Using CAPEXPRY to prevent system queues filling up when using MQTT durable subscriptions

FSWU\_Chris\_Matthewson |Dec 22 2016 Updated

One of the features of MQTT is the ability of clients to make durable subscriptions. If a client has made a durable subscription, this means that it can disconnected and the subscription will remain active. Any messages published to the topics subscribed to by this client will be kept by the queue manager until the client reconnects - at this point, all messages sent while the client was disconnected will be delivered. Durable subscriptions are created for a MQTT client if the connection is made specifying option '**cleanSession**' set to **false**.

While this is very useful, it can lead to some problems with message buildup. All of the messages intended for disconnected clients have to be stored somewhere, so they can all be delivered when the client reconnects - the place they are kept is the queue **SYSTEM.MQTT.TRANSMIT.QUEUE**.

The default maximum size of the SYSTEM.MQTT.TRANSMIT.QUEUE is 100,000. This limit can be reached quite quickly if one or a combination of these are the case:

* there are a large number of disconnected clients with durable subscriptions
* these clients have durably subscribed to a large number of topics
* the topics subscribed to by these clients have a lot of messages published to them

If this queue fills up and reaches its maximum depth, any messages published to topics durably subscribed to by disconnected clients cannot be stored for later delivery to those clients - if this happens, the MQXR service will generate FDCs and log an error in the mqxr.log

Until now, there have been a couple of ways to work around hitting this issue. The first is just to increase the maximum depth of the SYSTEM.MQTT.TRANSMIT.QUEUE. While this can delay hitting the problem, there is no guarantee that the new limit will not be hit at some later time. It is also possible to ensure that all published messages have some expiry set on them - if this is the case, messages which are being stored on the SYSTEM.MQTT.TRANSMIT.QUEUE will be removed once the specified expiry time has elapsed. This is only a suitable solution if all publishing applications can be modified to ensure this expiry is set on all messages.

From 8.0.0.4, a new attribute **CAPEXPRY** has been introduced that can be used to alleviate problems of messages building up and hitting maximum depth limits. Setting this attribute on a destination means that when a message is put to that destination, the expiry time of the message will be limited to the value set in CAPEXPRY. If no expiry was specified on the message, or if an expiry was set, but it was greater than the CAPEXPRY value, the expiry value will be set to that in CAPEXPRY. If a message had an expiry value smaller than CAPEXPRY this is not changed. This means that the administrator can effectively set, or limit, the expiry times of messages. There are more details on this attribute on the Knowledge Centre [here](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.ref.dev.doc/q097495_.htm) and [here](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.ref.adm.doc/q085770_.htm).

In the case of MQTT, this can be set generally, or different parts of the topic tree can have different expiry values set on them.

To set CAPEXPRY for a topic, create a topic object for the part of the topic tree to set the expiry for, with CAPEXPRY set as a custom attribute - for instance this can be created with the runmqsc command:

**DEFINE TOPIC(MQTT.EXPIRY) TOPICSTR('/testTopic') CUSTOM(CAPEXPRY('300'))**

This will set a maximum expiry for messages sent to /testTopic of 30 seconds - values of CAPEXPRY are set in tenths of a second, the same as MQMD expiry values.

Now if there are any currently disconnected clients which are durably subscribed to topic '/testTopic', any messages published to '/testTopic' with no expiry set will be put to SYSTEM.MQTT.TRANSMIT.QUEUE to be stored for delivery when the clients reconnect, but now with an expiry time set on each message of 30 seconds. Once 30 seconds has elapsed, the messages will be removed from the queue.

This means that there will never be any messages originally published to that topic older than 30 seconds - so when the client reconnects it will receive all messages sent during the preceding 30 seconds.

If a message is put to a part of the topic tree for which there is no topic object defining an expiry cap, there will be a check to see if one has been defined for its parent - continuing until either one is found or reaching the root. If no expiry cap is set, the default will be used. This is the equivalent of setting NOLIMIT as the CAPEXPRY value - there is no limit on the expiry time of messages put using this object.

So, it is possible to define branches of the topic tree as having specific expiry times by defining the CAPEXPRY attribute on topic objects where the TOPICSTR is the 'base' node for each branch, for instance:

**DEFINE TOPIC(QUICKEXPIRY) TOPICSTR('/quickExpiry') CUSTOM(CAPEXPRY('100'))**

**DEFINE TOPIC(SLOWEXPIRY) TOPICSTR('/slowExpiry') CUSTOM(CAPEXPRY('600'))**

**DEFINE TOPIC(NEVEREXPIRE) TOPICSTR('/neverExpire') CUSTOM(CAPEXPRY('NOLIMIT'))**

Having control over the expiry value of messages intended for disconnected durable subscribers like this makes it much less likely that the maximum depth limit of the SYSTEM.MQTT.TRANSMIT.QUEUE will be reached.