ECE 448/528
Application Software Design

Lecture 11. Publish-Subscribe Pattern and MQTT (Message Queuing Telemetry Transport)

Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Publish-Subscribe Pattern

Publish-Subscribe Pattern (Pub/Sub)

- A design pattern to further decompose event processing and event delivery; on top of network communications.
- **Publisher**: parties that produce events
- **Subscriber**: parties that consume events
- **Message broker**: a service that collects events from publishers and delivers them to subscribers.
  - Known as event bus, message queue middleware, etc.
  - Events are usually encoded into string/byte messages so that a message broker may work independently of specific applications.
- The publish-subscribe (pub/sub) pattern fully decouples event processing with event delivery, though there are many features an application needs to carefully choose from.

Observer Pattern Revisited

- In comparison with the observer pattern
  - **EventSource** is similar to the **Publisher** as both produce events.
  - **Observer** is similar to the **Subscriber** as both consume events.
- However, unlike the observer pattern where the EventSource knows and manages the observers, publishers and subscribers are fully decoupled.
- Publishers only communicate with the message broker to send events.
  - Publishers have no idea what subscribers are consuming the events they produce.
- Subscribers only communicate with the message broker to receive events.
  - Subscribers have no interest in who produces the events they consume.
- How does the message broker know to which subscribers those events should be delivered to?

Topic Management

- For each event, in addition to the message representing the event, the publisher needs to specify a ***topic***.
  - The topic usually is a meaningful string providing hints on what this message/event is all about.
  - For each event, the publisher sends (topic, message) to the message broker.
- Each subscriber, when establishing a connection with the message broker, should specify what topics it is interested in.
  - Then the message broker will only send (topic, message) with matching topics to this subscriber.

Topic Routing

- A subscriber may simply specify all topics it is interested in.
- This simple strategy may fail:
  - When there are too many such topics.
  - When new publishers start to publish new topics that the subscriber may be interested in.
- A message broker may allow subscribers to subscribe to a wildcard of topics.
  - Whether or not the subscriber has the capacity to process messages from all matching topics is not of concern here.

MQTT Topics Examples

MQTT Topics

Illinoistech/armourcollege/ece/engineering

- /: topic level separator; separates topic levels
- each topic must contain at least 1 character
- Topics are case-sensitive (Illinoistech vs illinoistech)

MQTT Wildcards (Single Level: +)

- Wildcards to be used to subscribe to multiple topics simultaneously
- A wildcard can be used to subscribe to topics only, not to publish a message

Illinoistech/armourcollege/+/engineering

✓ Illinoistech/armourcollege/ece/engineering
✓ Illinoistech/armourcollege/caee/engineering
X Illinoistech/siegelhall/ece/engineering
X Illinoistech/armourcollege/siegehall/ece/engineering

MQTT Wildcards (Multiple Level: #)

- Covering many topic levels but only allows at the end!

Illinoistech/armourcollege/#

✓ Illinoistech/armourcollege/ece/engineering/computer
✓ Illinoistech/armourcollege/caee/engineering/civil
X Illinoistech/siegelhall/ece/engineering

Delivery Guarantee

- Should the message broker acknowledge the publisher and should the subscriber acknowledge the message broker of the message being received?
  - A fundamental problem of network communication and messaging systems.
- Delivery Guarantee
  - At least once: resend messages not acknowledged
  - Exactly once: how?
  - At most once: don’t acknowledge at all
- It is up to the applications to choose a proper delivery guarantee.
  - And we may need to redesign an application if the delivery guarantee cannot be achieved.

Delivery Guarantee: Exactly Once

- The publisher may include a sequence number for each message.
- Publisher/message broker/subscribers all utilize at "least once" delivery.
- The message broker and subscribers use the sequence number to remove duplicated messages to achieve "exactly once" delivery.

Message Ordering

- Could assume network communications always arrive in order.
  - But some messages may get lost and need to be re-sent according to the delivery guarantee.
- At most once: messages always/should/must arrive in order
- Exactly once: messages may arrive out of order
  - Order can be recovered via the sequence number.
- At least once: depend on performance requirement
  - Wait for acknowledgment before sending a new message
    - in-order arrival but low performance.
  - Allow multiple unacknowledged messages: better performance but a lost message being resent may arrive out of order.

Persistence

- A subscriber may need to receive all historical messages published to a topic.
  - As needed by the application.
- A message broker may support persistence, storing all historical messages.
  - Persistence may require a substantial amount of storage.
- It may take a lot of time for a subscriber to receive all historical messages.
  - What if the subscriber fails and needs to restart?
  - Subscriber local state: assume messages are with sequence numbers, then a subscriber may store messages locally and request the message broker to send messages after the last one upon restarting.

Retained Messages

- Without persistence, a subscriber may need to wait indefinitely before a message is published.
  - Could be a problem if the subscriber needs to decide the status of the publisher and the publisher cannot publish its status periodically.
- Retained messages: the message broker only persists the last message on a topic.
  - A lightweight alternative to persistence for some cases.
  - Either as requested by the publisher or as configured.
  - New subscribers will receive it as the first message.

Multiple Publishers and Subscribers

- Multiple publishers are typically allowed to publish to the same topic.
  - There should be no communication between publishers.
  - No guarantee on ordering of messages from different publishers.
  - Each publisher should maintain its sequence number if needed.
- Multiple subscribers who subscribe to the same topic are not aware of each other.
  - Performance may degrade if there are more subscribers.

Multiple Message Brokers

- It is possible to utilize multiple message brokers.
  - Improve system performance when there are more publishers and subscribers.
  - Provide failover if a message broker fails.
- Publishers and subscribers need to be aware of those message brokers.
  - They would utilize timeout to decide if a message broker is failing and to switch to a different one if so.
- The message brokers will need to rely on certain consensus protocols to synchronize received messages.
  - A very important problem of distributed computing.

MQTT
Message Queuing Telemetry Transport

Message Broker Design Trade-Offs

- With so many features to choose from, many of which affect each other, many message broker products are making different design trade-offs.
- You may find two extremes.
  - A lightweight one that focuses on simplifying message delivery but provides no persistence or failover.
  - A full-featured one that would meet all needs by providing persistence and failover.
- Which one is better? It all depends on what is required by your application.

MQTT (MQ Telemetry Transport)

- A lightweight publish-subscribe protocol.
  - Designed in the late ’90s for devices with limited resources and bandwidth.
  - An open OASIS and ISO standard and widely used for IoT devices now.
- Architecture
  - A MQTT broker to relay the messages.
  - Clients connected to the MQTT broker to publish messages and subscribe to topics.

MQTT Features

- Topic: a UTF-8 string
  - Consist of topic levels separated by `/`.
  - E.g. `iot_ece448/action/plugA/on`
  - Clients may use wildcards to subscribe for multiple topics.
  - e.g. `iot_ece448/action/#` for any topics starting with `iot_ece448/action/`
- Support at most once, at least once, and exactly once delivery.
  - But please keep in mind since persistence is not supported, all delivery guarantees do not apply to past messages, in particular when a client needs to restart.
- Support retained messages.

Eclipse Mosquitto

- An open-source MQTT broker.
- Installed and runs on the course VM.
  - Use default TCP port 1883.
- Use `mosquitto_sub -t topic` to subscribe to `topic`.
- Use `mosquitto_pub -t topic -m message` to publish `message` to `topic`.
- By default, both `mosquitto_sub` and `mosquitto_pub` connect to the MQTT broker on localhost using the default TCP port.