCOMP Actions

As described in section 4.3.4.

3.1.17 SUBSCRIBE - Subscribe Request

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

	SUBSCRIBE FLAGS							
	No Local	*QoS*		*Retain as published*	*Retain Handling*		*Topic Type*	
Byte 3	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Topic Data MSB			**OR**			Topic Filter Byte 6 … N	
Byte 7	Topic Data LSB							

Table 36: SUBSCRIBE packet

The SUBSCRIBE packet is used by a client to subscribe to a certain topic name.

3.1.17.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.17.2 SUBSCRIBE Flags

The SUBSCRIBE Flags field is 1-byte and contains the following flags:

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

  It is a Protocol Error to send a Retain Handling value of 3.
- **Topic Type**: This is a 2-bit field in Bit 0 and 1 which determines the format of the topic data field. Refer to Table 10 for the definition of the various topic types.

3.1.17.3 Packet Identifier

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

3.1.17.4 Topic Data or Topic Filter

Contains Fixed Length UTF-8 Encoded String topic filter, topic alias (predefined or normal), or short topic name as indicated in the *Topic Type* field in flags. Determines the topic names which this subscription is interested in.

3.1.17.5 SUBSCRIBE Actions

When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a SUBACK packet [MQTT-SN-3.1.17.5-1]. The SUBACK packet MUST have the same Packet Identifier as the SUBSCRIBE packet that it is acknowledging [MQTT-SN-3.1.17.5-2].

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 Subscription's Topic Filter for the current Session, then it MUST replace that existing Subscription with a new Subscription [MQTT-SN-3.1.17.5-3]. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its Subscription Options could be different. If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Application Messages MUST NOT be lost due to replacing the Subscription [MQTT-SN-3.1.17.5-4].

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

The SUBACK packet sent by the Server to the Client MUST contain a Reason Code [MQTT-SN-3.1.17.5-5]. This Reason Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed [MQTT-SN-3.1.17.5-6]. The Server might grant a lower Maximum QoS than the subscriber requested. The QoS of Application Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published Application message and the Maximum QoS granted by the Server [MQTT-SN-3.1.17.5-7]. The server is permitted to send duplicate copies of a Application message to a subscriber in the case where the original Application message was published with QoS 1 and the maximum QoS granted was QoS 0.

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

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

> **Informative comment**

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

3.1.18 SUBACK - Subscribe Acknowledgement

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	SUBACK FLAGS							
	Reserved	*Reserved*		*Reserved*	*Reserved*	*Reserved*	*Topic Type*	
Byte 3	*0*	*0*	*0*	*0*	*0*	*0*	*X*	*X*
Byte 4	Topic Data MSB							
Byte 5	Topic Data LSB							
Byte 6	Packet Identifier MSB							
Byte 7	Packet Identifier LSB							
Byte 8	Reason Code							

Table 37: SUBACK packet

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

3.1.18.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.18.2 Flags

The SUBACK Flags field is 1-byte located in Byte 3 position of the SUBACK control packet. The SUBACK Flags includes the following flags:

- **Topic Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic data field. Refer to Table 10 for the definition of the various topic types.

3.1.18.3 Topic Data

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

3.1.18.4 Packet Identifier

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

3.1.18.5 Reason Code

Byte 8 in the SUBACK packet holds the Reason code in response to the SUBSCRIBE packet. The SUBACK Reason Codes are shown in Table 9: Reason Code Values.The Server sending the SUBACK packet MUST use one of the SUBACK Reason Codes.

3.1.19 UNSUBSCRIBE - Unsubscribe Request

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	UNSUBSCRIBE FLAGS							
	Reserved	*Reserved*		*Reserved*	*Reserved*	*Reserved*	*Topic Type*	
Byte 3	*0*	*0*	*0*	*0*	*0*	*0*	*X*	*X*
Byte 4	Packet Identifier MSB							
Byte 5	Packet Identifier LSB							
Byte 6	Topic Data MSB				**OR**	Topic Filter Byte 6 … N		
Byte 7	Topic Data LSB							

Table 39: UNSUBSCRIBE packet

An UNSUBSCRIBE packet is sent by the client to the GW to unsubscribe from named topics.

3.1.19.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.19.2 UNSUBSCRIBE Flags

For The UNSUBSCRIBE Flags is 1 byte field in Byte position 3 of the UNSUBSCRIBE packet.

The UNSUBSCRIBE Flags field includes the following flag:

- **Topic Type.** This is a 2-bit field in Bit 0 and 1 which determines the format of the topic Id value. Refer to Table 10 for the definition of the various topic types.

3.1.19.3 Packet Identifier

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

3.1.19.4 Topic Data or Topic Filter

Contains Fixed Length UTF-8 Encoded String topic filter, topic alias (predefined or normal), or short topic name as indicated in the *Topic Type* field in flags. Determines the topic names which this subscription is interested in.

3.1.19.4 UNSUBSCRIBE Actions

The Topic Filter (whether it contains wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription MUST be deleted [MQTT-SN-3.1.19.4-1], otherwise no additional processing occurs.

When a Server receives UNSUBSCRIBE :

- It MUST stop adding any new Application Messages which match the Topic Filters, for delivery to the Client [MQTT-SN-3.1.19.4-2].
- It MUST complete the delivery of any QoS 1 or QoS 2 Application Messages which match the Topic Filters and it has started to send to the Client [MQTT-SN-3.1.19.4-3].
- It MAY continue to deliver any existing Application Messages buffered for delivery to the Client.

The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet [MQTT-3.1.19.4-4]. The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-3.1.19.4-5].

3.1.20 UNSUBACK - Unsubscribe Acknowledgement

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

Table 40: UNSUBACK packet

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

3.1.20.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.20.2 Packet Identifier

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

3.1.20.3 Reason Code

Byte 5 in the UNSUBACK packet holds the Reason code in response to UNSUBSCRIBE packet. The UNSUBACK Reason Codes are shown in Table 9: Reason Code Values. The server sending the UNSUBACK packet MUST use one of the UNSUBACK Reason Codes.

3.1.21 PINGREQ - Ping Request

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
Byte 3	Packet Identifier MSB							
Byte 4	Packet Identifier LSB							
Byte 5…N	Client Identifier (optional)							

The PINGREQ packet is an ”are you alive” packet that is sent from or received by a connected client.

3.1.21.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.21.2 Packet Identifier

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

3.1.21.3 Client Identifier (optional)

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

The Client Identifier MUST be a Fixed Length UTF-8 Encoded String [MQTT-SN-3.1.21.3-1].

3.1.21.4 PINGREQ Actions

The Server MUST send a PINGRESP packet in response to a PINGREQ packet [MQTT-SN-3.1.21.4-1].

The Client MAY send a PINGRESP packet in response to a PINGREQ packet [MQTT-SN-3.1.21.4-2].

3.1.22 PINGRESP - Ping Response

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

Byte 3	Packet Identifier MSB
Byte 4	Packet Identifier LSB
Byte 5	Application Messages Remaining (optional)

Table 43: PINGRESP packet

A PINGRESP packet is the response to a PINGREQ packet and means ”yes I am alive”. PINGREQ packets flow in either direction, sent either by a connected client or the gateway. it has only a header and no variable part.

A PINGRESP packet is also sent by a Gateway to inform a sleeping CLient that it has no more buffered packets for that Client.

3.1.22.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.22.2 Packet Identifier

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

3.1.22.3 Application Messages Remaining

The number of Application Messages still queued for delivery at the Server when a Client is sent back to sleep. Optional – for use at the end of a client's awake period. Values can be:

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

Table 44: Allowed PINGRESP continuation values

3.1.23 DISCONNECT - Disconnect Notification

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

	Reserved				*Reason Code Present*	*Session Expiry Interval Present*	*Reason String Present*	*Retain Registrations*
Byte 3	0	0	0	0	X	X	X	X
Byte 4	Reason Code (optional)							
Byte 5	Session Expiry Interval MSB (optional)							
Byte 6	Session Expiry Interval (optional)							
Byte 7	Session Expiry Interval (optional)							
Byte 8	Session Expiry Interval LSB (optional)							
Bytes 9..N	Reason String (optional)							

Table 45: DISCONNECT packet

As with MQTT, the DISCONNECT packet is sent by a client to indicate that it wants to delete the Virtual connection. The gateway will acknowledge the receipt of that packet by returning a DISCONNECT to the client. A server or gateway may also send a DISCONNECT to a client, e.g. in case a gateway, due to an error, cannot map a received packet to a client. Upon receiving such a DISCONNECT packet, a client should try to create another Virtual Connection by sending a CONNECT packet to the gateway or server.

A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Reason Code of less than 0x80 [MQTT-SN-3.1.23-1].

A DISCONNECT packet with a *Session Expiry Interval* field is sent by a client when it wants to go to the “asleep” state. The receipt of this packet is also acknowledged by the gateway by means of a DISCONNECT packet.

3.1.23.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.23.2 Disconnect Flags

The Disconnect Flags is 1 byte field located at byte 3 which contains parameters specifying the behavior of the MQTT-SN sleep on the gateway.

The Disconnect *Flags* field includes the following flags:

- **Reason Code Present:** Stored in Bit 3 Does the reason code exist on the packet
- **Session Expiry Interval Present:** Stored in Bit 2, Does the session expiry interval field exist.
- **Reason String Present:** Stored in Bit 1, Does the reason string field exist
- **Retain Registrations:** Stored in Bit 0 and specifies whether registrations should be retained by the gateway during the sleep state. “0” indicates registrations should be removed during the sleeping period and renegotiated when AWAKE or ACTIVE. “1” indicates registrations should be retained during the SLEEP period, and therefore not renegotiated when AWAKE or ACTIVE.

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

3.1.23.3 Reason Code

The Reason Code for the DISCONNECT packet is optional. If provided, Byte 3 in the DISCONNECT control packet holds the Reason Code of the disconnection. If not provided, 0x00 (Normal disconnection) is assumed.

The DISCONNECT Reason Codes are shown in Table 9: Reason Code Values. The Client or Server sending the DISCONNECT packet MUST use one of the DISCONNECT Reason Code values [MQTT-SN-3.1.23.3-1].

3.1.23.4 Session Expiry Interval

The Session Expiry Interval is a four-byte integer time interval measured in seconds. If the Session Expiry Interval is set to 0 or omitted, the Session is transitioned to the "***disconnected***" state. When the value of this field is greater than zero, it is deemed to be sent by a client that wants to transition to the "***asleep***" state, see Section 3.19 for further details. At this point the keep alive timer becomes obsolete until the device issues a new CONNECT.

The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server [MQTT–SN-3.1.23.4-1].

3.1.23.5 Reason String

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

3.1.23.6 DISCONNECT Actions

After sending a DISCONNECT packet the sender:

- MUST NOT send any more MQTT-SN Control Packets on that Virtual Connection [MQTT-SN-3.1.23.6-1].

After sending a DISCONNECT packet the Server:

- MUST delete the Virtual Connection [MQTT-SN-3.1.23.6-2].

After sending a DISCONNECT packet the Client:

- SHOULD wait for a DISCONNECT packet in response from the Server.

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

- MUST discard any Will Message associated with the current Connection without publishing it [MQTT-SN-3.1.23.6-3], as described in section 3.1.4.9.
- MUST send a DISCONNECT packet in response [MQTT-SN-3.1.23.6-4], and SHOULD delete the Virtual Connection.

On receipt of DISCONNECT, the Client:

- SHOULD delete the Virtual Connection.

3.1.24 WAKEUP - Wake up request

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

Table ??: WAKEUP packet

The wakeup packet is a signal sent from the gateway to a client. It is an indication from the gateway that the client should wake up. The client is not obliged to honor this request, nor may it even receive the packet. It can choose to ignore the request, or undertake one of the sequences outlined in the 4.25 Sleeping clients section. The client need not respond to this packet.

3.1.24.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.24.2 WAKEUP Actions

The Client MAY choose to follow the AWAKE procedure in response to receiving a WAKEUP packet [MQTT-SN-3.1.21.4-2].

3.1.25 Forwarder Encapsulation

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

Table 53: Format of an encapsulated MQTT-SN frame

As detailed in Section 4, MQTT-SN clients can also access a gateway via a forwarder in case the gateway is not directly attached to their WSNs. The forwarder simply encapsulates the MQTT-SN Packets it receives on the wireless side and forwards them unchanged to the gateway; in the opposite direction, it decapsulates the Packets it receives from the gateway and sends them to the clients, unchanged too.

3.1.25.1 Length

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

3.1.25.2 Packet Type

Coded “0xFE”, see Table 6

3.1.25.3 Ctrl

The Ctrl byte contains control information exchanged between the GW and the forwarder.

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

Table 54: Format of the ctrl byte

3.1.25.4 Radius

Transmission radius (only relevant in direction gateway to forwarder)

3.1.25.5 Wireless Node Id

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

3.1.25.6 MQTT SN Packet

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

3.1.26 Protection Encapsulation

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type							
	PROTECTION FLAGS							
	Auth Tag Length				*Crypto Material Length*		*Monotonic Counter Length*	
Byte 3	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*
Byte 4	Protection Scheme							
Byte 5 - 12	Sender Id							
Byte 13 - 1(	Random							

Byte 17 - P	Crypto Material (Optional)
Byte Q - R	Monotonic Counter (Optional)
Byte S - T	Protected MQTT-SN Packet
Byte U - N	Authentication Tag

Table 53: Format of the protection encapsulated MQTT-SN packet

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

The sender is the originator of the “Protected MQTT-SN Packet” and responsible for its protection. This responsibility MUST NOT be delegated to a third entity like a Forwarder.

The sender identification is required as the sender and the receiver of the protected packet must have access to the same shared key to be used directly or after derivation. The authentication of the sender and the receiver, their authorizations and the provisioning of the shared keys used to protect integrity and optionally confidentiality of the protected packet content are out of scope.

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

//TODO - Break out the conformance aspects of this paragraph from recommendations.

When the PROTECTION packet is handled by a gateway, it is mandatory to use it to protect all MQTT-SN packets exchanged with a Client for which a shared key (indexed by its Sender Id) is available.

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

When the PROTECTION packet is handled by a Client, it is mandatory to use it to protect all MQTT-SN packets exchanged with a GW for which a shared key (indexed by its GwId) is available.

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

3.1.26.1 Length

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

3.1.26.2 Packet Type

Coded “0x1E”, see Table 63

3.1.26.3 Protection Flags

The PROTECTION Flags is 1 byte field in Byte position 3 of the packet, specifying the properties of the PROTECTION.

The PROTECTION Flags field includes the following flags:

- **(Auth)entication tag length -** (4 bits) should represent the number of 16 bits groups forming the authentication tag in big-endian order.
    - Only 14 of the 16 possible values are allowed:
        - If 0x00, the authentication tag length is provider defined
        - the values from 0x1 to 0x2 are Reserved;
        - any other value 0xZ, so between 0x3 and 0xF, is allowed and the authentication tag length will be (0xZ+1)*16 bits; for example
            - if the value is 0xF, the Authentication tag length will be (0xF+1)*16=256 bits;
            - if the value is 0x3, the Authentication tag length will be (0x3+1)*16=64 bits;
    - If a truncation of the output of the authentication algorithm is required, it has to be taken in most significant bits first order (leftmost bits).
    - If an extension of the output of the authentication algorithm is required, 0s are appended until the Authentication tag length is reached.
    - Some values are not allowed for some protection schemes. For instance the values 0x03, 0x04, 0x05, 0x06 are not allowed for AES-CCM-128-128, AES-CCM-128-192, AES-CCM-128-256, AES-GCM-128-128, AES-GCM-128-192, AES-GCM-128-256 and ChaCha20/Poly1305 as for those protection schemes the 128-bit authentication tag can't be truncated.

- **Crypto material length -** (2 bits) should represent the number of 16 bits groups forming the crypto material in big-endian order. Below the meaning of each possible value:
    - if 0x3, a crypto material field of 96 bits (12 bytes) is present
    - if 0x2, a crypto material field of 32 bits (4 bytes) is present
    - if 0x1, a crypto material field of 16 bits (2 bytes) is present
    - if 0x0, the crypto material field is not present.

- **Monotonic counter length -** (2 bits) should represent the number of bytes forming the monotonic counter in big-endian order. Only 3 of the 4 possible values are allowed:
    - the value 0x3 is Reserved;
    - if 0x2, a monotonic counter field of 32 bits (4 bytes) is present;
    - if 0x1, a monotonic counter field of 16 bits (2 bytes) is present;
    - if 0x0, the monotonic counter field is not present.

3.1.26.4 Protection Scheme

A (1 byte) field located at byte 4 should contain one of the not Reserved indexes in the following table. In general two types of protection scheme are considered: **Authentication only** (like HMAC or CMAC) and **AEAD** (Authenticated Encryption with Associated Data, like GCM, CCM or ChaCha20/Poly1305).

Index	Name	Auth Only	Key size	Tag size
0x00	HMAC-SHA256 (Note 1)	Yes	Any size (Note 2)	256 bits

0x01	HMAC-SHA3_256 (Note 1)	Yes	Any size (Note 2)	256 bits
0x02	CMAC-128 (Note 3)	Yes	128 bits	128 bits
0x03	CMAC-192 (Note 3)	Yes	192 bits	128 bits
0x04	CMAC-256 (Note 3)	Yes	256 bits	128 bits
0x05-0x3B	RESERVED			
0x3C-0x3F	Provider defined	Yes	Provider defined	Provider defined
0x40	AES-CCM-64-128 (Notes 4,5)	No	128 bits	64 bits
0x41	AES-CCM-64-192 (Notes 4,5)	No	192 bits	64 bits
0x42	AES-CCM-64-256 (Notes 4,5)	No	256 bits	64 bits
0x43	AES-CCM-128-128 (Notes 4,5)	No	128 bits	128 bits
0x44	AES-CCM-128-192 (Notes 4,5)	No	192 bits	128 bits
0x45	AES-CCM-128-256 (Notes 4,5)	No	256 bits	128 bits
0x46	AES-GCM-128-128 (Notes 6,7)	No	128 bits	128 bits
0x47	AES-GCM-128-192 (Notes 6,7)	No	192 bits	128 bits
0x48	AES-GCM-128-256 (Notes 6,7)	No	256 bits	128 bits
0x49	ChaCha20/Poly1305 (Notes 8,9)	No	256 bits	128 bits
0x4A-0xEF	RESERVED			
0xF0-0xFF	Provider defined	No	Provider defined	Provider defined

Note(s):

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

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

3.1.26.5 Sender Identifier

Located at Bytes 5 - 12 the Sender Id field (8 bytes) should contain:

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

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

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

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

Informative comment

8 bytes for the "Sender Id" field seems enough as it is calculated with a cryptographic hash, so the probability of collision is 1/2^64=5.42x10^{-20}.

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

Gateway Behavior
In order to create a whitelist of authorized senders, the MQTT-SN Gateway should store a map of ClientID and SHA256(ClientID) truncated to the leftmost 64 bits (8 bytes for each registered ClientID) for the clients having an active session and store a list of authorized Sender Ids for the clients not capable to establish sessions.

3.1.26.6 Random

Located at Byte 13 - 16 , the "**Random**" field (4 bytes) should contain a random number (not guessable) generated at the PROTECTION packet creation .

Informative comment

In case of CCM, in the worst case scenario where the "Crypto Material" and the "Monotonic Counter" optional fields are not present, the recommended nonce on 13 bytes will be calculated as SHA256 truncated to 104 bits of the sequence Byte 1 to Byte 16 (all packet fields until Protected MQTT-SN Packet). So considering the same Sender Id, the same nonce can be generated with a probability of 1/2^32=2.33x10^{-10}. With a shorter Random field of 2 bytes, the same nonce would be calculated with a probability of only 1/2^16=1.53x10^{-5}. As CCM is a derivation of CTR (see https://en.wikipedia.org/wiki/CCM_mode), the nonce should never be reused for the same key so the probability to generate two identical nonces should be kept as low as possible. Same for GCM and ChaCha20/Poly1305, the security

depends on choosing a unique IV of 12 bytes for every encryption performed with the same key (https://en.wikipedia.org/wiki/Galois/Counter_Mode).

3.1.26.7 Crypto Material

Located at Byte (17 - P), the optional field "**Crypto Material**" contains 0, 2, 4 or 12 bytes of crypto material that when defined it can be used to derive, from a shared master secret, the same keys on the two endpoints and/or, when filled partially or totally with a random value, to further reduce the probability of IV/nonce reuse for CCM or GCM or ChaCha20/Poly1305. For instance when the Crypto material length is set to 0x03, the Crypto Material field can be partially filled with a random value of 9 bytes (the remaining 3 bytes can be set to 0 if not used) in order to reach the 13 bytes used only once recommended for the nonce used by CCM or it can be partially filled with a random value of 8 bytes in order to reach the 12 bytes used only once recommended for the IV/nonce used by GCM or ChaCha20/Poly1305 .

3.1.26.8 Monotonic Counter

Located at Byte Byte (Q - R), the optional field "**Monotonic Counter**" contains 0, 2 or 4 byte number that when defined, is increased by the Client or GW for every packet sent. The counters should be considered independent of session or destination. E.g. The UE will keep a counter independently from the GW.

3.1.26.9 Protected MQTT-SN Packet

Located at Byte (S - T), the field "**Protected MQTT-SN Packet**" contains the MQTT-SN packet that is being secured, encoded as per its packet type.
The "Protected MQTT-SN Packet" should not be a "Forwarder-Encapsulation packet" as the shared key used directly or after derivation for the protection must belong to the originator of the content and not to a Forwarder that, in general, is not able to securely identify the originator.

3.1.26.10 Authentication Tag

Located at Byte (U - N), the field "**Authentication tag**" field has a length depending on the "Authentication tag length" flag and it is calculated, on the basis of the "Protection scheme" selected in Byte 4, on ALL the preceding fields.

4 Operational behavior

An important design point of MQTT-SN is to be as close as possible to MQTT. Therefore, all protocol semantics should remain, as far as possible, the same as those defined by MQTT.

4.1 Session state

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

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

The Session State in the Client consists of:

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

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 PUBLISH Packets which have been sent to the Client, but have not been completely acknowledged.
- QoS 1 and QoS 2 PUBLISH Packets pending transmission to the Client and OPTIONALLY QoS 0 PUBLISH Packets pending transmission to the Client.
- QoS 2 PUBLISH Packets which have been received from the Client, but have not been completely acknowledged.
- The Will Message.
- If the Session is currently not connected, the time at which the Session will end and Session State will be discarded.

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

4.1.1 Storing Session State

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

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

The Server MUST discard the Session State when the Virtual Connection is deleted and the Session Expiry Interval has passed [MQTT-SN-4.1.1-3].

Informative comment

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

4.1.2 Session Establishment

As with MQTT, an MQTT-SN client needs to set up a session on the server, unless it is publishing ONLY using PUBLISH WITHOUT SESSION packets. The procedure for setting up a session with a server is illustrated in Fig. 3a and 3b.

The CONNECT packet contains flags to communicate to the gateway that Auth interactions, or WILL interactions should take place.

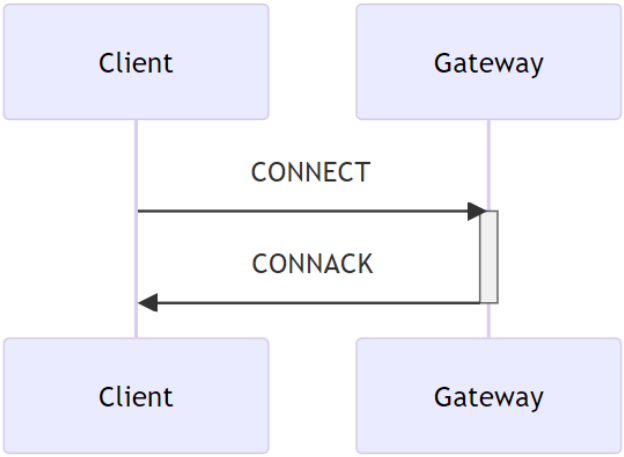


Figure 3a: Connect procedure (without Auth flag not Will flag set or no further authentication data required)

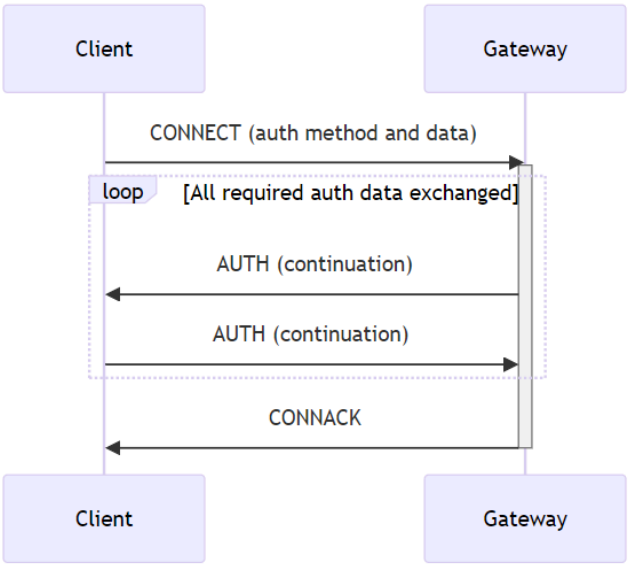


Figure 3b: Connect procedure (with Auth flag set and additional authentication data required)

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

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

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

4.2 Networks and Virtual Connections

The MQTT-SN protocol requires an Underlying Network to create a Virtual Connection. This carries Packets from a Client to a Gateway and a Gateway to a Client. The Underlying Network may also multicast Packets from a Client to more than one Gateway, and from a Gateway to more than one Client.

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

- The underlying transport does not need to be reliable, it is expected that Packets will be lost or delivered out of order.
- If the network might deliver a Packet more than once, then it is highly recommended that the PROTECTION ENCAPSULATION Monotonic Counter is used to eliminate the duplicates.
- The MQTT-SN protocol will tolerate out of order Packets and it will retransmit lost Packets.
- There is no packet error correction in MQTT-SN. If a corrupted or partial packet is received it will cause a protocol error.
- The MQTT-SN implementation may use either the origin network address or the sender identifier in the PROTECTION ENCAPSULATION to determine the identity of the Virtual Connection.
- The Underlying Network may be connectionless. Virtual Connections do not need to have an Underlying Network event that signals their creation or deletion.
- The Underlying Network may be a radio network.

Informative comment

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

Examples of possible consequences of not removing duplicate Packets are:
– DISCONNECT Packet applied to the wrong Virtual Connection
– SUBSCRIBE and UNSUBSCRIBE Packets applied to the wrong Virtual Connection
– PUBLISH QOS=2 published more than once

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

- DTLS v1.2 [RFC6347]

- DTLS v1.3 [RFC9147]
- QUIC [RFC9000]
- Non-IP protocols

- TCP/IP [RFC0793]
- TLS [RFC5246]
- WebSocket [RFC6455].

Informative comment

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

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

- created with a CONNECT packet
- deleted by any of:
  - a retry timeout
  - DISCONNECT packet
  - protocol error
- required for any MQTT-SN packet to be sent between MQTT-SN clients and servers, except any of the following Packets, including if they are Protection or Forwarder Encapsulated:
  - CONNECT, which creates a Virtual Connection
  - PUBWOS
  - ADVERTISE, SEARCHGW, GWINFO

4.3 Quality of Service levels and protocol flows

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

4.3.1 Publish without session: Any number of deliveries

No session is required to send a message. The message is delivered according to the capabilities of the underlying network. No response is sent by the receiver and no retry is performed by the sender. The message arrives at the receiver any number of times, including not at all.

The sender:

- MUST send a PUBWOS packet.

The receiver:

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

Informative:

PUBWOS packets may be Multicast or Unicast.

4.3.2 QoS 0: At most once delivery

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

In the QoS 0 delivery protocol, the sender

- MUST send a PUBLISH packet with QoS 0 and DUP flag set to 0.

In the QoS 0 delivery protocol, the receiver

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

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

4.3.3 QoS 1: At least once delivery

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

In the QoS 1 delivery protocol, the sender

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

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

In a difference to MQTT 5, the sender is NOT permitted to send further PUBLISH packets with different Packet Identifiers while it is waiting to receive acknowledgements. At any given time a sender must ONLY have 1 outstanding application message in flight.

In the QoS 1 delivery protocol, the receiver

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

Figure 4.2 – QoS 1 protocol flow diagram, Informative 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 (Note 1)
	<----------	Send PUBACK <Packet Identifier>
Discard message		

Note(s):

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

4.3.4 QoS 2: Exactly once delivery

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

In the QoS 2 delivery protocol, the sender:

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

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

In the QoS 2 delivery protocol, the receiver:

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

- MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL
- After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message

4.4 Packet delivery retry

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

1. when a Virtual Connection is deleted before the acknowledgement is received by the requester (and clean start is false)
2. when no acknowledgment is received by the by the requester within a configured timeout period during the existence of a Virtual Connection

These two situations are described in the following two sections.

4.4.1 Virtual Connection End

When a Client reconnects with Clean Start set to 0 and a session is present, both the Client and Server MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their original Packet Identifiers [MQTT-SN-4.4-1].

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

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver or retransmit a PUBLISH packet. The DUP flag must be set to 0 for all QoS 0 packets. [MQTT-SN-4.4-3].

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><Reason Code>
	<----------	

Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	---------->	
		Discard <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<----------	
Discard stored state		

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

4.4.1 Unacknowledged Packets

The Client or Gateway will start a retransmission retry timer, T_{retry}, when one of the following Packets is sent.

A Client MUST retransmit AUTH, REGISTER, PUBLISH Qos1, PUBLISH Qos2, PUBREL, SUBSCRIBE, UNSUBSCRIBE Packets, including a PROTECTION encapsulation if there is one, after T_{retry} has passed or the Virtual Connection deleted.

A Gateway MUST retransmit PUBLISH Qos1, PUBLISH Qos2, PUBREL Packets, including a PROTECTION encapsulation if there is one, after T_{retry} has passed or the Virtual Connection deleted.

The timer is canceled if the corresponding acknowledgement packet is received. The Client or Gateway MUST retransmit the Packet after T_{retry} has passed or delete the Virtual Connection.

If a Packet can be retransmitted it MUST be sent using a Unicast address.

If a Packet is retransmitted it MUST be identical to the previously transmitted Packet, the PROTECTION encapsulation need not be identical.

PUBLISH (used for QoS 0) and PUBLISH WITHOUT SESSION Packets MUST NOT be retransmitted.

If the Virtual Connection is deleted, the protocol will restart when a new CONNECT packet flows from the Client.

Informative comment

The value of the retry interval T_{retry} is not specified by the protocol, however, to be useful it ought to be longer than the network round trip time. If it is excessively long, the time taken to

detect and retransmit lost Packets will also be excessively long. Implementers need to take care not to use a retry interval that might cause the network to become congested with retried Packets.

The PINGREQ Packet described in [3.1.21 PINGREQ] can also be used to determine whether the Virtual Connection is alive.

An example of a retry algorithm is described in [Appendix E.F4]

4.5 Application Message receipt

When a Server takes ownership of an incoming Application Message it MUST add it to the Session State for those Clients that have matching Subscriptions [MQTT-SN-4.5-1]. Matching rules are defined in section 4.7.

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

4.6 Application Message ordering

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

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

Informative comment

When a stream of messages is published and subscribed to an Ordered Topic with QoS 1, the final copy of each message received by the subscribers will be in the order that they were published.

As no more than one message is “in-flight” at any one time, no QoS 1 message will be received after any later one even on re-connection. .For example a subscriber might receive them in the order 1,2,3,3,4 but not 1,2,3,2,3,4.

4.7 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-SN-4.7.1-1].

4.7.1.1 Topic level separator

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

4.7.1.2 Multi-level wildcard

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

Informative comment

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

- "sport/tennis/player1"
- "sport/tennis/player1/ranking
- "sport/tennis/player1/score/wimbledon"

Informative comment

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

4.7.1.3 Single-level wildcard

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

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

Informative comment

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

- "+" is valid
- "+/tennis/#" is valid

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

4.7.2 Topics beginning with $

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

Informative comment

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

Informative comment

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

4.7.3 Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

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

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

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

Informative comment

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

Informative comment

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

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

4.8 Subscriptions

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

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

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

4.9 Flow Control

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

- PUBLISH (QoS 1 and 2) and PUBREL
- REGISTER
- SUBSCRIBE
- UNSUBSCRIBE

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

Refer to section 3.1.12.7 for a description of how Clients and Servers react if they are sent more than one unacknowledged packet.

4.10 Server redirection

A Server can request that the Client uses another Server by sending a CONNACK or DISCONNECT packet with Reason Codes 0x9C (Use another server), or 0x9D (Server moved) as described in section 4.13.

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

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

4.11 Enhanced authentication

The CONNECT packet supports basic authentication of a Virtual 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. It might involve the exchange of AUTH packets between the Client and the Server after the CONNECT and before the CONNACK packets.

To begin an enhanced authentication, the Client sets the AUTH flag in the CONNECT packet and includes an Authentication Method and optionally Data, depending on the Authentication Method, used in the CONNECT packet. This specifies the authentication method to use and its parameters. If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in section 2.3.3 and MUST delete the Virtual Connection [MQTT-SN-4.10-1].

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

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

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

If the Server requires additional information to complete the authentication, it can send an AUTH packet to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-SN-4.10-2]. If the authentication method requires the Server to send authentication data to the Client, it is sent in the Authentication Data.

The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-SN-4.10-3]. If the

authentication method requires the Client to send authentication data for the Server, it is sent in the Authentication Data.

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

The Client can terminate the Virtual Connection at any point in this process by sending a DISCONNECT packet. The Server can reject the authentication at any point in this process. It MUST send a CONNACK with a Reason Code of 0x80 or above as described in section 4.13, [MQTT-SN-4.10-4].

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

The implementation of enhanced authentication is OPTIONAL for both Clients and Servers. If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet [MQTT-SN-4.10-6]. If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server [MQTT-SN-4.10-7].

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

Informative example showing a SCRAM challenge

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

Informative example showing a Kerberos challenge

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

4.11.1 Re-authentication

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

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

If the re-authentication fails, the Client or Server MUST send DISCONNECT with an appropriate Reason Code as described in section 4.13, and MUST delete the Virtual Connection [MQTT-SN-4.10.1-2].

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

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

4.12 Handling errors

4.12.1 Malformed Packet and Protocol Errors

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

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

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

The Reason 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
- 0xA2 Wildcard Subscriptions not supported

When a Client detects a Malformed Packet or Protocol Error associated with a Virtual Connection it SHOULD send a DISCONNECT packet containing an appropriate Reason Code and MUST delete the associated Virtual Connection. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in section 2.3.3.

When a Server detects a Malformed Packet or Protocol Error for any packet except ADVERTISE, SEARCHGW, GWINFO, PUBWOS and CONNECT, the Server SHOULD send a DISCONNECT packet with an appropriate Reason Code and MUST delete the associated Virtual Connection if one exists. [MQTT-4.13.1-1] In the case of an error in a CONNECT packet it MAY send a CONNACK packet containing the Reason Code. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in section 2.3.3. There are no consequences for other Sessions.

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

4.12.2 Other errors

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

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

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

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

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

4.13 Example MQTT-SN Architectures

There are three kinds of MQTT-SN components, MQTT-SN *clients*, MQTT-SN *gateways*, and MQTT-SN *forwarders*. MQTT-SN clients connect themselves to an MQTT server/broker via an MQTT-SN Gateway using the MQTT-SN protocol. An MQTT-SN Gateway may or may not be integrated with a MQTT server. Where an MQTT broker is involved, the MQTT protocol is used between the MQTT broker and the MQTT-SN Gateway. Its main function is the translation between MQTT and MQTT-SN.

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

Informative comment

The architectures described below are meant as examples and are not exhaustive.

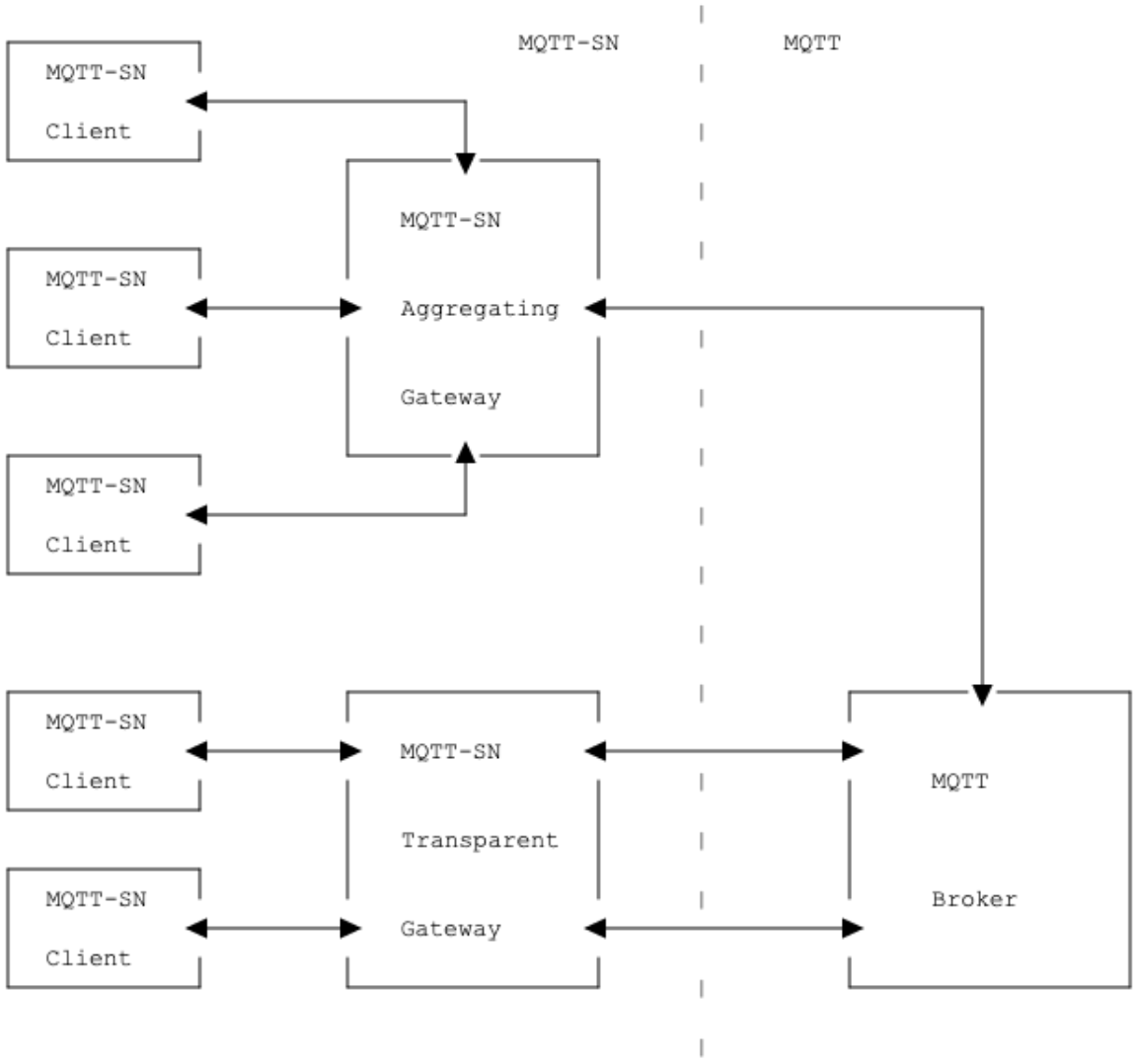


Figure 1: MQTT-SN Architecture

4.13.1 Transparent Gateway

For each connected MQTT-SN client a transparent Gateway will set up and maintain a MQTT connection to the MQTT server. This MQTT connection is reserved exclusively for the end-to-end and almost transparent packet exchange between the client and the server. There will be as many MQTT connections between the Gateway and the MQTT Broker as MQTT-SN clients connected to the Gateway. The transparent Gateway will perform a “syntax” translation between the two

protocols. Since all packet exchanges are end-to-end between the MQTT-SN client and the MQTT Server, all functions and features that are implemented by the server can be offered to the client.

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

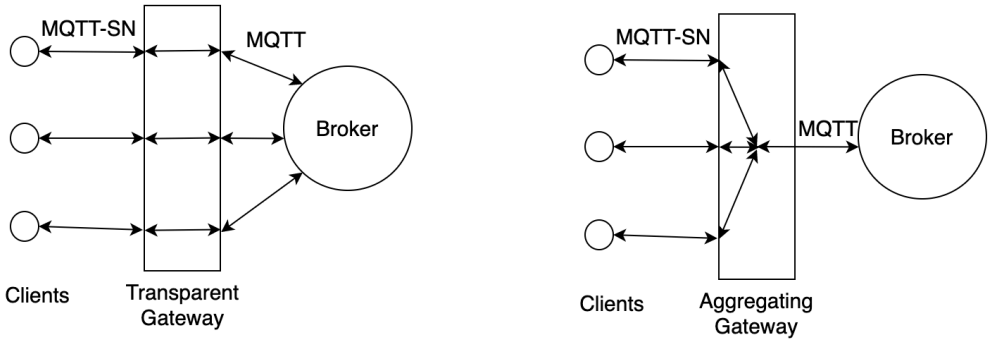


Figure 2: Transparent and Aggregating Gateways

4.13.2 Aggregating Gateway

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

4.14 Gateway Advertisement and Discovery

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

Multiple Gateways may be active at the same time in the same network. In this case they will have different identifiers. It is up to the Client to decide to which Gateway it wants to connect. At any point in time a Client is allowed to be connected to only one Gateway on the same network.

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

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

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

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

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

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

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

To give priority to the gateways a client will delay its sending of the GWINFO packet for a random time T_{GWINFO}. If during this delay time the client receives a GWINFO packet it will cancel the transmission of its GWINFO packet.

In case of no response the SEARCHGW packet may be retransmitted. In this case the time intervals between consecutive SEARCHGW packets should be increased by the exponential backoff algorithm described in the appendix.

4.15 Client states

At any given point in time, a client may be in one of **5 different states**. Transition through these states is governed by a strictly coordinated sequence of packets between client and server/gateway and further mediated by timers resident on the gateway. A client is in the *active* state when the server/gateway receives a CONNECT packet from that client. This state is supervised by the server/gateway with the “keep alive” timer. If the server/gateway does not receive any packet from the client for a period longer than the keep alive duration (indicated in the CONNECT packet), the gateway will consider that client as *lost* and activate for example the Will feature for that client. A client goes to the *disconnected* state when the server/gateway receives a DISCONNECT without a *session expiry interval* field. This state is not time-supervised by the server/gateway. A client moves into the asleep state by issuing a DISCONNECT with a *session expiry interval* field. For more information on the sleep state, please refer to the “Sleeping clients” section.

State	State Description	Possible Transitions
DISCONNECTED	The client is considered offline. The gateway may or may not have a previous session state for this client. From here a client may transition ONLY to the **ACTIVE** state.	**ACTIVE**

ACTIVE	The client is actively engaged in the session. It should be able to send and receive packets. Its state is supervised by the gateway with the associated “keep alive” timers. From here the client may transition to **ASLEEP** (by way of DISCONNECT with a session expiry interval > 0), **DISCONNECTED** (by way of DISCONNECT with a session expiry of 0) or **LOST** (by way of supervised gateway timers).	**ASLEEP** **DISCONNECTED** **LOST**
ASLEEP	The client is engaged in an ongoing session. It cannot receive packets; it can send packets. The gateway should not expect a response from the client in this state until further packets are received from the client. From here the client may transition to **AWAKE** (by way of PINGREQ), **ACTIVE** by way of CONNECT, **DISCONNECTED** (by way of DISCONNECT with a session expiry of 0) or **LOST** (by way of supervised gateway timers).	**AWAKE** **ACTIVE** **DISCONNECTED** **LOST**
AWAKE	The client is partially engaged in an ongoing session; it is obliged to not send ANY packets other than those involved in the receipt of PUBLISH packets (PUBACK, PUBREC, PUBCOMP, REGACK) or a DISCONNECT to transition to **DISCONNECTED**. The client transitions back to the **ASLEEP** state on receipt of a PINGRESP packet or **LOST** (by way of supervised gateway timers).	**ASLEEP** **DISCONNECTED** **LOST**
LOST	The client is considered offline and not able to receive packets until it has re-established a session with the GW by way of a CONNECT. The gateway **must not** attempt to send packets to a client in the **LOST** state**.** Any packets received from a client whose state is **LOST** should not be processed and a DISCONNECT with error should be sent in response, unless the packets received are PUBLISH WITHOUT SESSION or PUBLISH -1. Session state may exist on the GW for a client in the **LOST** state.	**ACTIVE**

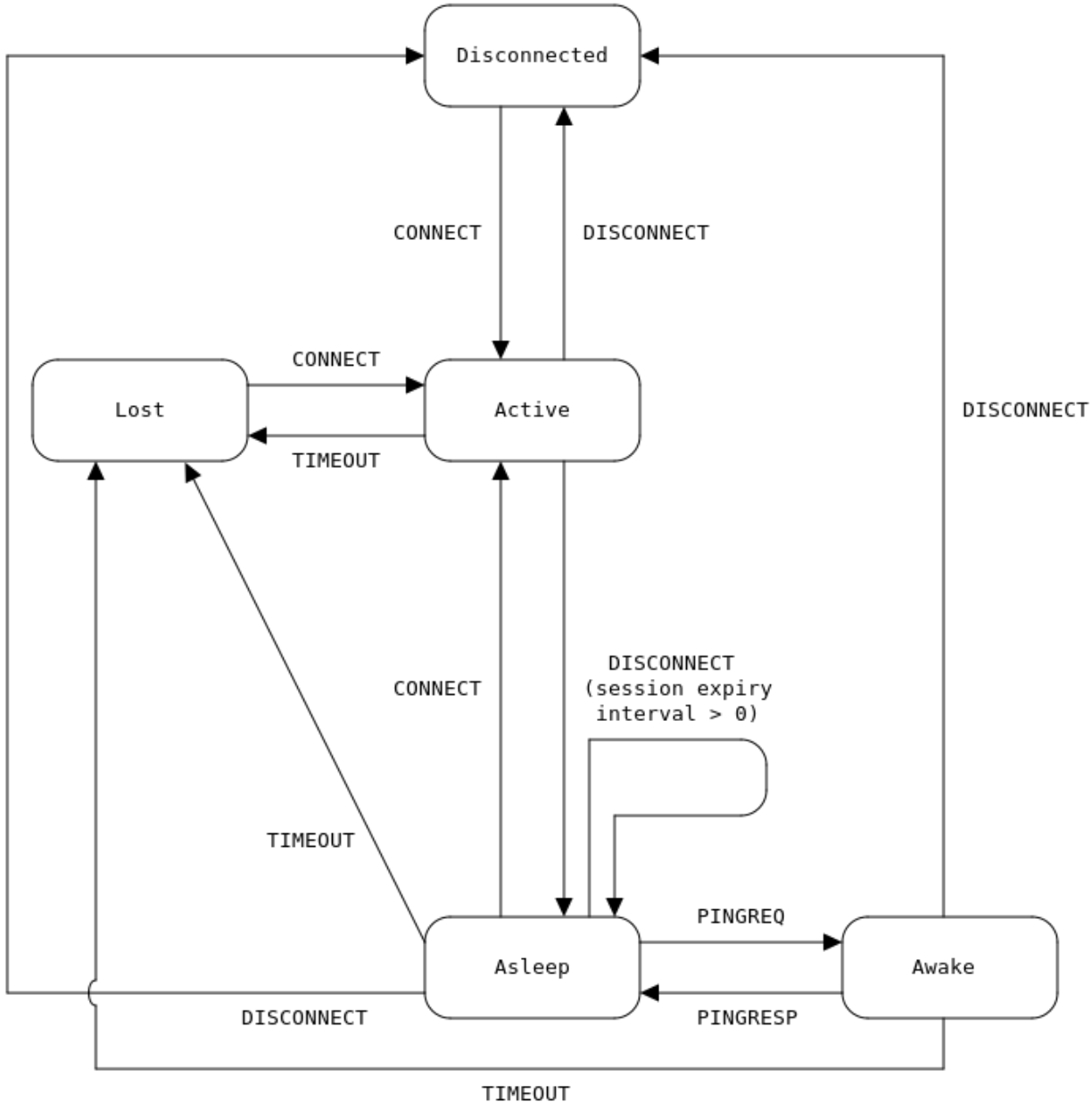


Figure 4: The Server View of the Client State

4.15.1 Gateway timers

The following timers must be managed by a Gateway per Client to handle the different Client states:

- “Keep Alive” timer based on the value defined in the CONNECT packet. If expired, a Client is moved from Active to Lost state or from Asleep to Lost state or from Awake to Lost state.
- “Session Expiry” timer based on the value defined in the CONNECT or the DISCONNECT packet. If expired, the session state associated with the Client can be removed.

4.16 Clean start

When a client disconnects, its subscriptions are retained for no less than the session expiration interval. They are persistent and valid for new (non clean start) sessions, until either they are explicitly un-subscribed by the client, or the client establishes a new session with the “clean start” flag set or their idle time exceeds the session expiry interval associated with the session.

The two flags “CleanStart” and “Will” in the CONNECT Packet have the following meanings:

- CleanStart=true, Will=true: The Gateway will delete all Session data related to the client, including Will data if present, and it will set the Will data in the Session state with the content of the CONNECT Will fields and will return CONNACK.
- CleanStart=true, Will=false: The gateway will delete all subscriptions and Will data, if present, related to the client and it will return CONNACK.
- CleanStart=false, Will=true: The gateway will keep all the client’s data and it will overwrite, if present, or add the Will data related to the client with the content of the CONNECT Will optional fields and it will return CONNACK.
- CleanStart=false, Will=false: The Gateway will keep all the client’s subscriptions and it will delete any Will data, if present, and it will return CONNACK.

Note that if a client wants to delete only its Will data at Virtual Connection creation, it could send a CONNECT packet with “CleanStart=false” and “Will=false”.

4.17 Topic Registration

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

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

To register a topic name a client sends a REGISTER packet to the Server. If the registration could be accepted, the gateway assigns a *topic alias* to the received topic name and returns it with a REGACK packet to the client.

If the client initiates a REGISTER against a topic which is known by the Server to have a predefined topic alias associated with it, it is an error case, but one which should not be terminal to the session since gateway updates could lead to this scenario. The gateway will specify its topic alias type to be predefined and set the topic alias value to match that defined on the gateway in the REGACK, it will also set a reason code on the REGACK to indicate the issue. The client can then choose to update its registry of predefined topic aliases if it so wishes.

If a Client sends a PUBLISH to a predefined topic alias, which is not defined on the Server, this is considered a protocol violation. [MQTT-SN-???]

If there are no predefined topic aliases, the gateway will pass back a SESSION topic alias type. If the registration can not be accepted, a REGACK is also returned to the client with the failure reason encoded in the *ReasonCode* field.

After having received the REGACK packet with *ReasonCode* =*"accepted"*, the client shall use the assigned *topicId* to publish data of the corresponding topic name. If, however, the REGACK contains a rejection code, the client may try to register later again. If the Reason Code was *"Congestion"*, the client should wait for a time T_{WAIT} before restarting the registration procedure.

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

A gateway sends a REGISTER packet to a client if it wants to inform that client about the topic name and the assigned topic alias that it will use later when sending PUBLISH packets of the corresponding topic name. This happens for example when no prior registrations exists, or when the client has DISCONNECTED with retain registration false, or the client re-connects without having set the "CleanStart" flag or the client has subscribed to topic names that contain wildcard characters such as # or +.

Informative comment

The gateway should attempt to make the best effort to reuse the same topic alias mappings that existed during any initial associated ACTIVE states.

4.18 Topic Mapping and Aliasing

On the gateway the mapping table between registered topic ids and topic names MUST be implemented per client (and not by a single shared pool between all clients), to reduce the risk of an incorrect topic id from a client matching another client's valid topic.

For performance and efficiency reasons the broker may choose to align topic aliases for registered normal topic aliases between multiple clients. The mapping table of predefined topic aliases is separate from normal registered aliases. It is global and shared between all clients and gateways and may overlap with registered aliases, since it is in a different pool.

4.19 Predefined topic aliases and short topic names

A "predefined" topic alias is a topic alias whose mapping to a topic name is known in advance by both the client' application and the gateway. This is indicated in the *Flags* field of the packet. When using pre-defined topic aliases, both sides can start immediately with the sending of PUBLISH packets; there is no need for the REGISTER procedure as in the case of "normal" topic aliases. When receiving a PUBLISH packet with a predefined topic alias, of which the mapping to a topic name is unknown, the receiver should return a PUBACK with the *ReasonCode*= *"*Unknown Topic Alias*"*.

The presence of a pre-defined topic alias does not imply any other meaning onto the topic name / topic filter itself. All lifecycle operations, for example SUBSCRIBE / UNSUBSCRIBE may still be used in the use of these aliases except for REGISTER.

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

4.20 Client's Topic Subscribe/Unsubscribe Procedure

To subscribe to a topic name, a client sends a SUBSCRIBE packet to the gateway with the topic name included in that packet. If the gateway is able accept the subscription, it assigns a topic alias to the received topic name and returns it within a SUBACK packet to the client. If the subscription cannot be accepted, then a SUBACK packet is also returned to the client with the rejection cause encoded in the *ReasonCode* field. If the rejection cause is *"Congestion"*, the client should wait for the time T_{WAIT} before resending the SUBSCRIBE packet to the gateway.

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

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

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

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

4.21 Client Publish Procedure

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

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

For QoS 1 or 2 PUBLISH packets the client may receive in response to its PUBLISH an error reason code:

- The *ReasonCode*= *"*Unknown Topic Alias*"*: in this case the client needs to register the topic name again before it can publish data related to that topic name; or
- The *ReasonCode*= *"Congestion"*: in this the client shall stop publishing toward the gateway for at least the time T_{WAIT}.

A Client or Gateway processes a single outbound QoS 1 or QoS 2 message at a time.

This prevents retransmitted Qos 1 and Qos 2 messages from being received out of order.

A Client MUST NOT send a Qos 1 or Qos 2 PUBLISH packet with a new Application Message until it has received a PUBACK or PUBCOMP Packet with the Packet Identifier corresponding to the PUBLISH packet previously sent [MQTT-SN-4.21-1].

4.22 Gateway Publish Procedure

Like the client publish procedure described in Section 4.21, the gateway sends PUBLISH packets with the topic alias value that was returned in the SUBACK packet to the client.

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

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

If the client receives a PUBLISH packet with an unknown topic alias value, it shall respond with a PUBACK packet with the *ReasonCode=*"Unknown Topic Alias*"*. This will trigger the gateway to delete or correct the wrong topic alias assignment.

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

A Gateway MUST NOT send a Qos 1 or Qos 2 PUBLISH packet with a new Application Message until it has received a PUBACK or PUBCOMP Packet with the Packet Identifier corresponding to the PUBLISH packet previously sent.

4.23 Keep Alive and PING Procedure

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

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

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

4.24 Client's Disconnect Procedure

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

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

4.25 Sleeping clients

Sleeping clients are clients residing on (battery-operated) devices that want to save as much energy as possible. These devices need to enter a sleep mode whenever they are not active and will wake up whenever they have data to send or to receive. The server/gateway needs to be aware of the sleeping state of these clients and will buffer messages destined to them for later delivery when they wake up.

If a client wants to sleep, it sends a DISCONNECT packet which contains a sleep session expiry interval. The server/gateway acknowledges that packet with a DISCONNECT packet and considers the client for being in *asleep* state. The *asleep* state is supervised by the server/gateway with the indicated sleep session expiry interval. If the server/gateway does not receive any packet from the client for a period longer than the sleep session expiry interval, the server/gateway will consider that client as *lost* and - as with the keep alive procedure - activates for example the Will feature.

During the *asleep* state, packets that need to be sent to the client are buffered at the server/gateway. The gateway MUST buffer application messages of quality-of-service 1 & 2.

Informative comment

The gateway may *choose* to buffer messages of Quality-of-Service 0, whilst the client is sleeping and is within its session expiry interval.

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

During the AWAKE state, for each packet the gateway sends to the client, the application messages' quality of service shall be honored, and a full packet interaction shall take place including all normative phases of acknowledgement, including any associated retransmission logic. If, during the delivery of application messages from the gateway to the client, the gateway detects a timeout in the delivery, it should transition the client state to LOST and a DISCONNECT packet with error sent to the device.

The transfer of packets to the client is closed by the server/gateway by means of a PINGRESP packet, i.e. the server/gateway will consider the client as *asleep* and restart the sleep timer again after having sent the PINGRESP packet. If the server/gateway does not have any packets buffered for the client, it answers immediately with a PINGRESP packet, returns the client back to the *asleep* state, and restarts the sleep timer for that client.

After having sent the PINGREQ to the server/gateway, the client uses the "retransmission procedure" of section 3.18 to supervise the arrival of packets sent by the server/gateway, i.e. it restarts timer Tretry when it receives a packet other than a PINGRESP, and stops it when it receives a PINGRESP. The PINGREQ packet is retransmitted, and timer Tretry restarted when timer Tretry times out. To avoid a flattening of its battery due to excessive retransmission of the PINGREQ packet (e.g. if it loses the gateway), the client should limit the retransmission of the PINGREQ packet (e.g. by a retry counter) and go back to sleep when the limit is reached and it still does not receive a PINGRESP packet.

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

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

//TODO SIMON – add some worlds around retain registration behavior

Topic Alias mappings exist only while a client is active and last for the entire session expiry interval of the active state. Therefore, the gateway must re-register any topic aliases during the AWAKE state, which will last until the last PINGRESP is issued.

Informative comment

The gateway should attempt to make the best effort to reuse the same topic alias mappings that existed during any initial associated ACTIVE states.

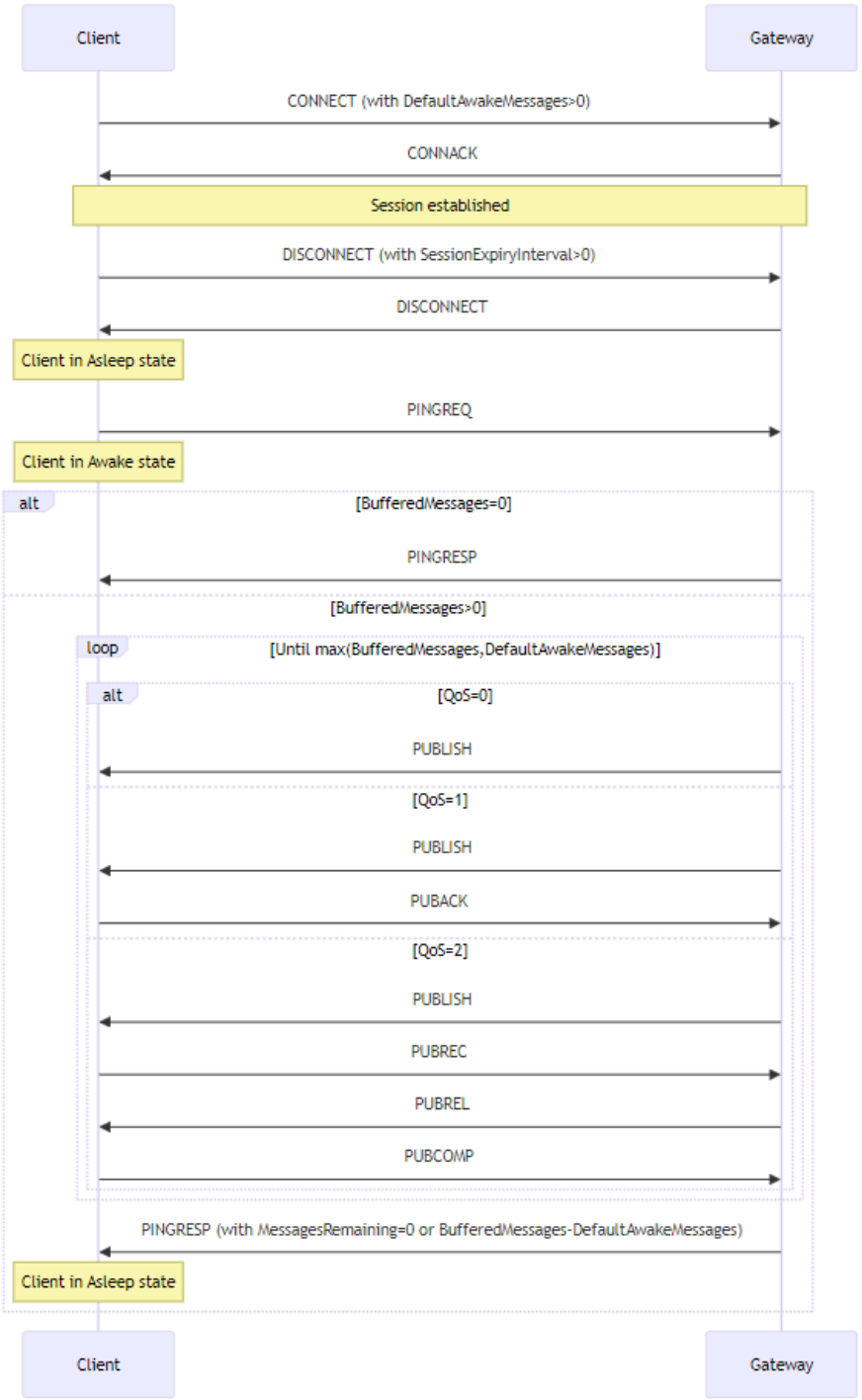


Figure 5: Awake ping packet flush

4.26 Retained Messages

If the RETAIN flag is set to 1 in a PUBLISH or PUBWOS packet received by a Server, the Server MUST replace any existing Retained Message for this topic and store the Application Message [MQTT-SN-4.26-1], so that it can be delivered to future subscribers whose subscriptions match its Topic Name. If the Publish Data contains zero bytes it is processed normally by the Server but any retained message with the same topic name MUST be removed and any future subscribers for the

topic will not receive a retained message [MQTT-SN-4.26-2]. A Retained Message with a Publish Data containing zero bytes MUST NOT be stored as a Retained Message on the Server [MQTT-SN-4.26-3].

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

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

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

Refer to section 3.1.17.2 for a definition of the Subscription Flags.

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

The setting of the RETAIN flag in an Application Message forwarded by the Server from an established Virtual Connection is controlled by the Retain As Published subscription option. Refer to section 3.1.17.2 for a definition of the Subscription Flags.

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

  Informative comment

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

4.27 Optional Features

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

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

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

5 Security (Informative)

Like the MQTT security chapter but with a focus on MQTT-SN. In particular:

- securing communications with:
  - username/password
  - challenge/response - AUTH
  - DTLS
  - Protection encapsulation

5.1 Introduction

6 Conformance

(**Note**: The OASIS TC Process requires that a specification approved by the TC at the Committee Specification Public Review Draft, Committee Specification or OASIS Standard level must include a separate section, listing a set of numbered conformance clauses, to which any implementation of the specification must adhere in order to claim conformance to the specification (or any optional portion thereof). This is done by listing the conformance clauses here.

For the definition of "conformance clause," see OASIS Defined Terms.

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

Remove this note before submitting for publication.)

Appendix A. References

[Required section.]

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

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

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

For all References – Normative and Informative:

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

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

[Citation Label]

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

For example:

[OpenDoc-1.2]

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

Reference sources:

For references to IETF RFCs, use the approved citation formats at:

https://docs.oasis-open.org/templates/ietf-rfc-list/ietf-rfc-list.html.

The most recent IETF RFC references are listed by the IETF at

https://www.rfc-editor.org/in-notes/rfc-ref.txt.

For references to W3C Recommendations, use the approved citation formats at:

https://docs.oasis-open.org/templates/w3c-recommendations-list/w3c-recommendations-list.html.

Remove this note before submitting for publication.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

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

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

[Reference]
[Full reference citation]

[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

[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/

A.2 Informative References

The following referenced documents are not required for the application of this document but may assist the reader with regard to a particular subject area.

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

[Reference]
[Full reference citation]

Appendix B. Backwards Compatibility

B.1 PUBLISH QoS -1 (Packet from MQTT-SN 1.2)

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Packet Type (0x0C)							
	PUBLISH-M1 FLAGS							
	DUP	*QoS*		*Retain*	*Reserved*	*Reserved*	*Topic Id Type*	
Byte 3	*X*	*X*	*X*	*X*	*0*	*0*	*X*	*X*
Byte 4	Topic Id MSB							
Byte 5	Topic Id LSB							
Byte 6	0x00 – Fixed Field Value							
Byte 7	0x00 – Fixed Field Value							
Byte 8 .. N	Data Or (Full Topic Name + Data)							

Table 27: PUBLISH packet

This packet is used by both clients and gateways to publish data to a topic.

> **Informative comment**
>
> If the Transport Layer supports multicast, like UDP/IP, the PUBLISH MINUS -1 packet is generally sent using the Multicast Address as destination.

B.1.1 Length & Packet Type

The first 2 or 4 bytes of the packet are encoded according to the variable length packet header format. Refer to section 2.1 for a detailed description.

B.1.2 PUBLISH Flags

The PUBLISH Flags field is 1-byte located in Byte 3 position of the PUBLISH control packet.

The PUBLISH Flags includes the following flags:

- **Topic Id Type**. This is a 2-bit field in Bit 0 and 1 which determines the format of the topic Id value.
- **QoS**: This is a 2-bit field stored in Bit 5 and 6. QoS has the same meaning as with MQTT indicating the Quality of Service. Set to “0b00” for QoS 0, “0b01” for QoS 1, “0b10” for QoS 2, and “0b11” for QoS -1. For a detailed description of the various Quality Of Service levels refer to section 4.3.

- **DUP**: 1 bit field stored in Bit 7 and has the same meaning as with MQTT. It notes the duplicate delivery of packets. If the DUP flag is set to “0”, it signifies that the packet is sent for the first time. If the DUP flag is set to “1”, it signifies that the packet was retransmitted.
- **Retain**: 1 bit field stored in Bit 4 and has the same meaning as with MQTT. The field signifies whether the existing retained message for this topic is replaced or kept.

B.1.3 Topic Id

Contains the topic id value or the short topic name for which the data is published.

B.1.4 Data

The published data.

Appendix C. Security and Privacy Considerations

[Optional section.]

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

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

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

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

The MQTT SN protocol is optimized for implementation on low-cost, battery-powered devices with limited processing and storage resources. The capabilities are kept simple and the specification allows for partial implementations. Device identities are typically created at manufacturing, eliminating the need for special configuration at deployment. MQTT-SN can work in isolation from other networks or in conjunction with MQTT.

MQTT-SN Client and Gateway/Server implementations SHOULD offer Authentication and Authorization options. Furthermore, the confidentiality and authenticity of the MQTT-SN messages can be provided by the underlying transport or can be obtained by encapsulating the MQTT-SN messages into the PROTECTION packet.

Applications concerned with critical infrastructure, personally identifiable information, or other personal or sensitive information are strongly advised to use these security capabilities.

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

Appendix D. Acknowledgments

[Required section.]

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

Remove these yellow notes before submitting for publication.

D.1 Special Thanks

Note: This is an optional subsection to call out contributions from TC members. If a TC wants to thank non-TC members then they should avoid using the term "contribution" and instead thank them for their "expertise" or "assistance".

Substantial contributions to this document from the following individuals are gratefully acknowledged:

[Participant Name, Affiliation | Individual Member]

D.2 Participants

Note: A TC can determine who they list here, however, Observers must not be listed. It is common practice for TCs to list everyone that was part of the TC during the creation of the document, but this is ultimately a TC decision on who they want to list and not list.

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

[Participant Name, Affiliation | Individual Member]

Appendix E. Revision History

[Optional section.]

Revisions made since the initial stage of this numbered Version of this document may be tracked here.

Note: If revision tracking is handled in another system like github, provide a link to it instead of using this table, if desired. Remove this note before submitting for publication.

Revision	Date	Editor	Changes Made
WD-01	[27th February 2020]	[Andrew Banks]	[Merge Initial Document and Input Specification]
WD-02	[4th April 2020]	[Andrew Banks] [Rahul Gupta]	[Terminology, DataTypes, CONNECT packet] [Specification Diagrams]
WD-05	[21st February 2021]	[Simon Johnson]	[Packet Diagrams, Bit Tables, Field Definitions]
WD-06	[10th March 2021]	[Simon Johnson]	[Sleeping client operational behavior, Terminology changes, 13 JIRA resolutions added to specification, Section numbering changes]
WD-07	[15th March 2021]	[Simon Johnson]	[Added 4 byte (32 bit) integer description]
WD-08	[26th March 2021]	[Simon Johnson]	[Added max packet size to CONNECT, Added Session Expiry Interval to CONNACK, Removed ZigBee references, Removed capabilities flag from CONNECT, AUTH packet added along with Authentication operational behavior. Standardized page margins]
WD-09	[05th May 2021]	[Simon Johnson]	[Added long topic type to topicIdTypes, updated PUBLISH to accommodate new topic type, added topic type matrix]
WD-10	[October 2021]	[Simon Johnson]	[Document format aligned with core specification, removal of introduction, addition of packet ID table, adding error code]
WD-11	[October 2021]	[Simon Johnson]	[MQTT-SN Architecture moved into operational behavior, removal of variable integer definition, addition of session state section, normative comments added to sleeping client operational behaviour]

WD-12	[November 2021]	[Andrew Banks]	Rework 1.5 Background
WD-13	[November 2021]	[Simon Johnson]	[Move Authentication and Retained messages into operational behavior, rationalized tables and figures, separated packet definitions of similar structures into distinct sections.]
WD-14	[Decmeber 2021]	[Simon Johnson]	[First implementation attempt, Fixed table references, Fixed PingResp packet]
WD-15	[December 2021]	[Simon Johnson]	[Tara added as editor, return code additions]
WD-15	[February 2022]	[Tara Walker]	Changed Return Code nomenclature to be more consistent w/5.0. Added Reason Codes to each control packet type
WD-16	March 2022	[Tara Walker]	Updated WILL*Types to correct Packet Type. Added Global Flags Table to Section 2. Updated each Control Packet Flags in Section 3 adding missing Flag Sections. Formatting: Auto update of Table numbering, Auto update of WD Revision numbering for footer.
WD-17	April 2022	[Simon Johnson]	Updated use of topic name and topic filter to be aligned with MQTT 5. Topic alias becomes topic alias type. Added quality of service protocol flow as it differed to MQTT 5 (inflight). Conformance references removed as these will need to be wholly owned OR externally referenced.
WD-18	June 2022	[Tara E. Walker]	Updated items based upon the feedback from Alex Kritikos.
WD-19	August 2022	[Simon Johnson]	Remove change tracking as document was becoming unworkable.
WD-20	September 2022	[Simon Johnson]	Integrate feedback from committee meeting relating to the work by Miroslav Prymek. Added resolution of CONNACK session present per MQTT 585
WD-21	October 2022	[Simon Johnson]	Client States section added to describe the 5 states. Updated the state transition diagram to accommodate new disconnect field and new transitions between Awake -> Lost and Asleep -> Disconnected.

			Security section added. Figure 2 – MQTT-SN Architecture diagram updated. Font updated to Arial from bespoke font. QoS -1 – Section added to the QoS chapter (NOTE: updated text to allow for bi-directional -1 PUBLISHING). Introduction of Exponential backoff algorithm. Applied issue issue 587 (max messages set in CONNECT flags).
WD-22	November 2022	[Simon Johnson]	Integrate MQTT 591 (sleep behavior) Replace instances of “return code” to “reason code” PINGREQ timeout aligned with Tretry (15 seconds) from the ill defined “reasonable amount of time” Exponential Algo fix (using the factor n assuming it was the product!) Client Identifier size clarification. Publish variants added; distinguish variant based on QoS field to save 2 bytes for single flight PUBLISH packets. Incorporated B4. Into retry timer.
WD-23	December 2022	[Simon Johnson]	CONNECT Client Identifier Informative and Normative définition update. CONNACK Client Identifier Informative and Normative définition update. CONNACK reason codes updated. KeepAlive boundary specified removing 0 as an option per the committee call. Added Session Expiry “reasonable” setting statement. Added sequence diagrams for CONNECT, CONNECT with WILL, CONNECT with AUTH. Network Connection Section (IANA Omitted but we need to add this to agenda)
WD-24	December 2022	[Simon Johnson, Davide Lenzarini, Ian Craggs]	Removal of Network Connection references. Modified PUBLISH -1 & 0 tables to remove topic length field

			Modified PUBLISH 1 & 2 tables to remove topic length field Changed Data field description on the above Updated sleeping device section Ensured the references to the Packet Length and type section was consistent in all packet types.
WD-25	January 2023	[Simon Johnson]	Broken out PUBLISH -1 into its own packet type Disconnect flags field moved and added existence flags for optional fields Introduction titles changed to better sign post where the information resides in the document
WD-26	May 2023	[Simon Johnson, Davide Lenzarini]	Backwards compatible PUBLISH -1, new OOS Publish message to repace it. Removal of security section to allow to rewrite.
WD-27	November 2023	[Simon Johnson, Davide Lenzarini]	Network Transport Layer chapter updated to define the impact of lower layers features on the MQTT-SN protocol. Replaced the term MQTT-SN “connection” with the term “Virtual Connection”.
WD-28	December 2023	[Davide Lenzarini, Stefan Hagen]	Ensured document structure is intact and replaced table footnotes with simple text tags and a subsequent notes listing.
	February 2024	[Ian Craggs, Simon Johnson]	Issue 560 resolution - full reason code table and add reason code fields to PUBREC, PUBREL and PUBCOMP. Duplicate reason code tables removed from packet descriptions. Will Data Sent in CONNECT Auth Data Sent in CONNECT & CONNACK Suback granted QoS 0,1,2 now reason codes not flags. Moved PUBLISH -1 to new Backward compatibility appendix
WD-29	March 2024	[Ian Craggs, Davide Lenzarini, Simon Johnson]	Update Terminology section. Add Operational Behavior sections from MQTT 5.0. Workshop revisions to Operational Behavior responding to Davide’s review.

Appendix F. Implementation Notes

F.1 Support of PUBLISH WITHOUT SESSION

Because PUBLISH WITHOUT SESSION could be sent at any time by clients (even with no Virtual Connection setup) a transparent GW needs to maintain for those packets a dedicated MQTT connection with the server. An aggregating or hybrid GW may use any aggregating MQTT connection to forward those packets to the server.

F.2 “Best practice” values for timers and counters

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

Timer/Counter	Recommended value
T_{ADV}	Greater than 15 minutes
N_{ADV}	2 -3
$T_{SEARCHGW}$	5 seconds
T_{GWINFO}	5 seconds
T_{WAIT}	Greater than 5 minutes
T_{RETRY}	Implement section F.4 with a starting value of 1 second after an initial wait period of 5 seconds. So the first retry will be ~6 seconds.
N_{RETRY}	3 – 5
$M_{BACKOFF}$	60 seconds

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

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

F.3 Mapping of Topic Alias to Topic Names and Topic Filters

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

F.4 Exponential Backoff

The following error handling strategy should be used for networked devices to avoid overwhelming recipient network entities whilst providing for efficient reestablishment handling. The client shall periodically retry a failed packet with increasing delays between attempts, constrained by a max retry time and interleaved with a suitable seed of randomness.

Algorithm:

An exponential backoff algorithm retries requests exponentially, increasing the waiting time between retries up to a maximum backoff time. For example:

1. Initial packet sent.
2. If the operation fails, wait 1000 + (random number) milliseconds (ran) and retry the operation.
3. If the operation fails, wait 2000 + (random number) milliseconds (ran) and retry the operation.
4. If the operation fails, wait 4000 + (random number) milliseconds (ran) and retry the operation.
5. Continued, up to a maximum backoff $M_{BACKOFF}$.
6. Continue waiting and retrying up to some maximum number of retries, but do not increase the wait period between retries.

The wait time is min(((2^n * sf) + ran), max) with n incremented by 1 for each iteration (or operation) and the scaling factor (sf) being set to some reasonable value (suggested 1000 as in the example above).

The random number helps to avoid cases where many clients are synchronized by some situation, and all retry at once. The value of the random number ran is recalculated after each retry. The random number (ran) should be no larger than the scaling factor (sf).

Appendix G. Notices

[Required section. Do not change.]

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

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

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

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF.

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

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

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

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual

property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

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