# Message Queues in Industrial IoT

Laksh Bhatia
Aalto University

***Abstract*—Message queues are tools used for inter-process communication. The communication can be between softwares, services or devices on the network.**

**This paper covers the aspects of various standards and protocol families that can be used to implement message queues. The following protocols are being discussed here: MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol), STOMP (Streaming Text Oriented Messaging Protocol) and ZMTP. It also describes the middlewares that implement the above mentioned protocols. The middlewares that are part of this paper are:Apache ActiveMQ, ZeroMQ, RabbitMQ.**

**The paper is finally concluded by giving a description of a typical Industrial IoT solution and which of the ActiveMQ,ZeroMQ and RabbitMQ would fit into which system.**

## I. Introduction

Message queues are software tools that are used for inter-process communication. [1] These tools are used to send and receive messages between one or many services or devices. Message queues are asynchronous mode of communication. As message queues are asynchronous mode of communication, the senders and receivers dont need to be active on the message queue at the same time.

Messaging Patters are protocols or methods that define how different parts of a message queue communicate with each other. It defines the kind of relations or roles, services or devices could play in a messaging system. Messaging patterns can be divided into four major types:

- Client / Server
- Publish / Subscribe
- Push / Pull
- Point to Point

## II. Messaging Patterns

### *A. Client / Server*

Client Server mode of messaging pattern is when a lot of clients i.e. devices need to be connected to a single server and request data from the server. In this kind of messaging pattern, the server has the possibility of letting a lot of clients connect to it. The clients then request data from the server and the server sends the data in return.

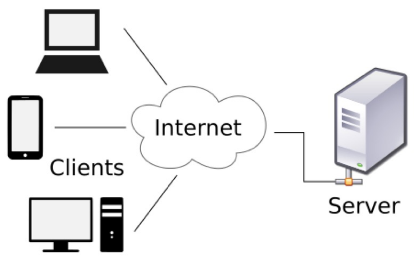

Fig. 1. Client Server Example

### *B. Publish / Subscribe*

A publish subscribe messaging pattern is similar to a notice board or a RSS feed kind of implementation. Multiple devices can subscribe to channels or topics and one of those devices can publish data to the topic or channel. This is a centralised kind of messaging pattern as all the devices need to be connected to a single device that act as a broker.

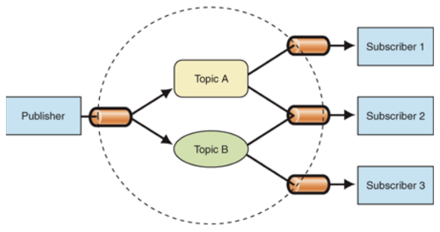

Fig. 2. Publish Subscribe Example

### *C. Push / Pull*

A push pull messaging pattern is defined as a messaging pattern when devices push data to the server and others can pull data from the server. Unlike the publish subscribe system where all devices that are subscribed to a topic get the data whereas in push/pull devices need to pull the data explicitly.

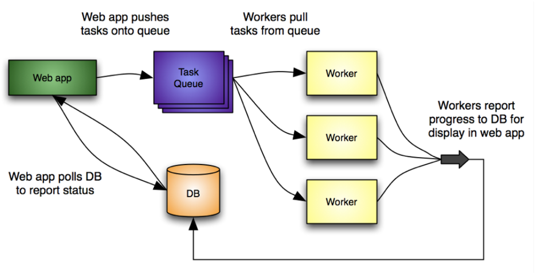
Web app pushes tasks onto queue
Workers pull tasks from queue
Web app
Task Queue
Worker
Worker
Worker
Workers report progress to DB for display in web app
Web app polls DB to report status
DB

Fig. 3. Push Pull Example

### D. Point to Point

A point to point messaging pattern is where the senders and the receivers both hold individual queues. The messages are sent directly from the sender to receiver. There are no brokers or central routing units in this system.

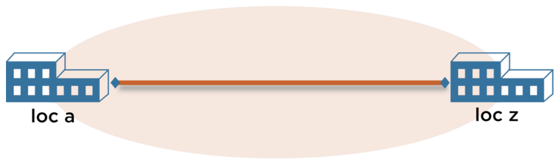

Fig. 4. Point to Point Example

## III. Standards and Protocol Families

### A. MQTT

MQTT (Message Queuing Telemetry Transport) [2] is a publish subscribe message queueing protocol. It was approved as an ISO standard in 2015. Its a very lightweight and flexible protocol. It only requries 2 byte of header information. MQTT is broker based architecture. The broker consists a list of all topics that are currently available. All the messages need to go through the broker. One of the most common brokers is Mosquitto. A typical MQTT system consists of a single or multiple brokers. The clients of this system either publish or subscribe to a topic or multiple topics. Whenever a client publishes a message to a topic, all the clients subscribe to it receive the message and can process it. This is illustrated in the figure below:

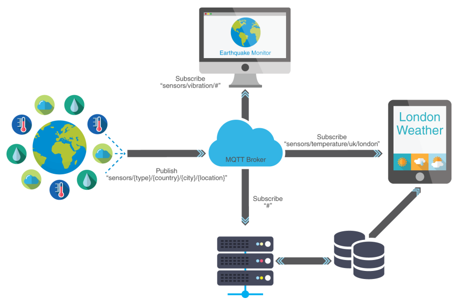
Earthquake Monitor
Subscribe "sensors/vibration/#"
London Weather
Subscribe "sensors/temperature/uk/london"
MQTT Broker
Publish "sensors/{type}/{country}/{city}/{location}"
Subscribe "#"

Fig. 5. MQTT Example

### B. AMQP

AMQP (Advanced Message Queuing Protocol) [3] also implements a topic based publish subscribe system. It has a range of features that it implements. It is a superset of MQTT. But unlike MQTT which has really low data overhead, the overhead of this system is fairly high. It assumes an underlying reliable protocol like TCP for transport which gives it a comparitive overhead to any well defined standards like FTP or SSH. One of the benefits that it has above the MQTT system is that it also gives the possibility of a point to point system. This system can be used to communicate between high level processes that require a lot of messages and need to talk?

### C. STOMP

STOMP (Streaming Text Oriented Messaging Protocol) [4] is a text based messaging queue protocol. It is a very simple protocol since it is text based. It can be run from the simplest of applications like Telnet. Since it is such a simple protocol there are no underlying security measures defined in this protocol.

### D. ZMTP

ZMTP (Zero Message Transport Protocol) [5] is a very flexible protocol. ZeroMQ is based on top of this protocol. It is hghly flexible and can implement a broker based and a decentralised solution.

## IV. Messaging Middleware

### A. Apache ActiveMQ

ActiveMQ [6] is a messaging middleware designed by Apache. It is a middleware built on top of the Java Messaging Service(JMS). It implements a lot of protocols. Some of them are AMQP, MQTT, STOMP, XMPP, OpenWire, etc. It is an enterprise level solution and hence is very well maintained. It also has a support for multiple clients, e.g. C,C++,Python. Since this messaging middleware supports both AMQP and MQTT, it provides the possibility to have both a broker based and a Point to point based architecture. Since it implements a central node architecture it could slow down the applications as there would be a limit to how many messages could the central broker handle. Like every message queue architecture this implements a similar system where there is a central broker that receive the messages and they are then sent over to the consumers of the messages.

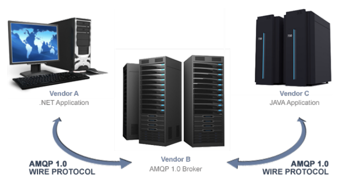

Fig. 6. Point to Point Example

### B. RabbitMQ

RabbitMQ [7] is a messaging middleware software. It is build in Erlang. Like ActiveMQ, it has support for AMQP, MQTT and STOMP. But it has been designed to be used mainly for AMQP. The support for MQTT and STOMP is through external plugins. Because it implements AMQP as a core it supports broker only architecture. In addition to the features that ActiveMQ provides, it also provides some additional services that make it more suitable for industires that generate a high amount of load. Some of the additional services that it implements are Routing, Load Balancing and support for Message Storing. RabbitMQ is implemented in a lot of major companies like Sony, Auth0, Wunderlist.
A typical RabbitMQ solution looks slightly different from a standard messaging middleware solution. Once a message

reaches the RabbitMQ central solution, it is sent to the exchange. The exchange then decides where to route this packet based on the rules set. The packet is then sent to a queue from where it is consumed by the clients. This scenario is described in the image below:

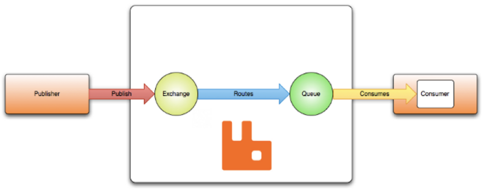

Fig. 7. Point to Point Example

### C. ZeroMQ

ZeroMQ [8] is a unique and flexible messaging middleware solution built on top of the ZMTP. It is written completely in C++. Clients and libraries in all other languages have already been written. This messaging middleware is designed for high throughput and low latency. Because of that it is highly suitable not only for a large volume of small messages but also for small number of large messages. It supports weather data load as well as it can support camera streaming load. The results for this can be found in [9].
ZeroMQ is such a flexible solution so it can support publish/subscribe, client/server, push/pull and point to point messaging systems. It can also support a broker like architecture which would make it similar to the previous two soultions. But besides the broker architecture it can be used to implement a decentralised solution such that a lot of brokers can be consumers and producers of the messages. Its a unique messaging system and is slowly being adopted in major corporations like Spotify, Microsoft.

## V. Industrial IoT

Industrial IoT is a broad field. Industries have had sensors for a long time that are constantly pulling data and collecting them. The problem with these sensors is that a lot of them arent connected to the internet or even to the local network. Thats why there was a lot of major man power required to collect this data. With the advent of IoT and cheap embedded devices, it is now possible to connect these sensors on the grid. It is now possible to gather data from them and analyse them to better understand the workings. But since industry is a big field, the problems faced are varied. Some industires require large number of sensor data and some require a small number of large sized data. They also face the additional problem that they are designed on different langagues and run different protocols.
To get some uniformity into all this chaos, the writer suggests the following pattern for how the messaging middlewares would fit in. Since industries have been around for a long time and an implementation of a protocol in the industry needs to last a long time, a commercial middleware like ActiveMQ should be implemented. ActiveMQ should be structured in the following way. All the low level sensors should communicate with a central node which would gather all the data. This data would then be put into a message queue run by AMQP. As and when the data is required by the client nodes, they would be consumed and removed from the queues.

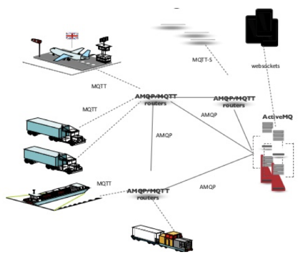

Fig. 8. MQTT Example

If the industry would not have any problem with dynamically changing the solution and implementing its own protocol it would be better for them to implement ZeroMQ. ZeroMQ is more flexible in terms of the deployment and language requirements. It also has a higher throughput that AMQP and MQTT in terms of the data it can handle.

## VI. Conclusion

This paper gave a brief description of some of the messaging patterns, middlewares and standards that are currently being deployed. It also shows the merits and demerits of ZeroMQ, ActiveMQ and RabbitMQ. The paper described an Industrial scenario and concluded which solution fits which use case. Since this is a broad field, there is no one fit for all solution.

## References

[1] CloudAMQP. (2014) What is message queuing? [Online]. Available: https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html
[2] MQTT. (2014) Mqtt specification. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
[3] AMQP. Amqp: Protocol specification. [Online]. Available: https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf
[4] STOMP. Stomp protocol specification. [Online]. Available: https://stomp.github.io/stomp-specification-1.2.html
[5] Zmtp:protocol of things. [Online]. Available: http://zmtp.org/page:read-the-docs
[6] ActiveMQ. Activemq: Design documents. [Online]. Available: http://activemq.apache.org/design-documents.html
[7] RabbitMQ. Rabbitmq: Documentation. [Online]. Available: https://www.rabbitmq.com/documentation.html
[8] ZeroMQ. (2011) Zeromq: Theoretical foundation. [Online]. Available: http://250bpm.com/concepts
[9] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting iot platform requirements with open pub/sub solutions," *Annals of Telecommunications*, pp. 1–12, 2016. [Online]. Available: http://dx.doi.org/10.1007/s12243-016-0537-4