Benchmarking di broker MQTT open-source

Pasquale Caramante
Università degli Studi di Salerno
Corso di Reti Geografiche: Struttura,Analisi e Prestazioni
Laurea Magistrale in Informatica - LM 18

MQTT

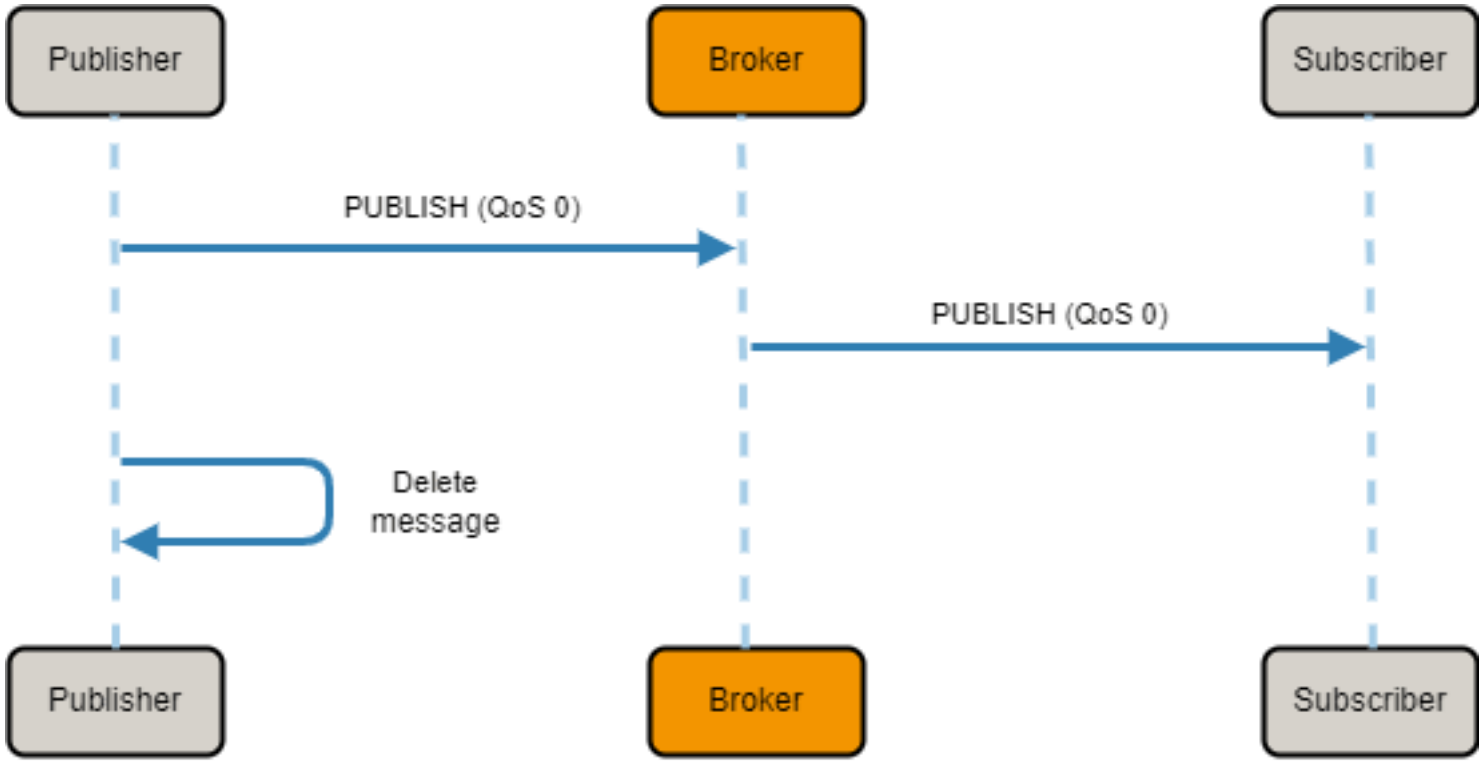
- **MQTT** è un protocollo di messaggistica *lightweight* basato su **publish/subscribe**, pensato per dispositivi con risorse limitate e reti con larghezza di banda ridotta.
- Standardizzato da OASIS / ISO.
- Caratteristiche chiave:
  - **Overhead minimo**, adatto a dispositivi embedded;
  - Tre livelli di **QoS** per garantire **affidabilità** in reti instabili;
  - Supporto per **sessioni persistenti**, messaggi “retained”, *Last Will*);
  - Sicurezza via **TLS/SSL**, autenticazione e autorizzazione;
- Applicazioni tipiche: IoT, App di messaggistica (Facebook Messenger)

MQTT Workflow

1. Connessione
   - Il client apre una **connessione TCP** (opzionalmente con TLS) verso il broker;
   - Il client stabilisce una **sessione MQTT**, inviando un pacchetto `CONNECT` al broker;
2. Publish/Subscribe
   - Il client può iscriversi (`SUBSCRIBE`) e/o pubblicare (`PUBLISH`) messaggi su un topic specifico;
3. Routing
   - Il broker distribuisce i messaggi ai client che si sono iscritti a quel topic, rispettando le regole di QoS.
4. Disconnessione
   - Quando un client si disconnette “normalmente” invia un messaggio `DISCONNECT` al broker;
   - Se la disconnessione è anomala il broker può pubblicare un **Last Will** predefinito;

QoS 0 — At most once

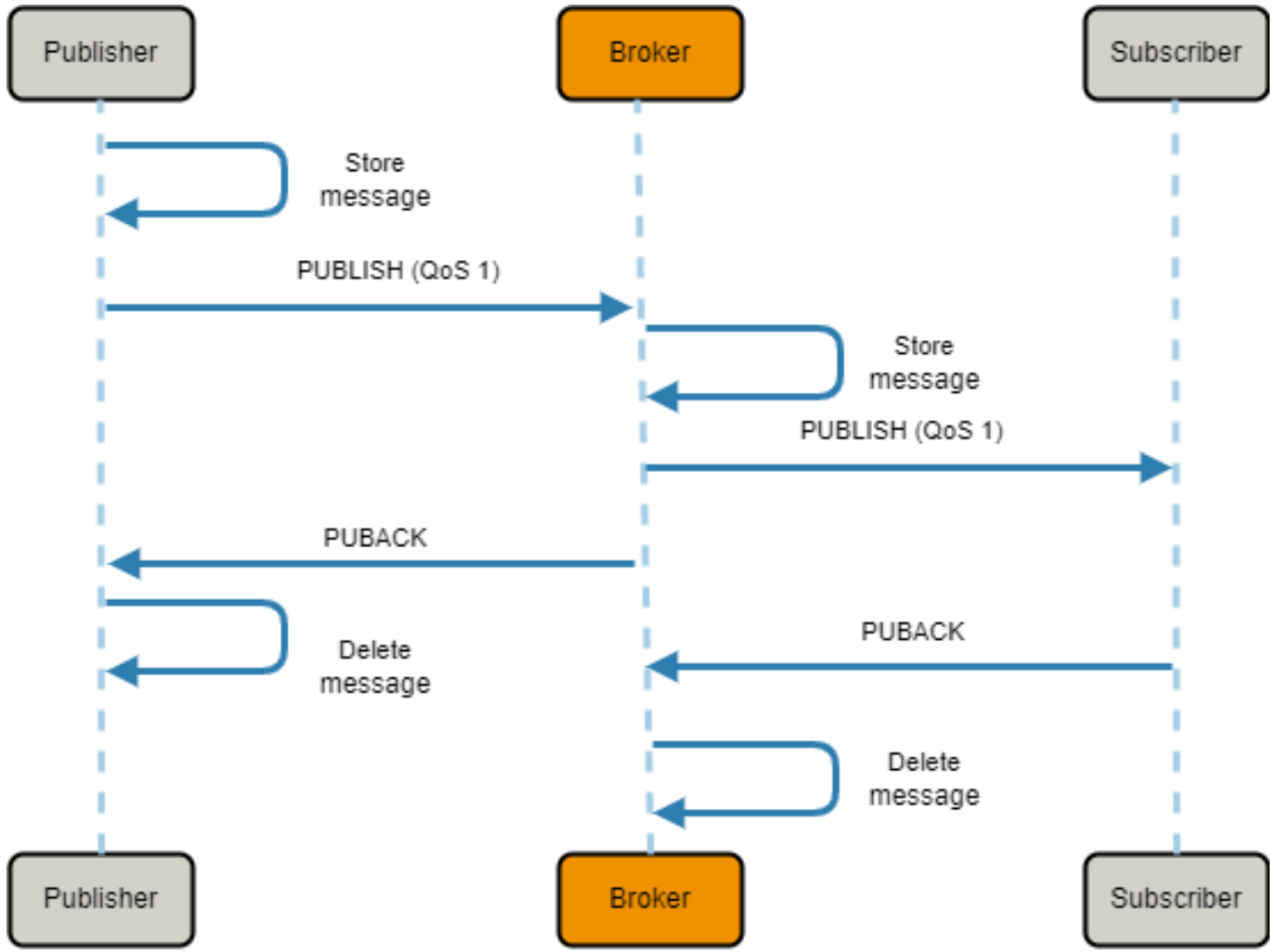
- Nessuna conferma di ricezione;
- È la modalità più veloce ma non garantisce affidabilità.



PUBLISH: Publish Message

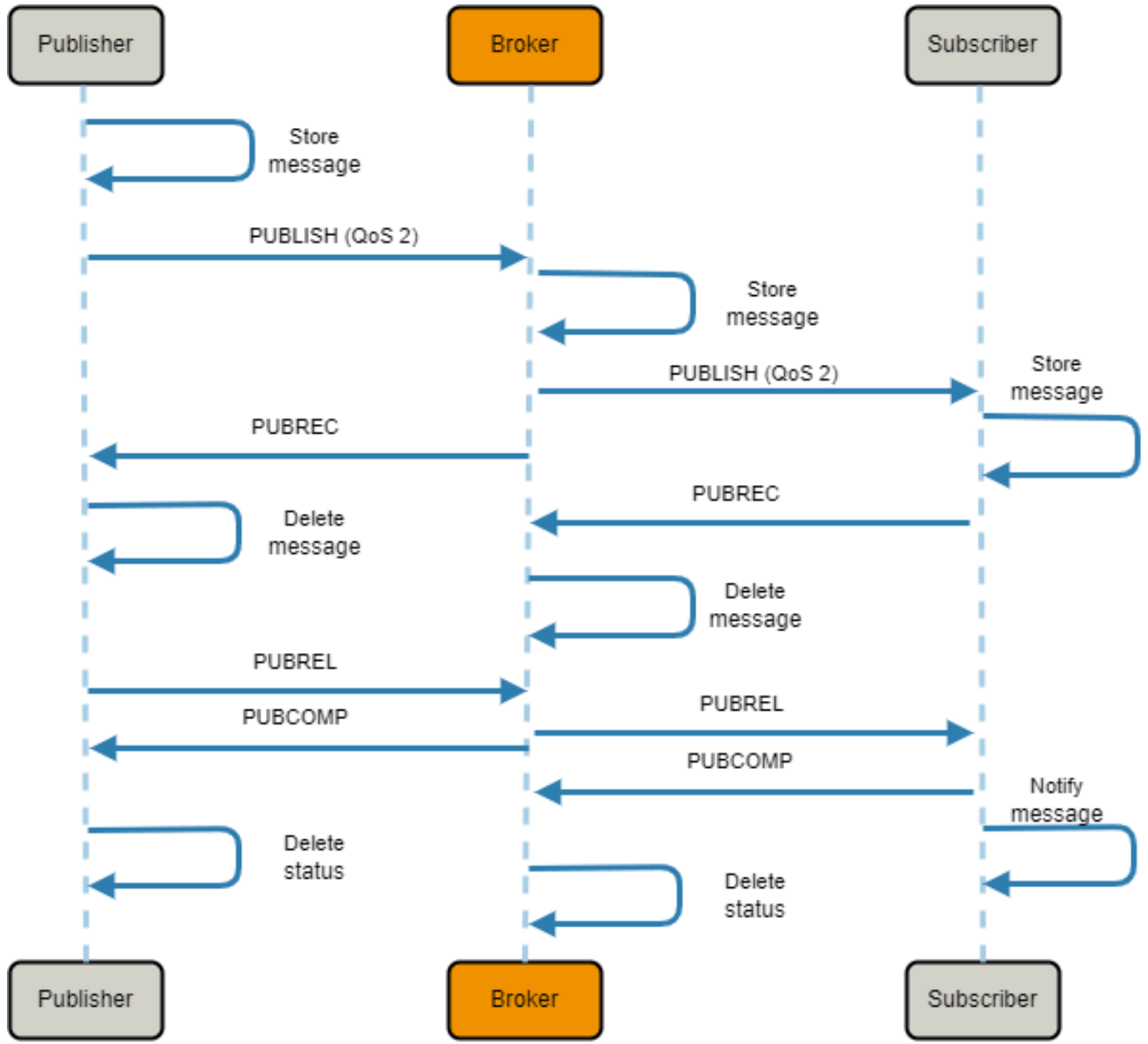
QoS 1 — At least once

- Richiede conferma di ricezione con pacchetto **PUBACK**;
- Possibile duplicazione dei messaggi.



QoS 2 — Exactly once

- Richiede un handshake a quattro fasi (PUBREC, PUBREL, PUBCOMP);
- Massima affidabilità ma con overhead maggiore.



Obiettivi

Il **broker** è il cuore del sistema MQTT: gestisce le connessioni, le sessioni e i topic, instradando i messaggi ricevuti ai destinatari corretti;

- **Obiettivo**: Testare diverse implementazioni di broker MQTT;
- Metriche di interesse:
  - **Latenza end-to-end**;
  - **Throughput dei messaggi**;
  - **Consumo di CPU** e **Memoria**

Le prestazioni di un broker possono essere influenzate da diversi fattori:

- Implementazione del broker;
- Configurazione MQTT;
- Architettura Pub/Sub;

Broker testati

Broker	Linguaggio	Multithread	Versioni MQTT	Max QoS	Transport Layer
Mosquitto	C	No	3.1, 3.1.1, 5.0	2	TCP, TLS, WebSocket
EMQX	Erlang/OTP	Sì	3.1, 3.1.1, 5.0	2	TCP, TLS, WebSocket, QUIC
NanoMQ	C (NNG)	Sì	3.1.1, 5.0	2	TCP, TLS, WebSocket, QUIC
rumqttd	Rust	Sì	3.1.1, 5.0	2	TCP, TLS, WebSocket
RMQTT	Rust	Sì	3.1.1, 5.0	2	TCP, TLS

Tool di Benchmarking — emqtt-bench

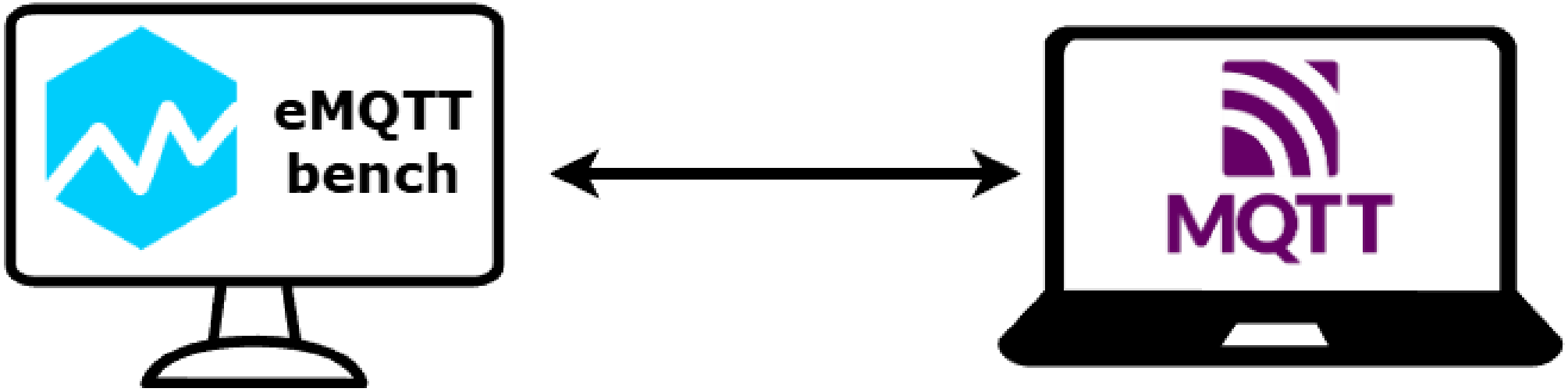
```
emqtt_bench pub -h localhost -p 1883 -t test/topic/%i -c 100 -q 1 -I 100 -s 64
emqtt_bench sub -h localhost -p 1883 -t test/topic/%i -c 100 -q 1
```

- `-t` : Topic di sottoscrizione/pubblicazione
- `-c` : Numero di client
- `-q` : Livello di QoS
- `-I` : Rate di pubblicazione dei messaggi (per ciascun publisher)
- `-s` : Dimensione del payload

Output:

```
11s recv total=160 rate=160.0/sec
11s publish_latency avg=5ms
12s recv total=448 rate=288.0/sec
12s publish_latency avg=8ms
```

Ambiente di Test



Intel(R) Core(TM) i7-8700K @ 3.70GHz 6 core / 12 threads

Memory: 16 GB DDR4

OS Ubuntu 24.04 LTS, 64 bit

Intel(R) Core(TM) i7-3610QM @ 2.3 GHz 4 core / 8 threads

Memory: 16 GB DDR3-SDRAM

OS, Kernel: Ubuntu 24.04 LTS, 64 bit

Tutti i broker sono stati eseguiti tramite **Docker** container, per facilitarne l'installazione e non incorrere nell'overhead di VM tradizionali.

Primo test - Architettura Pub/Sub

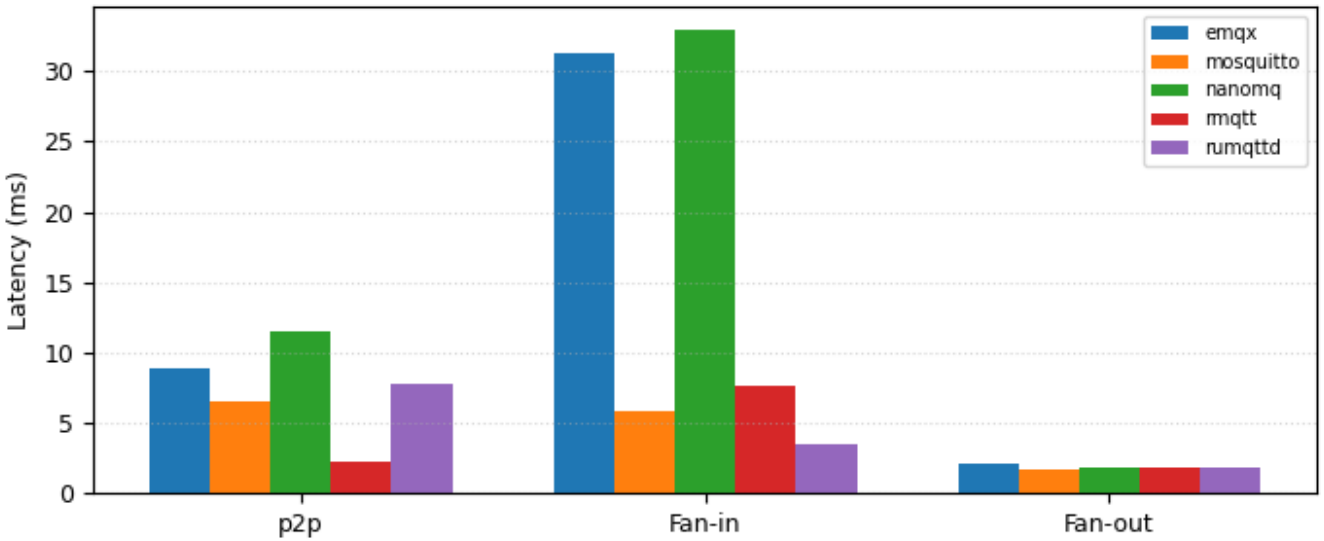
	Point-to-Point	Fan-In	Fan-Out
#Pub	100	100	1
#Topic	100	100	1
#Sub	100	1	100

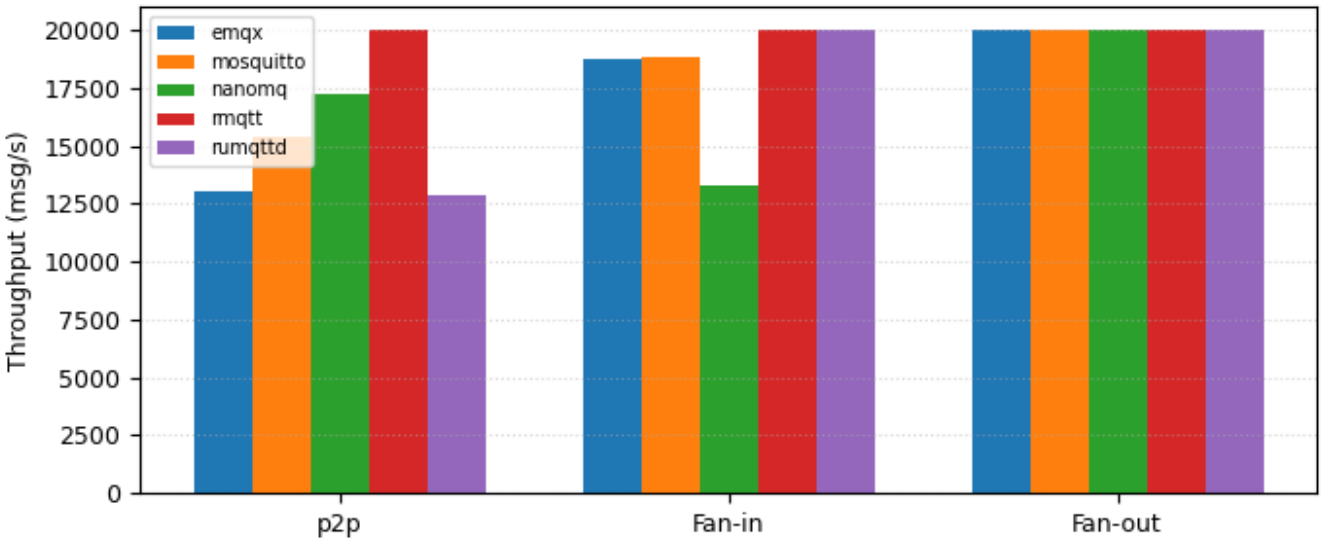
- QoS 1
- Payload 64 byte
- Publishing Rate: 20 msg/sec per publisher

Per ciascun broker eseguiamo un test della durata di **5 minuti** per ciascuna architettura, raccogliendo le metriche di **latenza**, **throughput** e consumo di **CPU** e **Memoria** medi.

- L'intero benchmark è stato ripetuto per 5 volte, riportando per ciascun broker i migliori risultati ottenuti.

Risultati - Architettura Pub/Sub



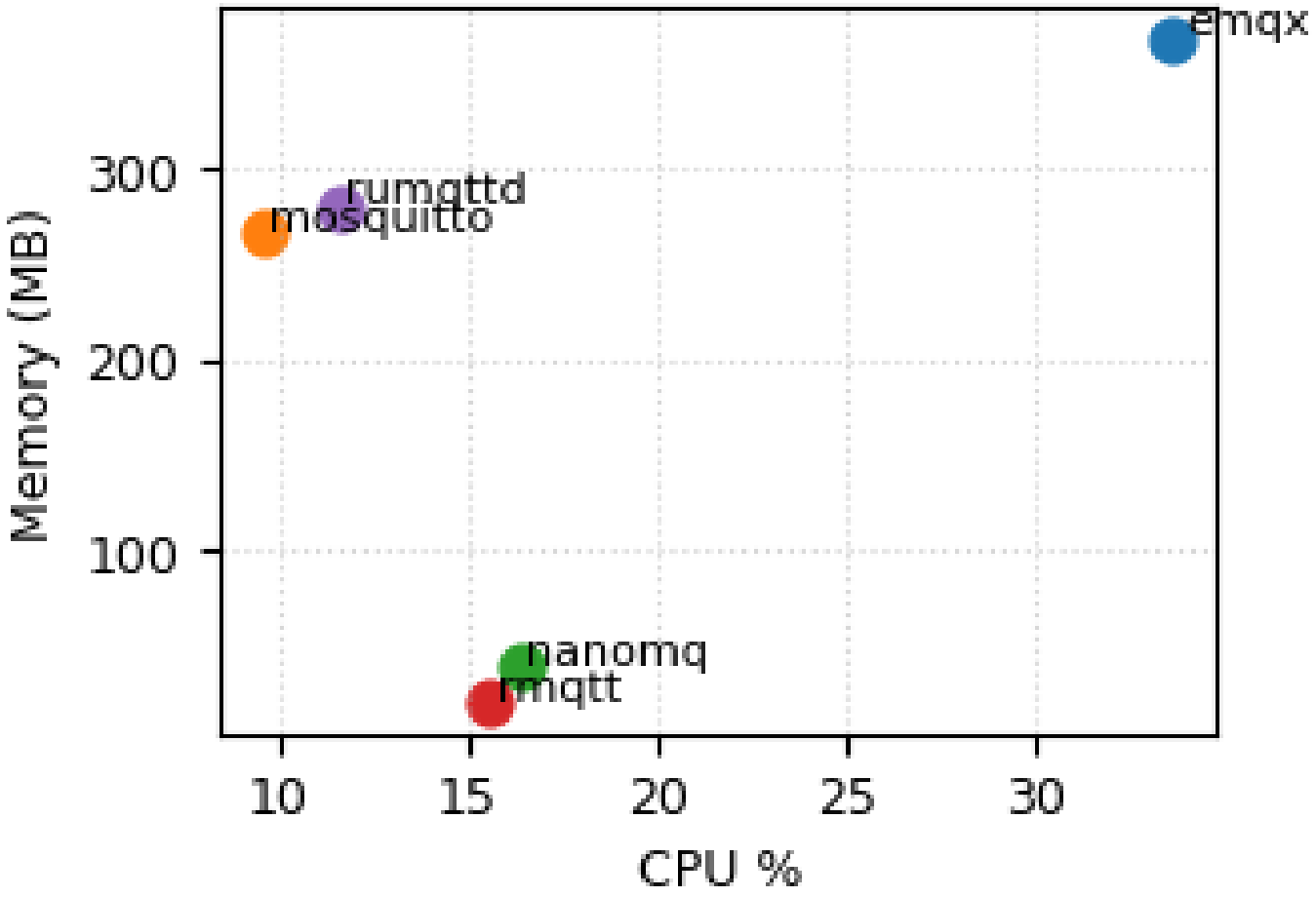


P2P: *RMQTT* e *Mosquitto* mantengono le migliori prestazioni, mentre EMQX e rumqttd soffrono di più.

Fan-In: latenza variabile (3.5–33 ms) e throughput medio-alto (~13–19k msg/s). EMQX e NanoMQ hanno latenza maggiore;

Fan-Out: Tutti i broker gestiscono bene il carico con latenza minima e throughput massimo

Risultati - Architettura Pub/Sub

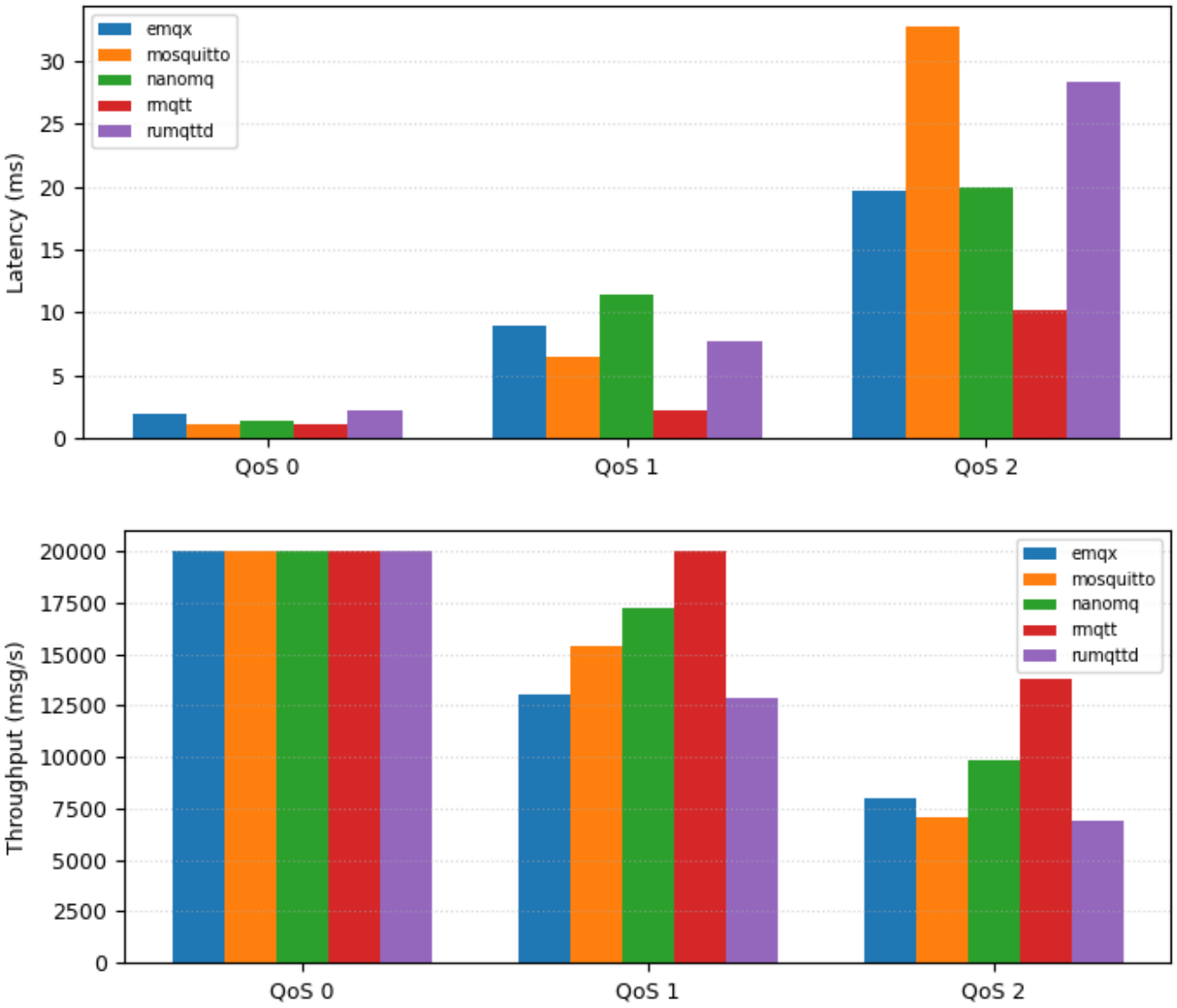


Secondo test - QoS

- Architettura **Point-to-Point**:
  - 100 publisher
  - 100 topic
  - 100 subscribers
- Payload 64 byte
- Publishing Rate: 20k msg/sec (20 msg/sec per publisher)

Test della durata di 5 minuti per ogni livello di QoS (0,1,2).

Risultati - QoS (Latenza e Throughput)

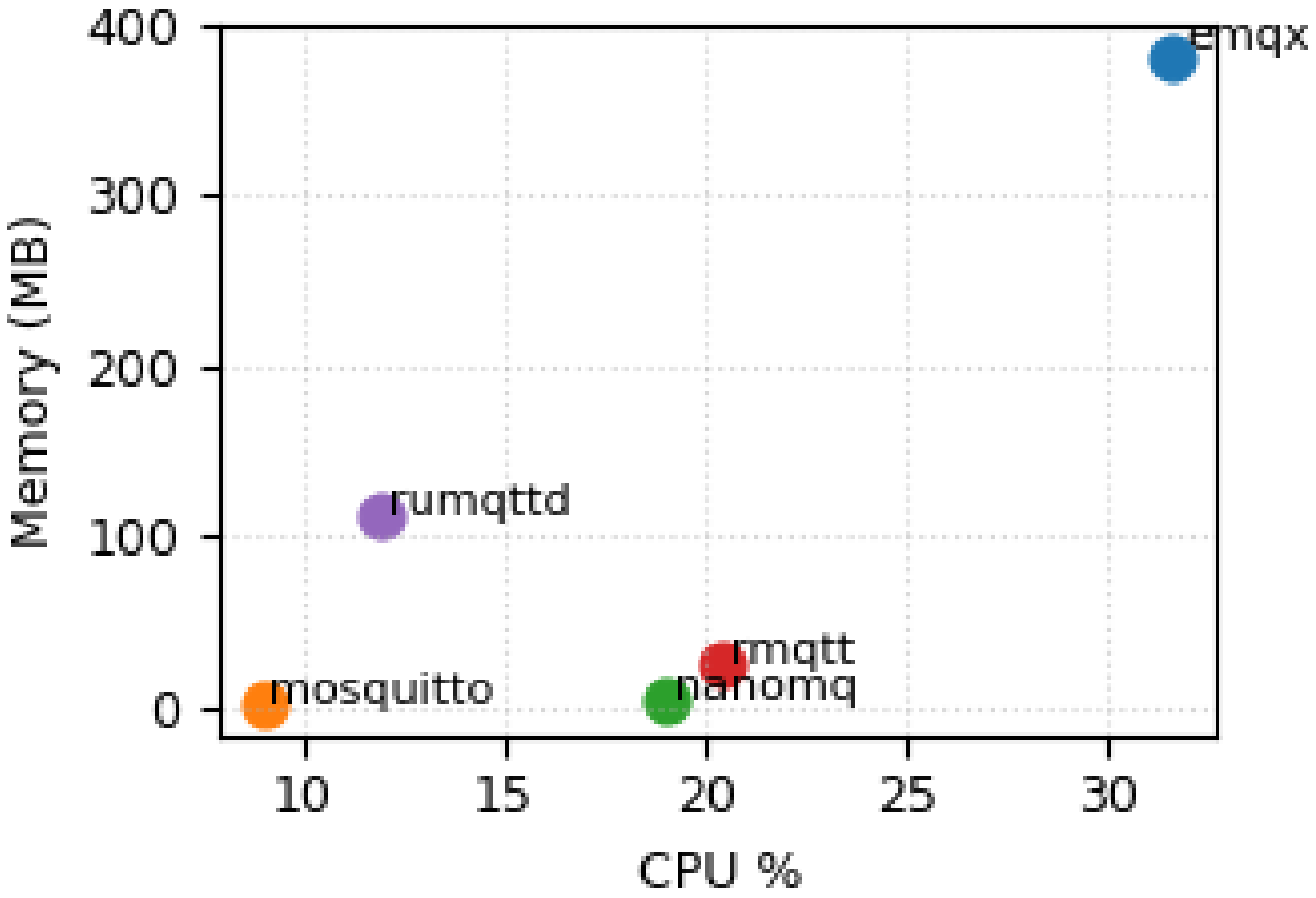


QoS 0: latenza minima (~1–2 ms) e throughput massimo (~20k msg/s) per tutti i broker.

QoS 1: la latenza cresce (fino a ~10 ms) e il throughput si riduce (~13–17k). *RMQTT* mantiene le migliori prestazioni (~20k msg/s).

QoS 2: latenza elevata (20–30+ ms) e throughput dimezzato (6–9k). *RMQTT* resta il più efficiente (~13.7k), mentre Mosquitto e rumqttd soffrono di più.

Risultati - QoS (Utilizzo risorse)



Fine