BIGDATAWIRE
DATA SCIENCE • AI • ADVANCED ANALYTICS

HighByte

When MQTT Falls Down: 5 Common Pitfalls in UNS Design

AUTHOR

Aron Semle, HighByte Chief Technology Officer

This article first appeared in its original form on bigdatawire.com in July 2024.

When MQTT Falls Down: 5 Common Pitfalls in UNS Design

Introduction

The Unified Namespace (UNS) is here to stay. In concept, it's a single location that represents the real-time state of your factory, using open standards. Evangelized by Walker Reynolds, President of 4.0 Solutions and the Board Chairman of Intellic Integration, the UNS is appealing because it strikes at the core problem we've faced in factories for decades: Getting access to machine data is hard.

In practice, the UNS is often an MQ Telemetry Transport (MQTT) broker with an ISA-95 topic hierarchy (Site/Area/Line) and edge applications to convert industry protocols (e.g., OPC UA, Modbus, Ethernet/IP, SIMATIC STEP 7) to contextualized, human readable JSON payloads or Sparkplug B.

But UNS as a concept is broader than a single technology. You could build a UNS using OPC UA, SQL, or many other technology stacks. So why is MQTT so common?

MQTT is simple. It's report by exception. It has a flexible topic hierarchy that's easy to understand, and it puts little to no constraints on the data, making it very flexible.

These qualities make MQTT an excellent choice for real-time machine data, but like all technologies, it has limitations. There are a handful of UNS design patterns we've seen that create challenges for MQTT, and other industrial data patterns that aren't an ideal fit. The purpose of this article is to discuss these in more detail so that you can understand the pros and cons and make informed decisions based on your unique environment.

01

Using MQTT As a Tunnel

MQTT is designed to be 1:N or N:1, meaning a single producer of data can have many subscribers listening, or many producers can have a single subscriber listening. This design is great for the UNS, but what if, for example, you're simply trying to get data between your SCADA system and Snowflake? This use case is 1:1, a single producer and a subscriber.

There are some advantages that lead customers down the path of using MQTT for these use cases.

- In the future, you may need other applications to consume the data. MQTT easily enables this;
- MQTT connects outbound from a secure network to an insecure network (e.g. DMZ) not requiring any open inbound firewall ports on the secure network (i.e. you don't need to talk with IT).

But if you don't have near-term plans to leverage the data in other applications, is it worth adopting MQTT? If you're spending a lot of time customizing the data payload for the end application, it might be a sign that you're using MQTT as a tunnel. This is even more apparent if you've adopted other technologies like Sparkplug B to enable the tunnel. Sparkplug B is a great enabling technology between devices and SCADA, but it creates integration challenges as you move up the stack.

The hidden cost of the tunnel solution is the adoption of an MQTT Broker, protocol converters, and the need to secure and support that stack. In software development, we call this technical debt.

Maybe this is OK, but when I see this pattern, my general guidance is to design a solution that can easily enable MQTT data access if needed, but not to require the technology until you're leveraging its benefits.

02

Publish, to Subscribe, to Publish (Repeat)

You often see edge data that isn't well formatted for MQTT. It either exists in a proprietary format, or maybe it isn't contextualized enough. In these patterns, an application sends raw data to a topic (e.g., /mytopic/raw) and then another application subscribes to the topic, transforms the data, and publishes on a separate topic (e.g., /mytopic/cleaned). This pattern might continue, with subscribing to the cleaned topic and publishing an aggregate topic, alarm topics, etc. Rinse, wash, and repeat.

If you've ever been fishing and had your line tangle, this is how I picture this pattern. The dependencies between topics become hard to track as the MQTT topic namespace becomes cluttered. This makes it hard to identify and debug failures.

The existence of this pattern isn't inherently bad, but it may be a sign that you'd benefit from doing more data preparation at the edge before publishing to MQTT.

03

Transactions Through a UNS

Transactions are a common data pattern in manufacturing. The most common example of transactions are writes. For example, you may want to write set points, clear an alarm, or some other function.

MQTT was developed in the 90s for the Oil and Gas industry, classically known as SCADA or "Supervisory Control and Data Acquisition." The key word is Supervisory. MQTT can publish to a topic as a way to issue a write, but given that it's a one-to-many protocol, there is no way to easily get a write response. Sparkplug B does this with Command (CMD) messages, which are sent on a unique topic, but to know if the write succeeded, you must monitor the data published by the device to see if a value changed. This may work for Supervisory Control, but inside a factory, it's often important to know immediately if a write succeeds or fails to take corrective action.

There are some clever patterns to try and simulate transactions over MQTT. For example, a common pattern is to issue a write on a topic with a unique transaction ID (e.g., writes/txid/123). The client then subscribes to another topic with the same unique ID to get the response (e.g., writes/response/txid/123). It's clever, but it's not a true transaction, it requires both clients to understand the protocol, and—worse—it quickly pollutes the MQTT topic namespace by creating new topics for each write.

REST or OPC UA are better suited for request/response interactions. They can be synchronous, meaning you immediately know the status of the write and can act accordingly.

If you're trying to do writes through the MQTT Broker, it might be a sign that you're misusing the technology.

04

Large Data Sets

Factories have a lot of data. Systems like historians and SQL databases can easily grow to terabytes in size. There are many use cases where you may want to move all or some of this data between applications.

In cases where an MQTT Broker is already available in the technology stack for real-time data, customers may try and use it for historical workflows. MQTT is limited to 256MB message size, which, in fairness, is pretty large. But MQTT is optimized for small to mid-sized messages delivered very quickly, not for moving large payloads infrequently.

The result is an inefficient data exchange that could impact performance of more time sensitive, real-time data in the broker.

This is made worse if publishers use the MQTT retained feature. Retained is useful when a new client wants to know the most recent publish on a topic, but if this publish is 200MB+ in size, it's problematic. The broker ends up storing and replicating the large message either in memory or on disk.

In most cases, SQL, historian, and large file data flows are better suited for other technologies like REST, FTP, etc., and should not be tunneled through MQTT.

05

Infrequently Used Data

Some data in a PLC is needed in near real-time, updating every second or faster. But there is a lot of other data that is needed less frequently, if at all. For example, maybe the PLC has registers that hold debug information about the last batch that was processed. This information is helpful in the case of an error, but in general, it is diagnostic information that isn't needed in real-time.

If MQTT or Sparkplug B are the only data paths you have to the PLC, this data must be configured and published continuously for it to be available in the event it's needed. If it's not, this will require someone to reconfigure the data exposed by the device, which depending on location and availability, could be challenging.

The root cause of this problem is that MQTT is report-by-exception and doesn't have a way to expose what topics/data are available or control what data is sent to consumers. Other protocols like OPC UA solve this by exposing a browse interface and allowing clients to browse the data that's available, select what they need, and determine how often to consume it. This approach is generally better when data is needed on-demand,

If you're forced to publish data to MQTT that is infrequently or never used, it might be a sign that you need more flexibility than what out-of-the-box MQTT provides for those use cases.

Conclusion

MQTT is a key enabling technology that has driven UNS adoption. It's a great solution for real-time machine data in the UNS, providing an open and flexible way to communicate machine state and subscribe from many applications.

But like all technologies, it has limitations. If some of the examples provided in this article resonate with your architecture, it doesn't mean your architecture is inherently wrong, but you may want to consider the pros and cons of the approach.

At HighByte, we're protocol and vendor agnostic. We believe that although MQTT is a key enabler for real-time machine data in the UNS, the UNS is broader than a single technology and encompasses all data patterns found in the factory. But more on that in a future article.

About the Author

Aron Semle is the Chief Technology Officer of HighByte, focused on guiding the company's technology and product strategy through product development, technical evangelism, and supporting customer success. His areas of responsibility include research and development and product development and validation.

Aron has more than 15 years of experience in industrial technology. He previously worked at Kepware and PTC from 2008 until 2018 in a variety of roles including software engineer, product manager, R&D lead, and director of solutions management, helping to shape the company's strategy in the manufacturing operations market.

Aron has a bachelor's degree in computer engineering from the University of Maine, Orono.

About HighByte

HighByte is an industrial software company founded in 2018 in Portland, Maine USA. The company builds solutions that address the data architecture and integration challenges faced by manufacturers and industrial companies as they digitally transform. HighByte Intelligence Hub, the company's proven Industrial DataOps software, provides modeled, ready-to-use data to the Cloud using a codeless interface to speed integration time and accelerate analytics. The Intelligence Hub has been deployed in more than a dozen countries by the world's most innovative companies spanning a wide range of vertical markets, including automotive, energy, food and beverage, life sciences, and mining and metals. Learn more at highbyte.com.