Communication Patterns for the IoT

**Telemetry:** Measuring from device
- data flows asynchronously from device to server

```
Server -> asynchronous data flow -> Device
```

**Notification:** Sending data to device
- data flows asynchronously from server to device

```
Server -> asynchronous data flow -> Device
```

**Inquiry:** Device requests info from server
- request-response (device making request)

```
Server <- request -> Device
           ^ response
```

**Command:** Server issues command to device
- request-response (device making request)

```
Server <- request -> Device
           ^ response
```
Quality Requirements

- Suited for constrained IP networks & devices
  - limited packet size
  - high degree of packet loss
    - devices continuously powered down & wake up
    - in context of energy management
  - Limited power, RAM & processing power
- Do NOT maintain sessions to devices
  - decouple via
    - sessionless protocols
    - intermediaries (queues, topics, ... )
Protocols for the IoT

- **CoAP**
  - REST/UDP
  - simple reliable/unreliable messaging

- **MQTT**
  - publish/subscribe based messaging
  - suited for general telemetry (sensing)
  - limited suitability for high-rate data
  - separate QOS (Quality of Service) requirement

- **AMQP**
  - point-to-point based messaging
  - flow-control for high-rate data
  - separate QOS (Quality of Service) requirement

- **DDS**
  - decentralized peer-to-peer sessionless comms
CoAP - The Constrained Application Protocol

- Application layer protocol for resource-constrained Internet devices
  - small low power sensors & actuators
    - e.g. Internet-based Wireless Sensor Networks (WSN) nodes.
    - switches, valves, motors, ...
  - efficient remote monitoring & control
    - over Internet
Protocols for the Internet of Things (IOT)
The Constrained Application Protocol (CoAP)

What is CoAP?

Definition
The Constrained Application Protocol (CoAP) is an efficient REST protocol over low-level connectionless transport protocol – typically UDP.

- Developed by The CoRE group (Constrained RESTful Environments)
- Standardized by Internet Engineering Task Force (IETF)
Features of CoAP (1/2)

- Efficient
  - low header overhead
  - low parsing complexity

- Support for REST operations
  - create, get, update, delete

- Asynchronous messaging

- Support for UDP mandatory
  - much simpler & more efficient than TCP
  - optionally support for other transport protocols
    - SMS, TCP, ...

- URI & content (MIME) type support

- Event subscription and notification
  - one device observe resourced on another
    - receive state change events

- Simple mapping to HTTP-REST
  - mapping independent of application
    - CoAP resources accessible from HTTP-REST
Features of CoAP (2/2)

- Simple *proxy & caching capabilities* based on `maxAge`
  - sensor in sleep mode can respond with last-see value
    - without waking up device
- *Non-reliable multi-cast messaging*
  - to send msg to group of devices
- Simple *reliability support*
  - with or without ACK
- *Security*
  - Support for DLTS (Datagram Transport Layer Security)
- Support for *resource discovery*
  - uses *CoRE Link* format providing web servers to describe hosted resources
    - their attributes
    - their services
Protocols for the Internet of Things (IOT)
The Constrained Application Protocol (COAP)
CoAP layers

CoAP Layers

- **Application Layer**
- **CoAP Request/Response Layer**
- **CoAP Messages Layer**
- **Transport Layer (UDP, SMS, TCP, ...)**
Messages

- CoAP has two layers
  - Message layer
    - basis for reliable/non-reliable messaging
  - Request/response layer
    - on-top of message layer
    - request/response over reliable/unreliable messaging
Message-Layer Model

- Message Layer supports 4 types message:
  - **CON**
    - confirmable messages
  - **NON**
    - non-confirmable messages
  - **ACK**
    - acknowledged messages
  - **RST**
    - reset messages
CON Messages (1/2)

- Supports reliable messaging
  - Message resent until either
    - **ACK** received with same id
    - **MAX_RETRANSMIT** reached.
  - Recipient can reject confirmable message
    - by sending **RST**
Protocols for the Internet of Things (IOT)
The Constrained Application Protocol (CoAP)
CoAP messages

**CON Messages (1/2)**

![Diagram showing CoAP message exchange]

1. **Peer-1** sends **CON[0x2345]**
2. **Peer-2** sends **CON[0x2345]**
3. **Peer-1** sends **CON[0x2345]**
4. **Peer-2** sends **ALT**
5. **Peer-1** sends **[ACK rcvd within timeout]**
6. **Peer-2** sends **ACK[0x2345]**
7. **Peer-1** sends **[RST rcvd within timeout]**
8. **Peer-2** sends **RST**
9. **Peer-1** sends **[ACK_TIMOUT reached]**
NON Messages

- Unreliable message transport
  - messages not ACKnowledged

Peer-1 \(\rightarrow\) NON[0x2345] \(\rightarrow\) Peer-2
Request-Response Layer

- On top of *message layer*
  - decoupling synchronous/asynchronous messages
    - from reliable/unreliable messaging
- 3 types of request/response messaging
  1. *Piggy-backed*
    - Response with ACK
  2. *Separate response*
  3. *Non-confirmable request/response*
Piggy-Backed Request/Response

- Reliable & efficient
Separate Response Messaging

- Reliable but less efficient
Non-Confirmable Request/Response Messaging

- Non-reliable & efficient

Client

NON [0x4d45]
Get/temperature
(Token 0x21)

Server

NON [0x4d45]
2.05 Content
(Token 0x21)
  "20.1°C"
Protocols for the Internet of Things (IOT)

The Constrained Application Protocol (CoAP)

CoAP messages

Registering as Resource Observer

Telemetry: CoAP

Device (server)

CON GET /<resource> Observe: 0 Token: 0xAB

ACK 2.05 Observe: 30 Token: 0xAB <resource>...

CON 2.05 Observe: 31 Token: 0xAB <resource>...

ACK Token: 0xAB

ACK Token: 0xAB

CON 2.05 Observe: 32 Token: 0xAB <resource>...

ACK Token: 0xAB

RST Token: 0xAB

ACK Token: 0xAB

System (client)
CoAP Message Structure

- 4-byte header
  - CoAP version no
  - Msg type: 0:CON, 1:NON, 2:ACK, 3:RST
  - Code: additional typing info for msg
- Variable-length token, options & payload
MQTT

- Light-weight publish-subscribe protocol
  - MQTT/TCP/IP
- Developed by IBM
  - from IBM-MQ product suite
- Now Open Standard
  - managed by OASIS (www.oasis-open.org)
    - Advancing open standards for the information society
- Requires physical channel implementation (message broker)
  - maintains & distributes messages
  - topic subscriptions
Publish-Subscribe Channels

- Multiple consumers may register with channel/topic
  - receive msgs published on channel/topic.
- Messages consumed by all consumers subscribed to channel when msg sent.
- **Durable subscribers**
  - Receive all msgs published during subscription period.
  - Also those sent whilst temporarily disconnected.
Benefits of Publish-Subscribe Channels

- **Information distribution channel**
  - Provides infrastructure to have
    - multiple information sources publish data
    - to multiple information consumers

- **Decoupling**
  - *Party decoupling*
    - publisher & subscriber do not need to know who the other party is.
  - *Location decoupling*
    - publisher & subscriber do not need to know where the other party is.
  - *Time decoupling*
    - publisher need not know when subscriber processes info
    - subscriber need not know when publisher publishes info
  - *Synchronization decoupling*
    - neither publisher nor subscriber process blocked.

- **Scalability**
  - load can be *spread across time*
Message Filtering

- Subscribers may only want to receive subset of messages published on channel.
  - Channels commonly support *message filtering*.
    - *Channel (broker) only feeds msgs to subscriber which pass filter*.

- **Filter types**
  - **Header filters**
    - Filtering based on information in message headers.
    - e.g. priority, subject, message encoding, encryption strategy, . . .
  - **Content filters**
    - Subscribers receive only messages for which content passes filter.
    - e.g. content of a particular type (including specializations of type)
MQTT Implementations

- MQTT very well supported
  - many broker implementations
    - Eclipse Mosquito, mqtt.js, ...
    - MyMQTT – Android implementation available on playStore
    - Many provide monitoring tools
    - e.g. MQTT.fx – JavaFX front-end, mqtt-spy – Java-8, ...

- Client bindings most programming languages
  - most O/S licensing
  - C, C++, Java, JavaScript, C#, Objective-C, Python, Ruby, ...

- Widely supported across IoT technology stacks & frameworks
  - Eclipse IoT stack
  - IoTivity, ...
Quality Of Service (QOS)

- Message delivery guarantee via diff msg exchange patterns
  - QOS=0 → at most once
    - Msg sent once
    - No guarantee that received
    - Used when
      - Scalability important (low messaging overheads)
      - Lost messages not critical
  - QOS=1 → at least once
    - Msg receipt guaranteed.
    - No Guarantee that sender knows
    - ⇒ sender may resend
    - Used when
      - Msg must arrive
      - Recipient can handle duplicates
      - Overheads for QOS=2 not acceptable
  - QOS=2 → exactly once
    - Msg guaranteed to be received once.
      - or at least sender will know that not received
    - Used when
      - Critical that all msgs must be received & only once
      - Messaging overheads not a problem.
At-Most Once (QOS=0)

MQTT

Quality of service (QOS)

Sender

PUBLISH

Recipient

QOS=0

No receipt confirmation (may not have been received)
At Least Once (QOS=1)

Sender

QOS=1

PUBLISH

Recipient

Receipt conf, but no conf that receipt conf received.
=> Sender may resend msg even though received.
Protocols for the Internet of Things (IOT)

MQTT

Quality of service (QOS)

Exactly Once (QOS=2)

Sender

Recipient

Msg sent to sender, conf that recipient received msg, conf that sender received conf, conf that recipient received conf.
MQTT Connections

- **MQTT/TCP/IP**
  - client maintains connection (session)
- **Connect via CONNECT msg**
  - channel/broker responds with CONNACK

**MQTT Packet**: CONNECT

- **clientId**: myClientId (mandatory)
- **cleanSession**: true (all others optional)
- **username**: "tandiPretorius"
- **password**: "&^%!872##"
- **lastWillTopic**: "/sensors/monitoring"
- **lastWillQos**: 2
- **lastWillMessage**: "Connection to sensors topic lost"
- **keepAlive**: 120
Protocols for the Internet of Things (IOT)

MQTT

Publishing messages (information)

Publishing Messages/Information

MQTT Packet: PUBLISH
packetID: 5552761

topicName: "sensors/building23/level4/room4_23"

qos: 0

whether msg saved by broker for last known value

retainFlag: false

whether msg is duplicate because ackn not rcvd (only relevant for QOS>0)

payload: "temp:24.3"

dupFlag: false
Subscribing to a Topic

MQTT Packet: SUBSCRIBE

packetID: 2312345
qos: 1

topicName: "sensors/building23/level4/room4_22"
qos: 1

topicName: "sensors/building23/level4/room4_23"
...
Advanced Messaging Queueing Protocol (AMQP)

- Few asynchronous messaging standards
  - Java Messaging Service (JMS) perhaps most successful
  - Wire-level comms not standardized
    - Only API
    - Limits inter-connectivity
    - Also only supports Java

- Need for open, high performance & reliability protocol for trading
  - Performance critical
    - Fraction of milli-second can cost lots of money
  - Reliability critical
    - Possibly multi-billion dollar trades

- Proven extremely useful for IoT
  - efficiency & performance
  - reliability
  - security
  - interoperability
  - decoupling
  - manageability
Overview

Aims of AMQP

- Open standard
  - replace proprietary products
    - though none met requirements fully

- Standardize both
  - API
  - Wire-level messaging
    - so that truly integrable

- High degree of interoperability
  - between AMQP providers,
  - with existing messaging technologies,
  - with different programming languages & technologies.

- Extreme scalability
  - From very small to very Large (> 1GB)

- Extreme reliability
  - Also for timeous delivery
  - e.g. when late delivery has business impact (trading data)
What is AMQP?

**Definition**

Open (public) wire-level protocol standard for asynchronous messaging supporting high performance, scalability, reliability, interoperability.

- Royalty and patent-free
- Uses binary protocol
  - for higher data-rate/efficiency
- Support for transactions and quality of service
History and Development Method (1/2)

2003  John O’Hara from JPMorganChase

- Initiated development of new asynchronous messaging std.
- Contracted iMatrix to build reference implementation
  - *OpenAMQ* (open-source & free)
  - reference implementation & standard developed concurrently
- $\Rightarrow$ agile method with short feedback cycles, not waterfall

2006  AMQP working group formed

- manage AMQP as open, community-driven standard
- provide *strong governance* of standard
- Original members:
  - JPMorganChase, CSICO Systems, Envoy Technologies, iMatrix Corporation, IONA Technologies, Red Hat, TWIST Process Innovations, 29West
2008  AMQP version 0.9.1 published
   - broker based
   - widely implemented and used

2012  Version 1.0 published as *OASIS standard*
   - spec reduced to more flexible messaging spec
     - broker removed
   - Some bad feelings around lost investments
     - ⇒ 0MQ as light-weight alternative (not std)
Broker-Based Infrastructure (Version 0.9.1)

- Flexible routing pipes-and-filters based routing
  - decoupled via flexible/powerful bindings
- Supports intermediate message enrichment
- Wire-level protocol standard + messaging capabilities
  - Includes standardization of management commands
Changes with version 1.0

- Only wire-level protocol
- Leave API to different messaging frameworks
  - e.g. JMS providers
    - Standard Messaging API for Java applications
    - Supports queues & Topics
    - Durable subscribers
    - Binary, text and map-messages
    - Object messages (object serialization)

- OpenMama
  - Std C/C++/C# API to messaging middleware
  - Established standard in Linux community.
    - 2011 version 1.1 for Linux
    - 2015 version 2.3.3 for Linux & Windows

- QPid, QPid.Net, IBM MQ, Swift MQ, …
AMQP Architecture

- Purely standardized protocol
- can be put on top of different transport layer protocols

Diagram:

```
JMS Application         .Net Application
JMS API                Qpid.Net API
Qpid.JMS               Qpid Provider
AMQP
TLS/SSL
TCP
IP
```
AMQP Features

- Envelope wrapping
- Session multiplexing
  - with full-duplex asynchronous communication
- Streaming
- Support for message transfer guarantees.
Overview

Implements Envelope Wrapper

- Message wrapped by envelope
  - Envelope contains *message metadata*.
    - e.g. message priority, subject, ...
    - routing info
    - consumed by routing nodes
  - Message never touched (immutable)
    - can be encrypted
    - can be verified with hash
  - Enriched in envelope/header
Session multiplexing

- Single connection
  - carrying multiple sessions
- Each session has independent
  - full-duplex asynchronous comms
    - messages flow in both directions
  - independent message sequencing
  - independent flow control
    - limits on no of messages per unit time
- input & output channel
Multi-Frame Streaming

- Protocol submits sequential frames
  - Each frame contains session (channel) number
    - $\Rightarrow$ multiple sessions across same connection
  - Frames preceded by frame size
    - facilitates efficient frame reading
  - Support for very large messages
Message Transfer Guarantees

- **Includes**
  - message settlement & acknowledgement

- **Support for Quality Of Service (QOS)**
  - **Best effort:** send message and forget
    - equivalent to *at-most once*
  - **At least once:** Delivered until acknowledged
  - **Exactly once:** Bi-directional acknowledgement
    - Message processor only commits changes once confirmation of confirmation received.

- **Support for Transactions**
  - Local transaction for specific message exchange
  - Distributed transactions involving other resources
    - ⇒ 2-phase commit
Flexible Security

- TLS/SSL
  - Internet Engineering Task Force (IETF) standard
  - TLS $\rightarrow$ Transport Layer Security
  - SSL $\rightarrow$ Socket Layer Security
    - predecessor of TLS
  - X.509 certificates
    - Generate public and possibly private keys
    - sign with private key
    - decrypt with public key
  - AMQP: Encryption and Authentication via X.509 certificates.
- Can also use SASL (Simple Authentication & Security Layer), …
Pipes & Filters in AMQP version 1.0+

- Must be manually
  - Outside protocol
- Develop various nodes:
  - routing nodes
  - processing nodes
  - logging nodes
- Each node can forward to various others.
AMQP Implementations

- **Version 1.0**
  - Apache QPid Proton
    - High-performance light-weight
    - C, Java & JavaScript implementations
  - SwiftMQ
    - Small light-weight JMS compatible Java API
  - MuleSoft AnyPoint
    - For integration into Mule ESB (SOA)
  - Node AMQP
    - node.js implementation
  - IBM MQ Lite
  - Service Bus .Net
    - .Net AMQP client
    - connectivity to .Net service bus