Development of MQTT-Channel Access Bridge

Jiro Fujita – Creighton University/STAR Experiment

STAR Experiment

- Solenoidal Tracker At RHIC
- Relativistic Heavy Ion Collider
- Brookhaven National Laboratory, Upton, NY, USA
- EPICS is used for the control for the most subsystems from the beginning
  - Mostly EPICS 3.14, but some in older version…
- Control & Monitor roughly 40,000 operating parameters

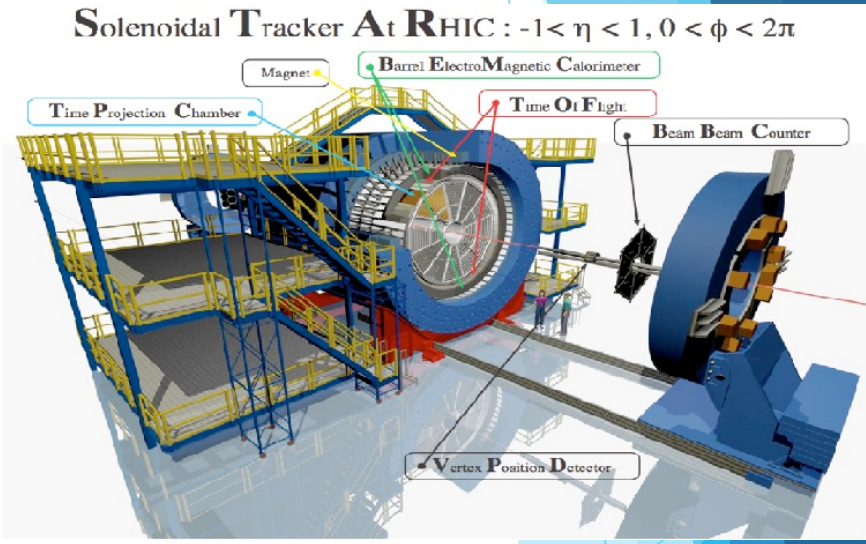


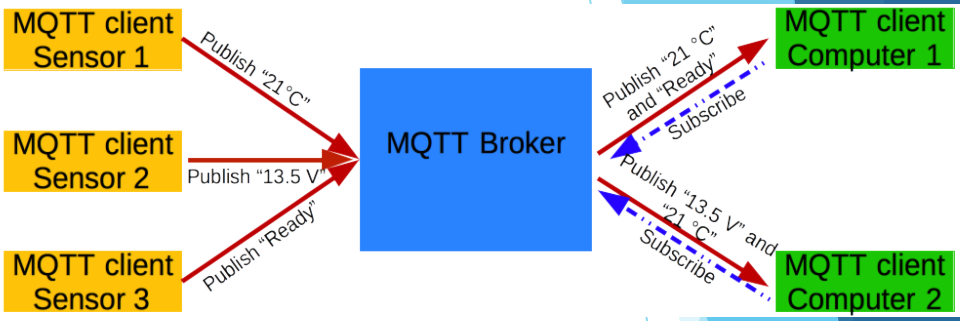
Image from RHIP Group at UT Austin: http://www.rhip.utexas.edu/

MQTT: Message Queue Telemetry Transport

- Originally developed by IBM and Eurotech in 1999
  - IBM & Eurotech donated MQTT to Eclipse project in 2011
  - MQTT v3.1.1 is ISO standard as of 2016 (ISO/IEC PRF 20922)
- Runs on top of TCP/IP (as well as UDP and ZigBee)
- Relatively simple and easy to write/work with
- Lightweight & low overhead
- Quality of Service
- Nearly Real-Time
- Widely used by Internet of Things (IoT) and other places
- Very different from Channel Access

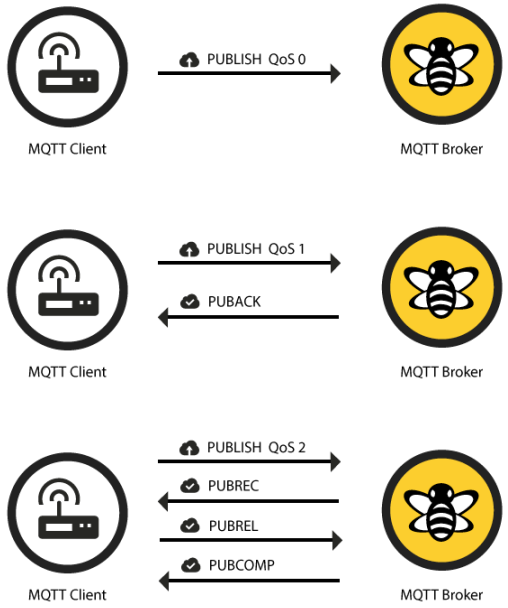
MQTT Concept

- Message Broker
  - Acts as a central communication point
  - Can be authenticated
- Clients
  - Publish
    - Devices publish specific information/data (topic)
  - Subscribe
    - Devices could also subscribe to specific information/data (topic)
  - Topic
    - Routing information to the broker (e.g. “Status”, “Voltage”, “TPC”, “Beamline1”)
  - Message
    - The “data” that clients publish/subscribes
    - Only in ASCII format



MQTT Quality of Service

- MQTT has concept of Quality of Service (QoS) built-in
  - QoS0
    - It only guarantees a best effort of delivery
    - Essentially, no checking
  - QoS1
    - Guarantees at least delivered once
    - It could be delivered more than once
  - QoS2
    - Guarantees delivered once and once only
    - Safest, but slowest, as requires extra confirmation



Images from HiveMQ: MQTT Essentials
http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels

Motivation & Requirements

- STAR had adapted DAQ/Offline/Slow Control integration based on MQTT
- Current Slow Control is based mostly on EPICS since the beginning of the Experiment
  - Somebody had to write something to bridge EPICS Channel Access and MQTT in both direction
- EPICS stays as it is for the existing control systems

Motivation & Requirements (part 2)

- General concept of what has been proposed at STAR Experiment
- Slow Control resides in STAR protected network in this diagram
- Send/Receive MQTT messages in JSON
- Receive/Send Slow Control Data in Channel Access

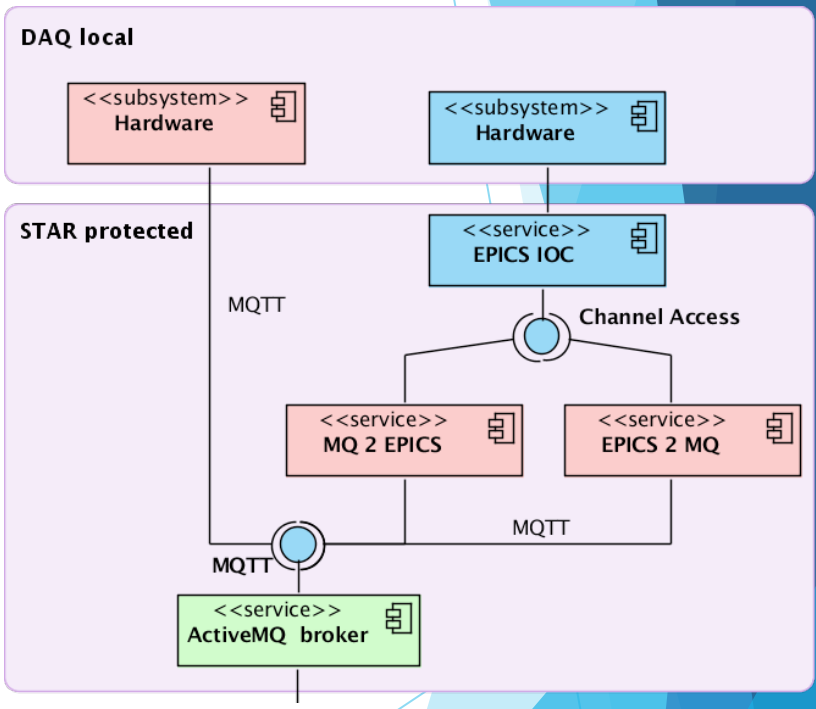


Image from "Bridging EPICS and High-Level Services at STAR"
https://drupal.star.bnl.gov/STAR/comp/db/development/epics_dcs

Tools used & rational

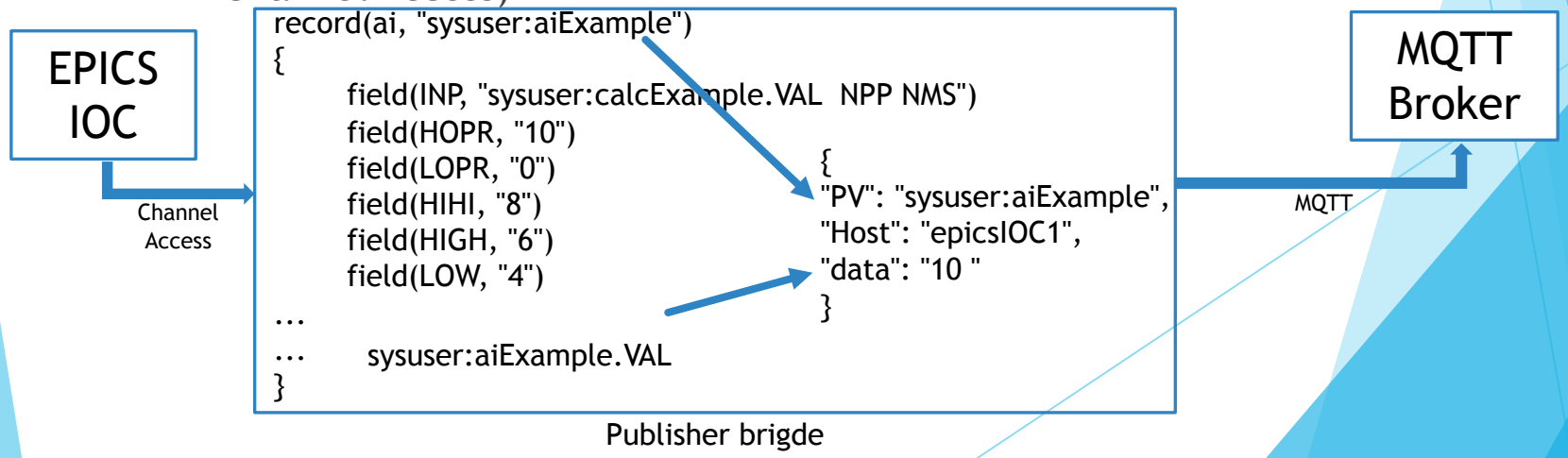
- C (or C++)
  - This was a request from the DAQ expert to the control group
- Paho Library
  - Appears to support C/C++ fairly well
  - http://www.eclipse.org/paho
- JSON parser library
  - The message content is in JSON format
  - https://github.com/json-c/json-c
- Apache ActiveMQ Apollo for the message broker
  - The Offline/DAQ integration has already been using Apollo as the message broker
  - https://activemq.apache.org/apollo
- MQTT.fx
  - To check MQTT status easily from the office computer
  - http://mqttfx.org

The first prototype…

- Written in about 2 weeks, including setting up the broker and MQTT test programs
  - No prior knowledge/experience of MQTT
- Fairly easy to write a program using MQTT
- Two different components
  - Publisher — Channel Access to MQTT
  - Subscriber — MQTT to Channel Access
- Single Topic (EPICS_CA) is used for now

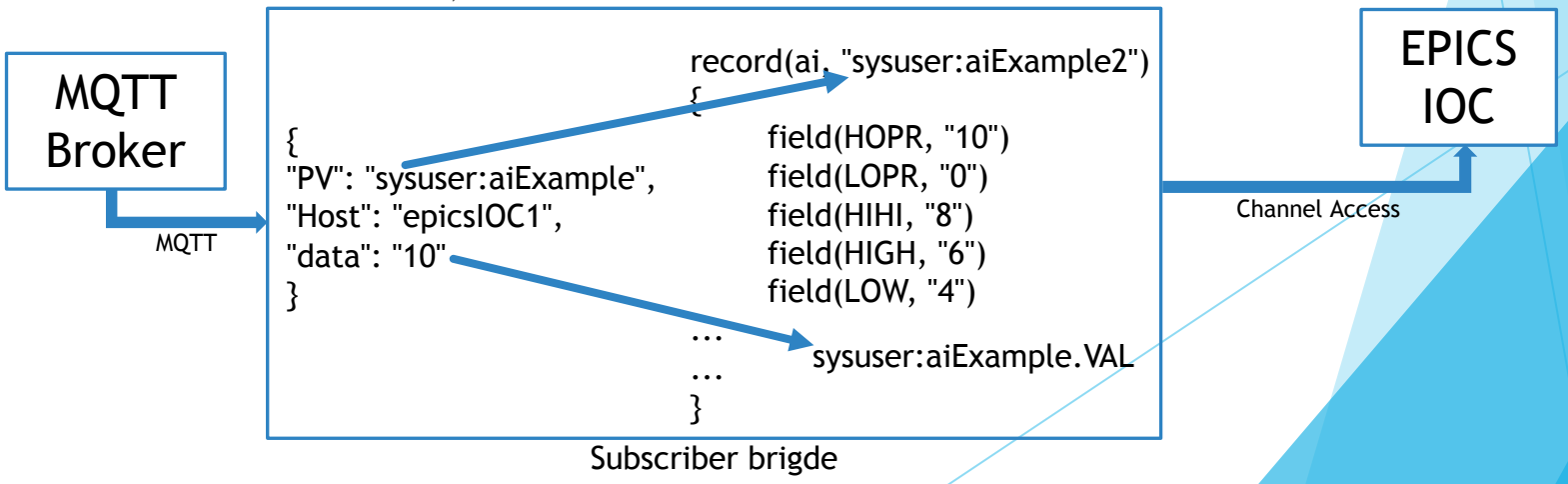
Publisher Component

- Publishes Channel Access data to MQTT broker
- Written in C using Portable Channel Access library
- Sends Channel Access data in MQTT in JSON
  - { "PV": "PVname", "Host": "Hostname", "data": "PV value"}
  - Host field is needed, as MQTT does not necessary know where the data really comes from (not really needed in Channel Access)



Subscriber Component

- Subscribes data from MQTT broker to EPICS IOC
- Written in C using Portable Channel Access library
- Receives MQTT in JSON, broadcasts in Channel Access
  - { "PV": "PVname", "Host": "Hostname", "data": "PV value"}
  - Host field is needed, as MQTT does not necessary know where the data really comes from (not really needed in Channel Access)



Performance Testing

- Testing was done in sending thousands of MQTT data and/or Channel Access data per seconds
  - Several computers were involved (all in the local network, some in different subnet)
    - EPICS IOC host computers
      - More than one IOC computers were used for testing
    - Computer running MQTT bridge programs
    - MQTT data sender/receiver computer running simple MQTT publisher/subscriber program written in Node.js
    - MQTT broker (Apache Apollo) computer in different subnet
    - MQTT monitoring computer (aka my office computer)
      - not really needed, but convenient to have one
- It appears to withstand both directions at least up to about 1000 data/second or so
  - Not very quantitative measure, as I had no idea how to quantify the number easily, as there are many different factors involved (computer performance, network speed, etc)

Future Plan

- Rewrite with C++ using CAFE from PSI/SLS
  - Possibly easier to write a code in C++ using CAFE than portable Channel Access in C
- Fuse publisher and subscriber into one program
- Offline/DAQ group wants use MQTT to view the Slow Control status via a web browser (and possibly even control in some cases)
- Yet to be tested for real system…
- Code is available upon request

Acknowledgement

- US Department of Energy Office of Science

- STAR Collaboration

- Creighton University College of Arts & Science

Creighton
UNIVERSITY
College of Arts and Sciences