Oasis, Hursley, January 2016.

Andrew Banks

MQTT 256 Message Format indication and message metadata in general.

MQTT 249 Add expiry capabilities to MQTT.

MQTT 234 Shared Subscriptions.

MQTT 256 Message Format indication and message metadata in general.

Fixed Header	Variable Header	Message Payload
Existing MQTT 3.1.1 publish packet		
Pub(3),Flags,RL	TopicName, Packet Identifier (if Qos 1,2)	Payload
Proposed publish packet		
Pub(3),Flags,RL	MetaData, Packet Identifier (if Qos 1,2)	Payload

RL Remaining Length

MQTT 256 metadata encoding

Variable header

TemplateId (byte)	Id/[Value]	Id/[Value]	…	Delimiter	[PacketId (2 bytes)]

- TemplateId byte indicates metadata cached in both the sender and receiver,
- Up to 256 possible templates for each direction.
- Add a field in Connect/Connack variable header to indicate receivers maximum cache size, this may be zero.
- In case of cache size zero, template Id is omitted, delimiter must be 7F, no templates are cached.
- Id encoded in the same way as remaining length in the fixed header,
- Defines both the value key and data type,
- Large set of possible Id types but the lower 127 Id's each occupy a single byte.
- Cached values for each templateId are replaced with the new values and the cache updated each time the TemplateId is used.
- Cached values are discarded when the network connection ends.
- If packet loss is possible, all non default fields must be flowed, no valid cache may be assumed.

MQTT 256 metadata identifiers

Metadata Id	Usage	Value data type	Initial value
1	TopicName	UTF8 String	None (zero characters).
10	Payload format	One byte 0=bytes, 1=UTF-8 string, 2=JSON …	0
7E	Delimiter	void	void
7F	Delimiter , clear cached id after use.	void	void

Allowed data types are:

UTF8 String	Id=01-0F, 8000-807F
One byte	Id=10-1F, 8100-817F
Four byte unsigned integer	Id=20-2F, 8200-827F
Void	Id=70-7F, 8300-837F

Or, make all unused id’s a protocol error.

MQTT 256 Publish packet examples

Existing publish Topic="abc", Qos=1,PacketId=4, payload ="aaa".	320a0003 61626300 04616161
Same publish, in the new format using template 0.	320d0001 00036162 637E0004 616161
Another publish on the same topic, reuse template 0.	3207007E 00046161 61
Now publish Topic="def", message format string, Qos=1,PacketId=5, payload ="aaa" using template 1.	320f0101 00036465 6602017E 00050003 616161
Another publish on the same topic, reuse template 1.	3209017E 00050003 616161

MQTT 256 Alternative 1, publish the template as a retained message .

Publish retained topic=$meta/topicNameXYZ payload=Metadata
Publish topic=topicNameZYZ

- No modification to the specification, just document the convention.
- Any authorized client may publish the metadata at any time.
- The receiving client subscribes for the $meta topic each time it receives a new topicName.
- Many possible encoding schemes.

But

- Existing clients and servers will disconnect if they see the $meta/… topic
- Only useful for long lived metadata such as Payload format because it is not possible to coordinate the metadata with a specific publish packet.
- An update to the metadata cannot be coordinated with a particular publication because it may be published by a separate client and there is no ordering guarantee across topics.

MQTT 256 Alternative 2, augment the publish packet with metadata in the variable header.

Fixed Header	Variable Header	Message Payload
Augmented publish packet		
Pub(3),Flags,RL	TopicName, Packet Identifier (if Qos 1,2),Metadata flags,MetadataX,MetadataY	Payload

- Looks like a less disruptive change than the proposal.
- The Metadata flags indicate which metadata fields are present.
- Many possible encoding schemes.

But

- No caching.
- Publish packet is always larger then before, because the Metadata Flags must be flowed.
- The client and server code still needs to be modified to use the new packet format.

MQTT 256 Alternative 3, augment the publish packet repeat using packet Identifier.

Fixed Header	Variable Header	Message Payload
Augmented publish packet		
Pub(3),Flags,RL	TopicName, Packet Identifier, Metadata flags, MetadataX ,MetadataY	Payload
Pub(3),Flags,RL	TopicName, Packet Identifier, RepeatMetadata flag	Payload

- The Metadata flags indicate which metadata fields are present.
- Metadata flags also indicate whether to cache.
- Metadata flags indicate repeat previous packetId metadata.
- Many possible encoding schemes.

But

- Flow packet identifier for Qos=0.

MQTT 256 decision points.

- Use caching or repeat metadata on each packet?
- Use type value pairs and difference encoding or flags to indicate presence?
- Overload packet Identifier to indicate cache line or use separate cache line identifier (Template id)i.
- Include TopicName in the metadata or keep separate.

MQTT 256 conclusions.

- Add metadata fields to the publish packet variable header using either id/value pairs or flags to indicate the presence of positional metadata fields.
- Include a metadata field to carry application data.
- Use a zero length topic name to indicate that the previous topicName should be used on the current packet in the same TCP connection. This is a basic mechanism for caching the topicName and avoiding repeated transmission where a single topic is used repeatedly.
- No caching to be supported for other metadata fields.

MQTT 249 Add expiry capabilities to MQTT.

- Expiry of messages.
  - Add time to live value to publish packet metadata.
  - The receiver may reject (nack) the message if the time to live is too long??
  - The receiver may unilaterally reduce, but not increase, the time to live??
  - The sender is not expected to send messages if more the time to live has passed.
  - If a Qos=2 publish is sent then the PubRec, PubRel, PubComp exchange must be completed regardless of the time to live.
  - The sender should store the time of day when the message ceases to exist and compute the remaining time to live just before transmission. The receiver should calculate the time of day of the message expiry immediately on receipt.
  - Four byte positive integer in seconds gives a maximum lifetime of approximately 136 years. Could be 1 byte 2**N seconds e.g. 1s,2s,4s … 2**69 years? No.

Metadata Id	Usage	Value data type	Initial value
20	Message Time to live in seconds	Four byte positive integer	0, forever

MQTT 249 Add expiry capabilities to MQTT.

- Expiry of session state.
  - Like cleanSession=true, but the action is delayed in both the client and server until some time after the network connection is terminated.
  - Enables the client to continue a session across a network break or temporary disconnect, within a time limit.
  - Keeps the ability to clean up unused state in the client and server
  - Deprecate use of the cleanSession flag, it must now be 0.
  - Add a four byte time to live into the connect variable header after the keep alive timer meaning the session expiry time in seconds.

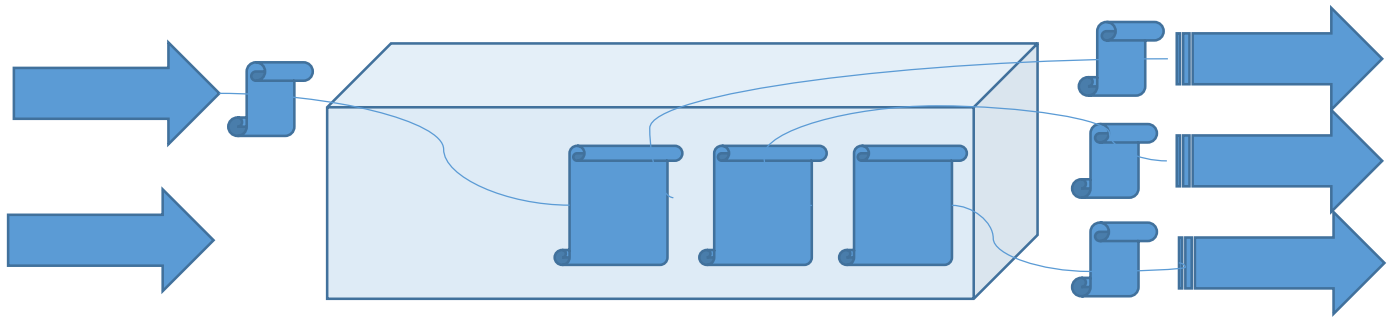
    ProtocolName=MQTT, ProtocolVersion=6?, ConnectFlags, KeepAlive, SessionExpiry

MQTT 249 Add expiry capabilities to MQTT.

- Expiry of session state.
  - SessionExpiry=0 means clean immediately on TCP disconnect. Any existing session state is discarded on connect regardless of prior SessionExpiry or CleanSession usage, as with cleanSession=1 today.
  - SessionExpiry=FFFFFFFF means keep indefinitely as with cleanSession=0 today.
  - If the client connects and sets a SessionExpiry time other than 0 or FFFF, then re connects using the V3.1.1 protocol using cleanSession=0 before the session has expired, then the session state is kept indefinitely.
  - Where SessionExpiry is not zero, the Session Present flag on ConnAck indicates whether the server had any session state for the client, as today.
  - If disconnect is sent by the client, this has no special meaning, the session state is retained for the SessionExpiry time.
  - Sending of the will message is delayed until session expiry has passed, unless a normal disconnect packet is received by the server.
  - The disconnect packet should enable a new expiry time to be set, subject to server constraints.
  - The old clean session flag could be used to indicate to the server that the client is willing for its data to be kept in volatile storage.
- Expiry of subscriptions.
  - Not needed if the session state expires.

MQTT 234 Shared Subscriptions. Motivation and model.

- Enable a set of clients to share the consumption of messages.
- The set of subscribers is created by the use of a common share name.
- Shared subscriptions provide a facility similar to point to point messaging in that the processing of messages is not limited to the availability or capacity of a single consumer.
- The share name is analogous to the queue name in that it names the list of messages to be processed.
- Message ordering among members of the group size >1 is not defined, hence no ordering guarantees are made even for group size=1.



Producers Queue Consumers

MQTT 234 Shared Subscriptions. Specification.

- Use a new topic filter syntax eg:

  `$share/shareName/topicFilter`

- shareName defines the set of consumers using the topicFilter it must not contain any “/” characters.
- Each client using this type of topic filter above will receive a subset of the messages, subject to the normal Qos contraints.
- This specialised topic filter takes the place of the normal topic filter in the subscribe packet.
- The "$share/" prefix uses a reserved $ name and will therefore not cause a conflict with other topic filters.
- There is no guarantee of fairness as to how messages are allocated to consumers.

Note, jira 234 uses “:” instead of “/” as a separator.

MQTT 234 Shared Subscriptions. Specification continued.

- Subscribing clients will be a member of the share only if both the shareName and topicFilter are identical. A different shareName with the same topicFiler will create a different share. Similarly a shareName followed by a different topic filter will create separate share.
- Servers may decline to support shared subscriptions by not accepting subscribe packets containing topic filters starting with the "$share/" prefix. However, if they do accept the "$share/" prefix they must follow the specification.
- The subscribing clients may make both non durable and durable subscriptions by setting session expiry (was clean session).
- The shared subscription ceases to exist if the number of subscriptions reaches zero. If the shared subscription ceases to exist the messages stored by the server are deleted, just as with a non shared subscription.
- A good server implementation will avoid allocating messages to subscribers with durable subscriptions while the client is disconnected. Instead the messages will be allocated when a suitable client connects.
- A Qos=1 message should be reallocated to another client sharing the same subscription if the network connection to the client is lost before the PUBACK is received.
- Qos=0 and Qos=2 messages must not be reallocated once allocated.