Messaging-Technologien für das IIoT

Die Qual der Wahl?

Es gibt viele unterschiedliche Messaging-Protokolle, die man für vernetzte industrielle IoT-Applikationen einsetzen könnte. Welches sollten Entwickler nutzen, wenn sie hochperformante Applikationen umsetzen wollen, bei denen viele Embedded Computer Systeme miteinander und auch mit Clouds in Echtzeit kommunizieren müssen?

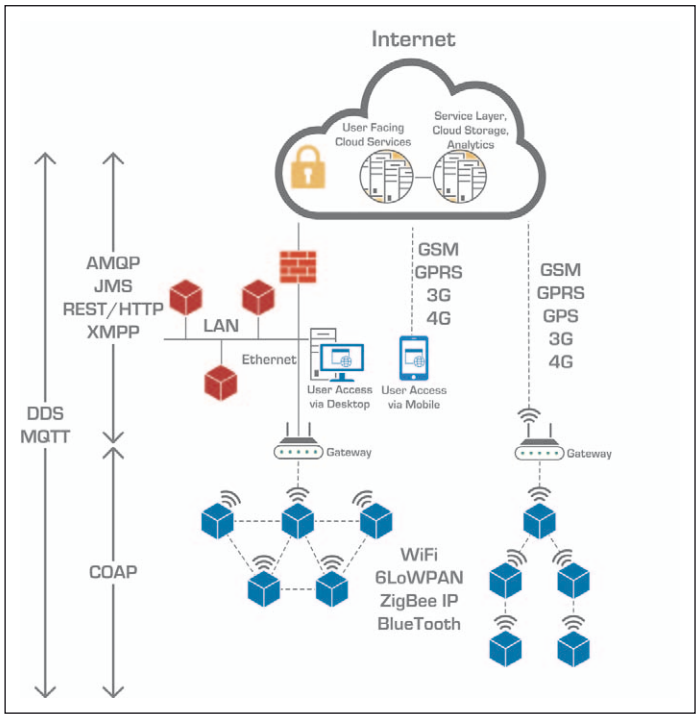


Bild 1: Reichweite der Messaging-Protokolle über unterschiedliche Infrastrukturen hinweg. Nur DDS und MQTT kann überall eingesetzt werden

Vernetzte Applikationen mit vielen verteilten Embedded Computer Systemen gibt es viele. Nicht alle müssen auch dezentral untereinander kommunizieren können. Auch haben nicht alle Echtzeitanforderungen oder hohe Performanceansprüche. Doch nahezu jede vernetzte industrielle IoT-Applikation hat genau diese Anforderung: Unterschiedlichste Maschinen- und Anlagenbestandteile müssen stets miteinander in Echtzeit kommunizieren, um zum Beispiel interaktive modulare (Multivendor-)anlagen umsetzen zu können. Ähnlich ist es auch in Windparks oder in Microgrids, wo unterschiedliche Energiequellen und Verbraucher in Echtzeit synchronisiert werden müssen. Der Schienenverkehr muss für Smart Railway Applikationen vernetzte, durchgängig echtzeitfähige Infrastrukturen und Züge bekommen. Gleiches gilt für smartes Straßenverkehrsmanagement, smartes Lighting und smarte Cities sowie auch autonome Roboter und Fahrzeuge aller Art.

Wer für solche vernetzten IoT-Applikationen das richtige Messaging-Protokoll aussuchen muss, der steht vor einem Problem. Es gibt zwar viele Protokolle und jedes Protokoll hat seine eigenen Vorteile, die in der Architektur oder in der Art des Messaging begründet liegen. Doch nur wenige sind wirklich dafür prädestiniert, als universelles Echtzeit-Protokoll eingesetzt zu werden, wie ein Überblick über die sieben wichtigsten industrietauglichen Protokolle AMQP und JMS, MQTT, REST, CoAP, XMPP sowie DDS zeigt.

Die wichtigsten Protokolle im Überblick

Die Protokolle AMQP und JMS wurden für schnelle und zuverlässige Business-Transaktionen entwickelt, also beispielsweise um den elektronischen Zahlungsverkehr zu regeln oder Börsenhandel zu betreiben. JMS fokussiert dabei auf Java-zentrierte Systeme. Es gibt aber auch einige Hersteller, die eigene C und C++ JMS API Mappings entwickelt haben, die mit dem JMS Broker genutzt werden können. Als API Standard kann JMS aber keine Interoperabilität garantieren. Hersteller und Nutzer können also unterschiedliche JMS Implementierungen nicht bedarfsgerecht kombinieren. Applikationen sind folglich in sich geschlossen und damit proprietär, was für viele neue IoT-Applikationen nicht gewünscht ist, denn was hilft es – einfach gesagt – wenn das Fenster meldet ‚Ich bin offen', die Heizung aber nicht versteht, dass sie sich dann abschalten soll.

MQTT: Zusammenarbeit muss organisiert werden

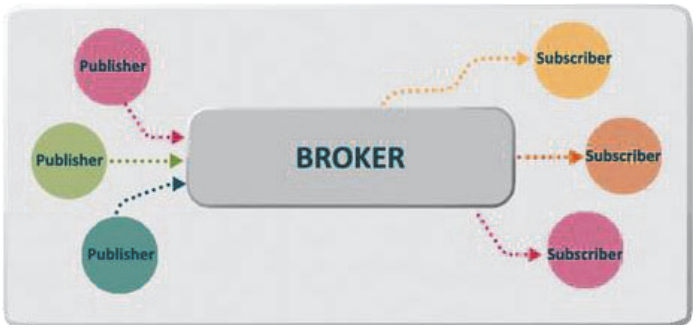
MQTT ist weniger komplex und für eine einfache und schlanke Datenerfassungslösung für Devices aller Art gemacht. Allerdrings kann auch hier nur teilweise eine Interoperabilität zwischen einem MQTT Publisher und Abonnenten garantiert werden. Nachrichten können zwar problemlos zwischen verschiedenen MQTT Implementierungen ausgetauscht werden. Aber die Nachricht kann nicht verstanden werden, wenn es zwischen Sender und Empfänger keine Vereinbarung über das Schema des Nachrichteninhalts gibt. Das setzt das aktive Aufsetzen einer koordinierten Zusammenarbeit zwischen den einzelnen Teilnehmern voraus, was bei komplexen Installationen ein kompliziertes, fehleranfälliges und teures Unterfangen werden kann.

REST: Einfach aber rechenintensiv

REST bietet mit Request & Reply zwar eine einfache Art der Client-Server Kommunikation, die man für Systeme nutzen kann, die über das Internet kommunizieren müssen. REST unterstützt aber nicht den lose gekoppelten, asynchronen Publish & Subscribe Datenaustausch. Das Modell der Zustandslosigkeit von HTTP kann zwar Server-Designs vereinfachen. Das hat

Autor:

Daniel Piper ist Marketing Manager bei ADLINK Technology



Bildt 2: Eine Architektur mit Brokern verbraucht Ressourcen produziert Latenzen. Nicht immer lässt sich das jedoch vermeiden. Beispielsweise in Richtung Feld oder auch in Richtung Enterprise Level. Zwischen den verteilten Devices ist es jedoch sinnvoll, auf ein einheitliches Protokoll wie beispielsweise DDS zu setzen

aber den Nachteil, dass es notwendig sein kann, die fehlende Zustandsbeschreibung als zusätzliche Information in jede Anfrage integrieren zu müssen. Und diese muss sodann auch zusätzlich vom Server interpretiert werden. Dies kann sich sehr ineffizient auf die Verarbeitungszeiten von Anfragen auswirken und wertvolle Ressourcen verbrauchen (wie zum Beispiel die Anzahl der möglichen TCP/IP-Verbindungen). Damit eignet sich auch REST nicht für große Installationen.

CoAP: Latenzen unvermeidbar

CoAP wurde entwickelt, um die Konnektivität von einfachen elektronischen Low-Power-Devices (z. B. drahtlosen Sensoren) mit Internet-basierten Systemen zu unterstützen. Es eignet sich sehr gut für die Datenerfassung in Systemen. Voraussetzung ist aber, dass weder eine sehr hohe Leistung noch ein Echtzeit-Daten-Sharing oder eine Echtzeit-Gerätesteuerung erforderlich sind. In vielen Fällen werden Daten für eine nachträgliche ‚Offline'-Verarbeitung gesammelt. Ein CoAP-Gerät wird dabei mit einem Cloud-basierten System über einen HTTP-Proxy verbunden, der ein Standard CoAP-HTTP-Mapping verwendet. Der Einsatz eines solchen Proxys oder einer Brücke erhöht dabei sowohl den Kommunikationsaufwand als auch die Nachrichtenlatenz.

XMPP: Zu komplex für Low Power

Das XMPP Protokoll basiert auf der Extensible Markup Language (XML) und wurde ursprünglich für das Instant Messaging (IM) und die Online-Anwesenheitserkennung entwickelt. Die Kernstandards dieses Protokolls bilden ein Framework für den Aufbau von Messaging-Anwendungen wie Mehrparteien-Chats, Sprach- und Videoanrufe, Kollaboration, Lightweight Middleware, Content-Syndication und generalisiertes Routing von XML-Daten. Obwohl XMPP Sicherheitsfunktionen wie Authentifizierung und die Verschlüsselung von Nachrichten unterstützt, bietet es keine Unterstützung für die Quality-of-Service Anforderungen, wie sie typischerweise in Industrial Internet-Systemen benötigt werden. Das XML-Parsing erfordert zusätzlichen Verarbeitungsaufwand und ein XML-Parser addiert zusätzlichen Speicherbedarf, sodass sich XMPP nicht für den Einsatz in Low-Power Embedded-Systemen eignet.

DDS: Datenzentrisches Publish & Subscribe

Das DDS-Protokoll ist eine datenzentrische Publish & Subscribe Technologie, die für missions- und geschäftskritische Applikationen wie den Finanzhandel, die Flugsicherung, das Smart Grid Mangement und andere Big-Data Applikationen entwickelt wurde. DDS unterstützt dynamische Discovery-Prozesse, benötigt also anders als AMQP, MQTT oder JMS keinen einen Broker/Makler.

DDS wird zudem auch zunehmend als Schlüsselprotokoll für das interoperable Messaging zwischen Echtzeit-Devicenetzwerken und cloud-basierten Datencentern eingesetzt. Implementierungen von DDS können zudem auch innerhalb der Devices (Intra-Nodal) einen hochperformanten Datenaustausch ermöglichen. Kernstück ist der datenzentrische Publish & Subscribe Layer, der über ein kohärentes Set an standardisierten Profilen einen Informationsaustausch in Echtzeit ermöglicht. DDS ist unabhängig von Programmiersprachen und Betriebssystemen. Es gibt standardisiert APIs für zahlreiche Programmiersprachen wie Ada, C, C++, C#, Java, Scala, Lua und Ruby die auch eine Portierung zwischen den Implementierungen unterschiedlicher Hersteller sicherstellen.

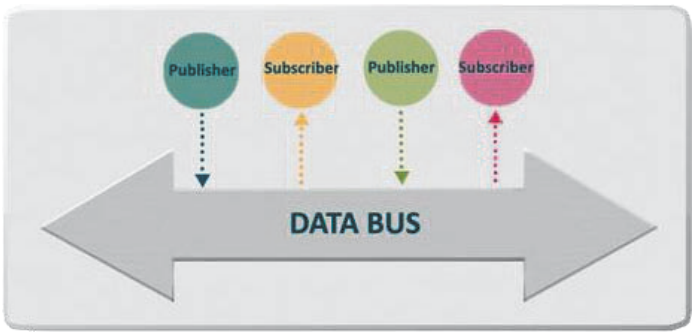


Bild3: Der DDS Datenbus kann unterschiedliche Publisher und Subscriber verwalten (tunneln) und je nach Topic bedarfsgerecht verbinden

Sicherheit gewährleisten

Die Sicherheit eines Systems, das potenziell mit vielen tausenden Geräten fehlertolerant und sicher kommunizieren muss, ist von großer Bedeutung. Die meisten vorgestellten Messaging-Technologien widmen sich dieser Fragestellung jedoch nur mittelbar. Führende Anbieter ergänzen ihre Implementierungen beispielsweise in der Regel um proprietäre Lösungen auf Basis von erprobten Third-Party-Sicherheitstechnologien wie SSL oder TLS. So binden AMQP und XMPP die Schnittstelle SASL zur Nachrichten-Authentifizierung an und die kürzlich (Juni 2016) verabschiedete OMG DDS Sicherheitsspezifikation liefert nun auch einen umfassenden Rahmen für die Standardisierung der Sicherheit von DDS-basierten Systemen.

Offen bleiben

Es sollte auch noch erwähnt werden, dass es in einigen Fällen – zum Beispiel aus Legacy-Gründen – sinnvoll sein kann, ein System zu entwickeln, das mehr als nur eine Messaging-Technologie innerhalb der gleichen Architektur verwendet. In diesem Fall werden zum Datenaustausch individuelle Vermittlungsschemata und Bridges zwischen Clients mit unterschiedlichen Protokollen erforderlich. Dies ist zwar nicht die optimale Lösung. Es ist aber ein gängiges Szenario. Besonders wenn man IoT-Systeme entwickelt, die sowohl neue und Legacy-Devices integrieren müssen.

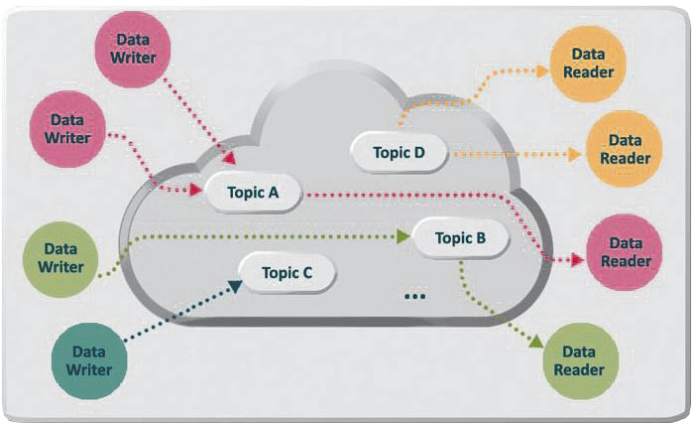


Bild 4: Die Adlink SEMA Cloud Lösung unterstützt MQTT und wird in Kürze auch DDS unterstützen

	DDS	MQTT	AMQP	JMS	REST/HTPP	CoAP	XMPP
Abstraction	Pub/Sub	Pub/Sub	P2P or Pub/Sub	P2P or Pub/Sub	Request/Reply	Request/Reply	P2P or Pub/Sub (based on draft spec/ XEP-0060)
Implementation Architecture	Global Data Space	Brokered (most common)	Brokered (most common)	Brokered (most common)	Client-Server	Client-Server	XMPP Server (broker)
User configurable QoS	22	3	3	3	None	Confirmable or non-confirmable messages	None
Interoperability	Yes	Partial	Yes	No	Yes	Yes	Yes
Hard Real-time	Yes	No	No	No	No	No	No
Transports	UDP by default but other transports such as TCP can also be used	TCP	TCP	Not specified but typically TCP	TCP	UDP	TCP
Subscription Control	Partitions, Topics with message filtering	Topics with hierarchical matching	Exchanges, Queues and Bindings in v0.9.1 standard, Queues and message filtering in v1.0 standard	Topics and Queues with message filtering	N/A	Provides support for Multicast addressing	Nodes which are analogous to a Topic defined In draft spec XEP-0060
Data Serialization	CDR	Undefined	AMQP type system or user defined	Undefined	No	Configurable	XML
Standards	OMG's RTPS and DDSI standards	Proposed OASIS MQTT standard M	OASIS AMQP	JCP JMS standard	Is an architectural style rather than a standard	Proposed IETF CoAP standard	XMPP Standards Foundation
Encoding	Binary	Binary	Binary	Binary	Plain Text, also supports various types of content encoding e.g. ZIP, compress, deflate	Binary	Plain Text
Licensing Model	Open Source & Commercially Licensed	Open Source & Commercially Licensed	Open Source & Commercially Licensed	Open Source & Commercially Licensed	HTTP available for free on most platforms	Open Source & Commercially Licensed	Open Source & Commercially Licensed
Dynamic Discovery	Yes	No	No	No	No	Yes	Yes
Mobile devices (Android, iOS)	Yes	Yes	Yes	Dependent on JAVA capabilities of the OS	Yes	Via HTTP proxy	Yes
6LoWPAN devices	Yes	Yes	Implementation specific	Implementation specific	Yes	Yes	No
Multi-phase Transactions	No	No	Yes	Yes	No	No	No
Security	Vendor specific but typically based on SSL or TLS with proprietary access control	Simple Username/Password Authentication, SSL for data encryption	SASL authentication, TLS for data encryption	Vendor specific but typically based on SSL or TLS. Commonly used with JAAS API	Typically based on SSL or TLS	DTLS	TLS and SASL

Tabelle 1: Zusammenfassung der Schlüsselkriterien

Fazit

Alle vorgestellten Messaging-Technologien DDS, MQTT, AMQ P, JMS, REST, COAP und XMPP eignen sich, Geräte in einem verteilten Netzwerk zu verbinden. Allerdings eignet sich nicht jedes für alle Anwendungsbereiche und Einsatzszenarien. Dazu zählen unter anderem Faktoren wie die Inter- und Intra-Gerätekommunikation, die Geräte-zu-Cloud-Kommunikation sowie auch die Kommunikation zwischen den jeweiligen Datenzentren. Zudem sind selbstverständlich wichtige Anforderungen an die Embedded IT Systeme wie Performance, Servicequalität, Interoperabilität, Fehlertoleranz und Sicherheit zu berücksichtigen. Müssen in Applikationen viele Geräte miteinander kommunizieren und ist eine hohe Leistung und Echtzeit-Kommunikation erforderlich, dann hat DDS deutliche Vorteile gegenüber den anderen Messaging-Technologien. Zwar braucht nicht jede Applikation alle Funktionen, die DDS zu bieten hat. Als Universalprotokoll wäre DDS aber prädestiniert. Warum es also nicht auch für Applikationen nutzen, die nicht so hohe Anforderungen stellen? Eine Standardisierung der Kommunikation in IoT-Umfeld können wir nämlich durchaus gebrauchen. Unterstützt Ihr Embedded-Computing Lieferant dieses Protokoll und auch wichtige Konvertierungen sowohl auf der Device- bzw. Feldebene als auch auf Enterprise-Level, haben Sie bereits die wichtigsten Voraussetzungen geschaffen, Ihre hochperformante Applikationen umsetzen zu können, bei denen viele Embedded-Computer-Systeme miteinander und auch mit Clouds in Echtzeit kommunizieren müssen.

■ *ADLINK Technology*
www.adlinktech.com