Thingstream has commercially implemented a gateway implementation of MQTT-SN in Java, with client implementations in both C and Java. We put the protocol over USSD (3gpp signalling channel) and UDP. We have found the specification to be concise and fairly thorough. We noted most of the complexity and discussion points during implementation revolved around topic aliasing contracts between gateway and device.

Implementation Notes

**Topic Id Namespaces**

- The implementation notes “strongly suggest“ using a separate namespace per device for topicId mapping. From experience this should be mandated. *NB: our predefined namespace is global, and the client mappings are on a per client basis.*
- Some of our device manufacturers have commented how it is confusing to have overlapping topicIds in the predefined namespace match topicIds in the normal namespace but mean different things. *NB: to avoid customer confusion our broker starts the predefined topicIds at 100 and normal topicIds at 1. Should predefined and normalIds be allowed to clash?*

**Publish -1**

Whilst the common use case is for a device to only publish at -1 when it has no connection, we found a surprising number of device manufacturers wanting to use this feature as follows; long lived sleeping sessions collecting messages, and a device not wanting those messages, but wanting to send a message to the broker without the overhead of waking up fully. Potentially this
acted as a panic message. I confirmed with the spec and found.. “Because PUBLISH messages with QoS level -1 could be sent at any time by clients (even with no connection setup”). This statement doesn’t make it definitive that a device cannot do the above so we allowed it, and it came in handy!

**Topic Aliasing**

Generally speaking the issues noted regarding topic aliasing involved the client trying to ascertain which topicId type was in use for a given message / interaction when there were a number of possible options.

- In the case where a **PREDEFINED** topicId exists on the broker, but NOT on the client (a common use case when client firmware is outdated), the spec suggests that an unrecoverable error is generated. If, however, a client issues a **REGISTER** against the topic that happens to also be predefined, is this allowed? If so, which topicIdType should be used for delivery of broker-to-client messages? *NB: we decided to allow this and use an order of precedence when publishing messages from broker to device;*
  - *Normal (if confirmed in session by REGISTER)*
  - *Predefined (if specified in broker) - which would result in a PUBACK error*
  - *Short name (if <= 2 bytes AND no predefined)*
  - *REGISTER a normalTopicId in session and repeat..*

- What happens when a client calls **REGISTER** on a topic name which falls within the purview of a short name. Should this lead to a **REGACK** error or should the normal registration be allowed?

- Further to the above, only normal aliases can be transported by the **REGACK** message. Where a predefined topic alias ~may exist or indeed a short alias would apply, would it make sense for the broker to be able to inform the client of the specification it intends to use for **PUBLISH** messages thereafter?

- In the case of short topics, when the topic in question is a single character, the specification states; “**Short topic names are topic names that have a fixed length of two octets**”. Does that mean that single character topic names cannot be used in short topics? *NB: We assumed they could be and used zero padding with network byte ordering.*

**Asleep State**

- Should QoS 0 messages be stored for a device on the broker when it is in **ASLEEP** state. *NB: We assumed that they should, but this was not stated clearly.*
• When a device goes into the **ASLEEP** state by issuing a **DISCONNECT** with duration, there is a lack of clarity on what topic "session" state is maintained on the client. *NB: given our clients maintained topicId registry's in ephemeral memory, we cleared these down on expectation that a new REGISTER will be received upon incoming PUBLISH during a ping flush cycle, we assumed this was the correct behaviour but more clarity in implementation notes would have helped.*

**Queue State**

• There is no consideration given for the size a device message queue could get during an asleep cycle. When a battery constrained device wakes up, issues a **PINGREQ** in anticipation of receiving it's outstanding messages, we saw occasions where a device became “stuck” receiving messages in the flush cycle, since it had no way of knowing what its outstanding queue size was. *NB: we elected to add a broker monitored timer to the flush cycle so the gateway would only flush messages for a maximum period of time before issuing a **PINGRESP**. It was suggested that a PINGREQ could contain a field to specify a maximum number of messages to receive in this window.*

• Further to the point above, it was discussed that even when in an **ACTIVE** state, it would be good for a device to get an idea if there were lots of messages in it's queue, especially in the case where the protocol is constrained by the transport layer *NB: we had discussed allowing a **PINGRESP** to optionally allow the broker to specify how many messages were in a devices’ queue. This gives the client more control to decide what it may like to do, ie. wait for messages, send messages, or reconnect entirely and clean the queue.*

**Keep-Alive**

• The specification allows for 16-bits to transport its duration for both **ASLEEP** and **CONNECT** timeout. Aligned with MQTT 3.1.x, this is defined as **SECONDS** time-unit. It allows for a session duration OR sleep duration of a maximum **18 hours 12 minutes and 15 seconds**. This presented a problem for us when it came to devices wanting to sleep to protect their battery life. In reality, low power devices in the field may well want to stay asleep for far longer than is allowed for by a 16-bit SECONDS time-unit. *NB: We changed the meaning of duration to be a **MINUTE** time-unit which allowed us ~45 days sleep time.*

**Error Messages**
- We felt there was a lack of scope for details and verbose error messages. We felt that error messages should be able to be sent with all ACK operations and also with DISCONNECTS.

Paho Library Comment(s)

We started by exploring the reference libraries;

**eclipse/paho.mqtt-sn.embedded-c**: Paho C MQTT-SN gateway and libraries for embedded systems. Paho is an Eclipse IoT project.

We noted in both the Java and C libraries, there was no implemented support for messages that required 2 octets to transmit their length, so when topic names or payloads got larger the codecs failed, we ended up rewriting the codecs.

<table>
<thead>
<tr>
<th>Author</th>
<th>Version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon Johnson, Keith Rautenbach, Phil Dubach, David Denholm</td>
<td>1.0</td>
<td>Initial version</td>
</tr>
<tr>
<td>Simon Johnson</td>
<td>1.1</td>
<td>Version table added. Clarified PAHO library issue. Added error messages</td>
</tr>
</tbody>
</table>