CSC0056 Data Communication

Broker-Based Data Communication—Model and Design

Instructor: Chao Wang

Networked Cyber-Physical Systems Laboratory
Department of Computer Science and Information Engineering
National Taiwan Normal University

Sep. 13, 2024

NATIONAL TAIWAN NORMAL UNIVERSITY

Agenda

1. References
2. Introduction
3. The pub/sub model
4. The MQTT messaging protocol
5. Messaging server design and implementation
6. Recap

References

1. The MQTT official website: https://mqtt.org/
2. MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
3. MQTT Essentials: https://www.hivemq.com/mqtt-essentials/
4. Brandur Leach (2017). Designing robust and predictable APIs with idempotency. Link: https://stripe.com/blog/idempotency
5. William Yeh (2020). Idempotency Key：原理與實測. Link: https://william-yeh.net/post/2020/03/idempotency-key-test/
6. Eclipse Mosquitto, an open source MQTT broker. https://mosquitto.org/

Motivations: data communication applications

- appplications

The client-server model, its pros and cons

- definition
- pros
- cons

The event service model, its pros and cons

- definition
- pros
- cons

The publish-subscribe model, its pros and cons

- definition
- pros
- cons

Data communication models, a quick recap

1. Client-server model
   - RPC (remote procedure call)
   - gRPC (Google's RPC) https://grpc.io/
2. Event service model
   - Event supplier, event consumer, and event channel
   - Event filtering
   - Event delivery is implemented like making two RPC invocations
   - CORBA (Common Object Request Broker Architecture) https://www.corba.org/
3. Publish-subscribe model (aka the pub/sub model):
   - message publiser $\leftrightarrow$ event supplier
   - message subscriber $\leftrightarrow$ event consumer
   - message broker $\leftrightarrow$ event channel

- In the following we shall focus on the pub/sub model

Topic, message, and message delivery

- Topic is the subject of interest
- Message is a piece of data for a certain topic
- Message delivery variations
  - Variation 1: Each subscriber, once subscribed to a certain topic, may receive all published messages of that topic (potentially from different publishers).
    - Example: MQTT https://mqtt.org/
  - Variation 2: Each published message will be delivered to only one of the subscribers.
    - Example: NSQ, a real-time distributed messaging platform https://nsq.io/

A note on the second variation of message delivery

- Why is it useful to deliver each message to only *one* of the subscribers?
- Answer: Sometimes, the message should be processed only once.
  - Example: counting the number of clicks on a web page
  - Typical for cloud computing and big data analytics, in which case the message consumers are cloud servers
- Another perspective: scalability of a service
- Question for you: could we build a messaging service that runs upon the first variation conform to the semantics of the second?

Related messaging platforms and libraries

- Kafka: https://kafka.apache.org/
- Flink: http://flink.apache.org/
- NSQ: https://nsq.io/
- ZeroMQ: https://zeromq.org/
- In practice, people may use them *in combination* to build their business infrastructure!

The need for quality-of-service (QoS)

- Motivations
  1. Application semantics
  2. Service scalability
- Example QoS for message delivery:
  1. At most once (aka best-effort)
  2. At least once
  3. Exactly once

The MQTT protocol

- History
  - Invented in 1999 by Andy Stanford-Clark and Arlen Nipper
  - Internal use at IBM until 2010 (MQTT 3.1)
  - In 2014, approved as an OASIS standard (MQTT 3.1.1)
  - March 2019, MQTT 5 specification
- Design objectives (i.e., requirements)
  - Simple implementation
  - Quality-of-Service data delivery
  - Lightweight and bandwidth efficient
  - Data agnostic
  - Continuous session awareness
- Implementation for MQTT clients
  - https://github.com/mqtt/mqtt.org/wiki/libraries

MQTT terminology

- Client vs. broker
- Sender vs. receiver
- Control packets vs. data packets
- A complete list of MQTT control packets (15 in total):
  - Connection establishment and destruction:
    - CONNECT, CONNACK, DISCONNECT
  - Message subscription:
    - SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK
  - Message delivery and QoS control:
    - QoS 0: PUBLISH
    - QoS 1: PUBLISH, PUBACK
    - QoS 2: PUBLISH, PUBREC, PUBREL, PUBCOMP
  - Liveness:
    - PINGREQ, PINGRESP
  - Authentication:
    - AUTH

Connection establishment

- A client must send a CONNECT control packet to the broker to establish/resume a session.

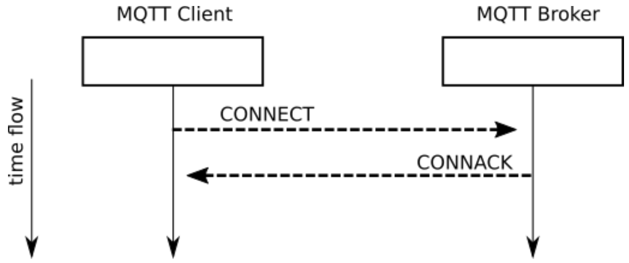


Figure: Sequence diagram of connection establishment

Message subscription

- A client sends a SUBSCRIBE control packet to the broker to subscribe to a set of topics.

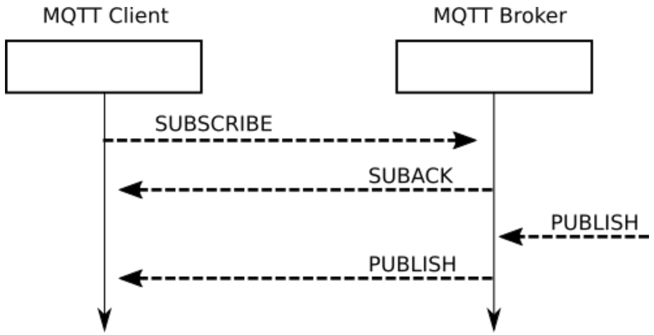


Figure: Sequence diagram of message subscription

Message subscription (cont.)

- Similar to a typical networking protocol, each MQTT packet contains a *header* and a *payload*.

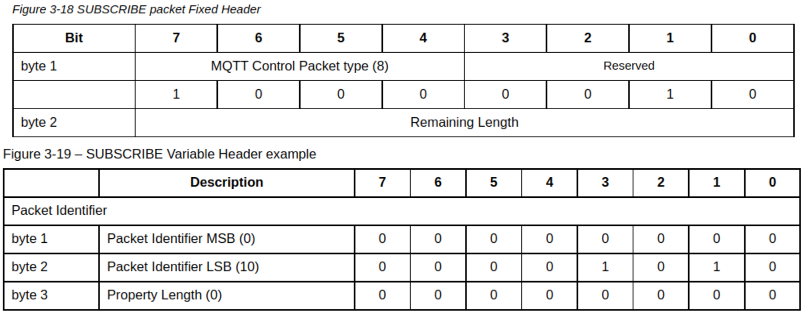


Figure 3-18 SUBSCRIBE packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
byte 2	Remaining Length							

Figure 3-19 – SUBSCRIBE Variable Header example

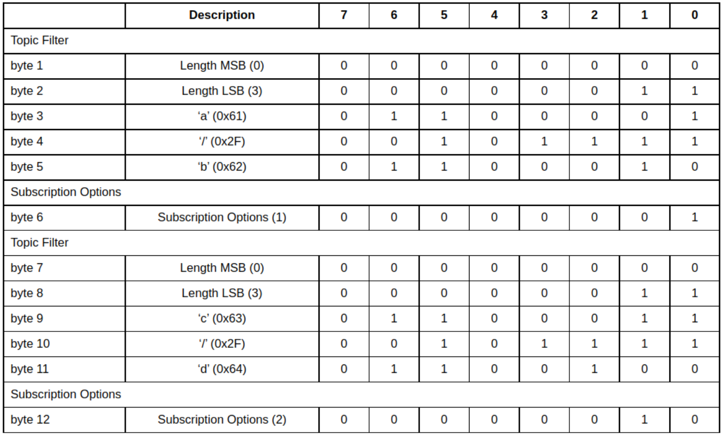
	Description	7	6	5	4	3	2	1	0
Packet Identifier									
byte 1	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 2	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0
byte 3	Property Length (0)	0	0	0	0	0	0	0	0

Figure: Example from MQTT 5 Specification.

Message subscription (cont.)

- An example SUBSCRIBE packet with two topics:

Figure 3-21 - Payload byte format non-normative example



	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	‘a’ (0x61)	0	1	1	0	0	0	0	1
byte 4	‘/’ (0x2F)	0	0	1	0	1	1	1	1
byte 5	‘b’ (0x62)	0	1	1	0	0	0	1	0
Subscription Options									
byte 6	Subscription Options (1)	0	0	0	0	0	0	0	1
Topic Filter									
byte 7	Length MSB (0)	0	0	0	0	0	0	0	0
byte 8	Length LSB (3)	0	0	0	0	0	0	1	1
byte 9	‘c’ (0x63)	0	1	1	0	0	0	1	1
byte 10	‘/’ (0x2F)	0	0	1	0	1	1	1	1
byte 11	‘d’ (0x64)	0	1	1	0	0	1	0	0
Subscription Options									
byte 12	Subscription Options (2)	0	0	0	0	0	0	1	0

Figure: Example from MQTT 5 Specification.

Message publication and QoS (Quality of Service)

- Three quality-of-service levels:
  - QoS 0: at most once delivery
  - QoS 1: at least once delivery
  - QoS 2: exactly once delivery

QoS 0: at most once delivery

- Also known as *the best effort strategy*:

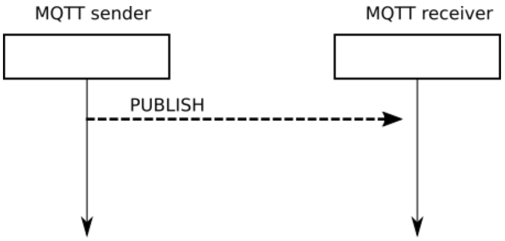


Figure: Sequence diagram of QoS 0 message delivery.

QoS 1: at least once delivery

- Application requirement: each message be delivered at least once
- Design requirement: ensure that *the sender* knows that *the receiver* has received the message

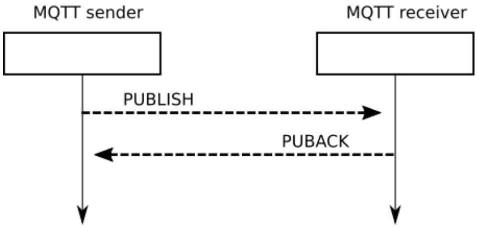


Figure: Sequence diagram of QoS 1 message delivery.

- Question for you: How to make *the publisher* know that *the subscriber* has received the message?

QoS 2: exactly once delivery

- Besides the requirement for QoS 1, ensure that the receiver may safely release the bound on the packet identifier.

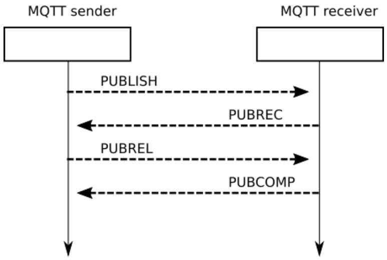


Figure: Sequence diagram of QoS 2 message delivery.

- Compare and contrast: Two-phase commit protocol in reaching a consensus in a distributed system
  - https://en.wikipedia.org/wiki/Two-phase_commit_protocol

QoS consolidation

- note:

Alternative to QoS 2: leveraging idempotency

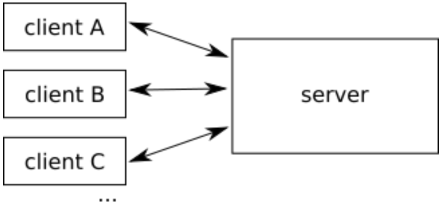
- To meet the application's requirement of exactly once message delivery, it may suffice to just enforce the at least once semantics at MQTT, provided that the receiver of the message can handle duplicated messages in the sense that *the receiving of duplicated messages would impact the receiver in the same way as if it only received one of them.*
- Example idempotent operations
- A practical implementation to enforce idempotency: idempotency key

Request/response using MQTT

- In some sense, it works like supporting RPCs in MQTT ;)
    - review page 8
- Motivating examples:
    1. Polling some sensor data
    2. Commanding some actuator execution
    3. Offloading some computation to a cloud server
- To learn more, see Section 4.10 in the MQTT 5 Specification
    - https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

Messaging server design considerations

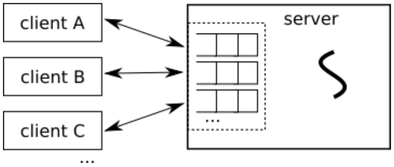
- Service design for latency performance



- Reactive server vs. concurrent server
- Patterns for concurrency design
  - The Half-Sync/Half-Async pattern
  - The Leader/Followers pattern

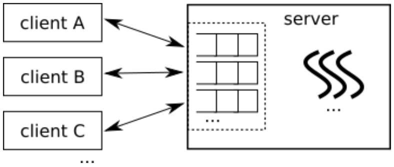
Reactive server

- Requests from multiple endpoints were handled in round-robin order.



Concurrent server

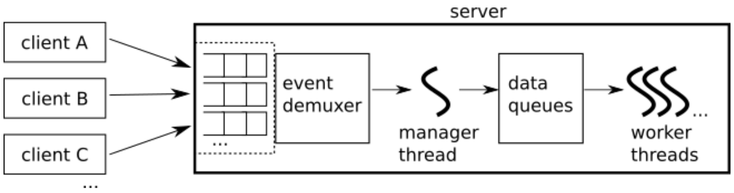
- The server uses multiple threads to handle requests simultaneously
  - Spawning a new thread per request
  - Using a pool of pre-spawned threads



- Question for you:
  - For Internet-of-Things applications with low data rates, would you choose a reactive server or a concurrent server?

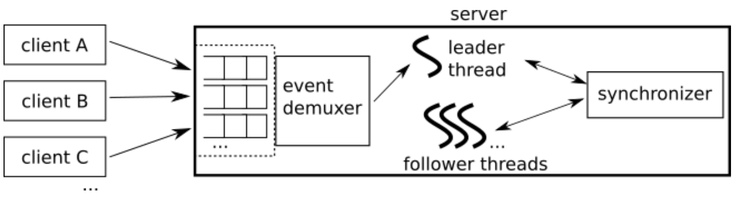
The half-sync/half-async pattern

- Three layers: the asynchronous layer, the queueing layer, and the synchronous layer
- Separation of concerns
- Context switch issues



The leader/followers pattern

- No context switch
  - one thread (the leader thread) would work to completion, and one of the pending threads (the follower threads) would take over as a new leader.
- More complex to implement



- Compare and contrast: In which application scenarios would you choose this one over the half-sync/half-async pattern?

Mosquitto an MQTT implementation

- Official website: https://mosquitto.org/
- A public MQTT broker for testing: https://test.mosquitto.org/
- The Mosquitto broker implements a reactive server
- We will look at the implementation next week!

Takeaway today

- Broker-based data communication:
  - model: the pub/sub model
  - design: MQTT
  - implementation (next week): Eclipse Mosquitto
- Messaging server design and implementation