Analisi delle problematiche di sicurezza relative al protocollo MQTT

Edoardo Di Paolo

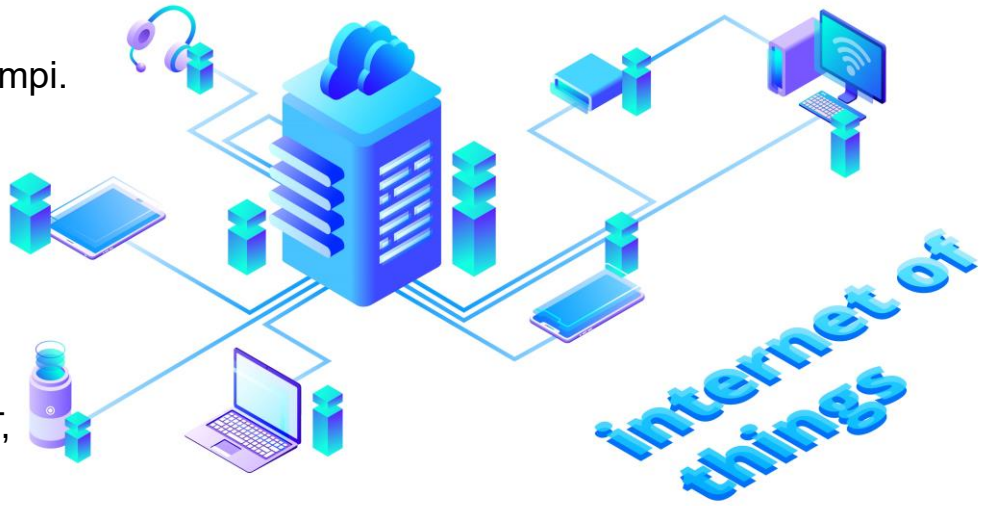
Corso di Laurea in Informatica

A.A. 2019/2020

SAPIENZA
UNIVERSITÀ DI ROMA

Presentazione dello scenario

- L’**Internet of Things (IoT)** è in continua evoluzione e sempre più dispositivi son connessi simultaneamente.
- L’IoT coinvolge continuamente nuovi campi.
- Aumento degli attacchi nella rete Internet.
- Sviluppo di nuovi protocolli come MQTT, CoAP e AMQP.



Il protocollo MQTT

- **MQTT** (*Message Queue Telemetry Transport*) è un protocollo di tipo **publish-subscribe**.

- Uso del protocollo **aumentato** di molto negli ultimi anni a causa della crescita del numero di dispositivi IoT connessi.

- Molti dispositivi collegati in rete senza alcuna protezione per quanto riguarda l'accesso. **Chiunque può entrare.**

MQTT nel 2018

MQTT nel 2020

Il protocollo MQTT

- Il protocollo è semplice da utilizzare ed è adattabile sia a sistemi semplici che a sistemi molto complicati.

- Per poter funzionare richiede pochissima banda e pochissime risorse da parte del dispositivo, perciò è un protocollo definito *leggero*.

- Supporta la comunicazione attraverso TLS/SSL per *cryptare* la connessione tra dispositivo e server.

- Può assicurare la ricezione dei messaggi attraverso il *Quality of Service*.

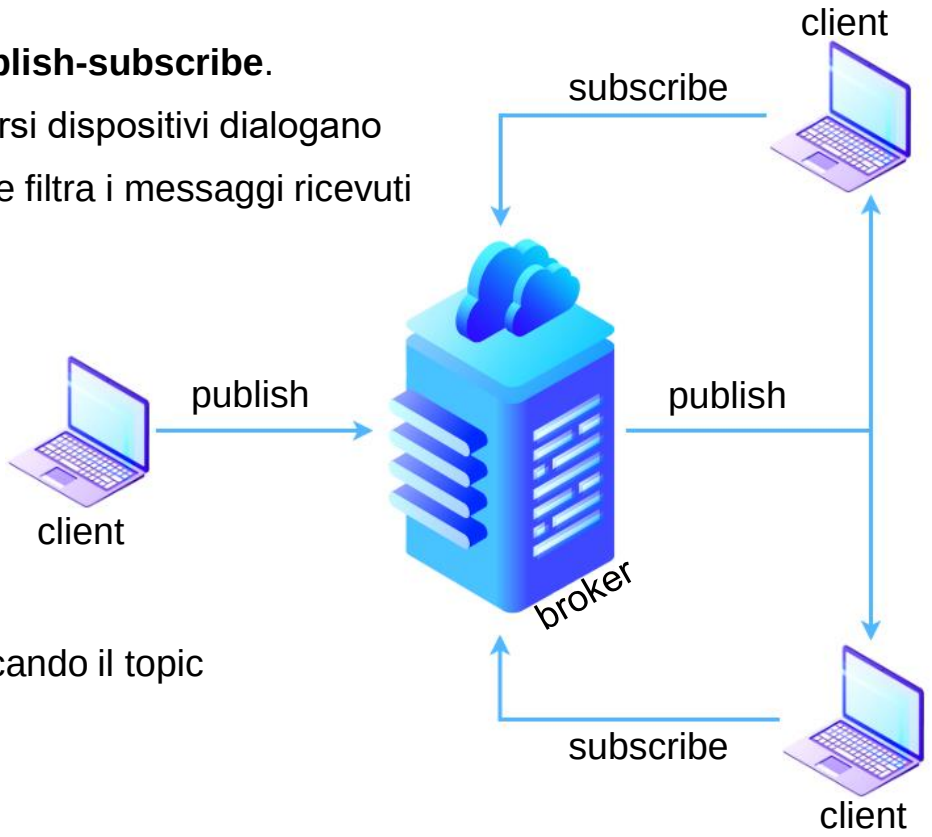
Il protocollo MQTT: publish-suscribe

- L'architettura del protocollo è del tipo **publish-subscribe**.
  In un'architettura di questo genere, i diversi dispositivi dialogano attraverso un tramite chiamato *broker* che filtra i messaggi ricevuti in base al topic del pacchetto ricevuto.

- I diversi client non comunicano *mai* direttamente tra di loro.

- Un client può pubblicare un messaggio attraverso il pacchetto *«publish»,* specificando il topic a cui pubblicare

- Un client può sottoscriversi ad un topic attraverso il pacchetto *«subscribe».*



Il protocollo MQTT: Quality of Service

Il **Quality of Service** è un *«contratto»* stipulato tra mittente e destinatario che definisce la garanzia di consegna di un messaggio. In MQTT ci sono **3** livelli di QoS:

- *Livello 0:* in questo caso non c'è garanzia della consegna del messaggio poiché il destinatario non conferma la ricezione del messaggio;

- *Livello 1:* in questo caso c'è la garanzia che il messaggio venga consegnato almeno una volta al destinatario. Il mittente memorizza il messaggio finché non riceve indietro un pacchetto *PUBACK* che conferma la ricezione del messaggio, tuttavia è possibile che il messaggio venga inviato o consegnato più volte;

- *Livello 2:* in questo caso c'è la garanzia che il messaggio venga consegnato *esattamente* una sola volta ai destinatari. Questo livello di servizio è il più alto ma allo stesso tempo il più lento. Si ha un doppio scambio di pacchetti fra *client* e *broker*: prima viene ricevuto il *PUBREC* dal client che a sua volta invia un *PUBREL* e infine riceve indietro un *PUBCOMP*.

MQTT Broker

Il broker, nel protocollo MQTT, ha il compito di filtrare i messaggi che riceve e di distribuirli ai vari subscribers.

- **MOSQUITTO:** broker molto utilizzato, open source e leggero. Supporta tutte le versioni del protocollo;
- **EMQ X:** broker molto utilizzato, open source scritto in *Erlang.* Permette di gestire milioni di connessioni simultanee anche con un unico server;
- **HiveMQ Community Edition:** scritto in *Java* ed open source, supporta tutte le versioni disponibili di MQTT;
- **Moquette:** broker meno conosciuto scritto in *Java* ed open source, supporta tutte le versioni disponibili di MQTT;
- **Aedes:** broker meno conosciuto scritto in *NodeJS* e *non* supporta MQTT 5. Ha molte librerie con le quali può essere integrato.

Implementazione del protocollo

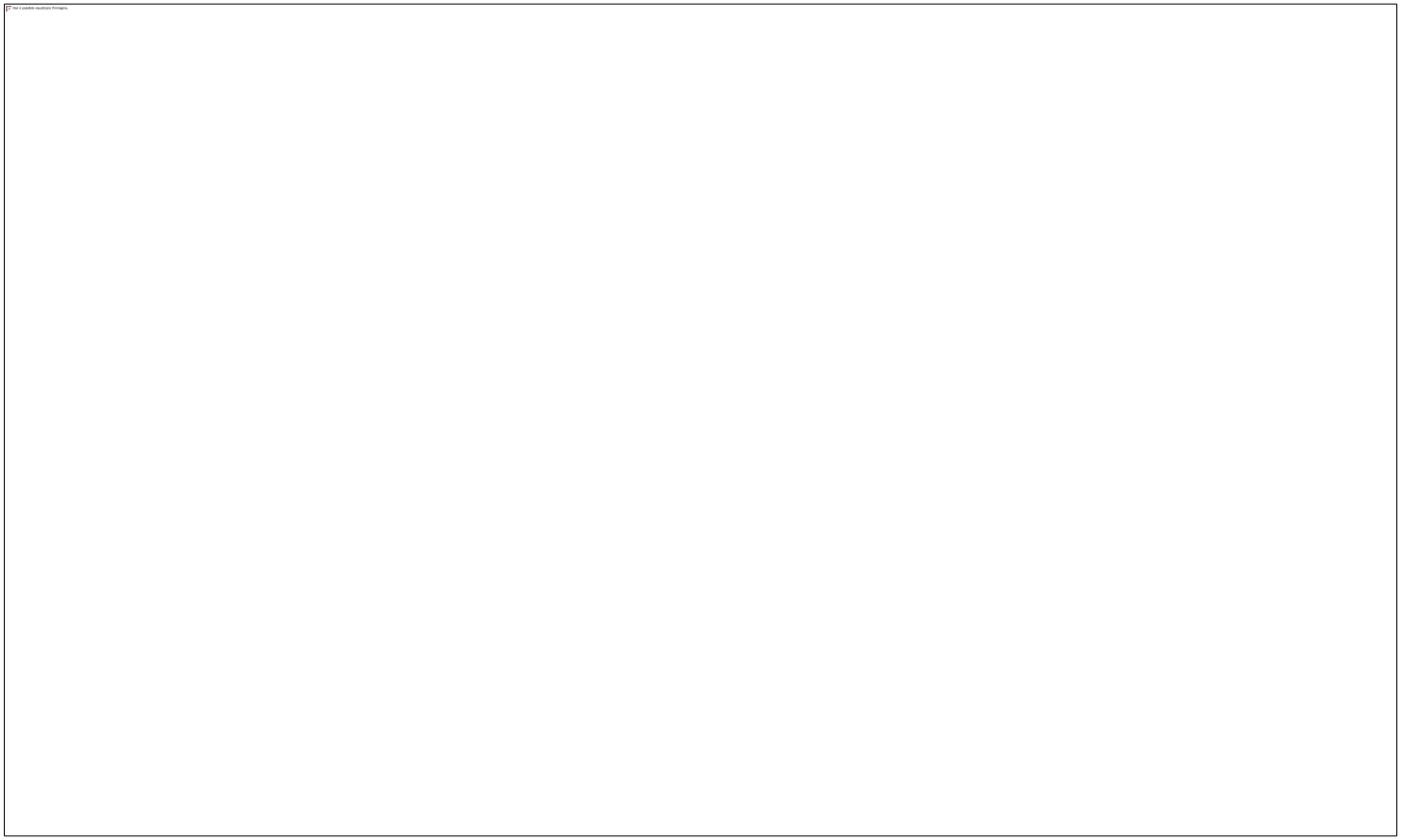
- Il protocollo è stato implementato al fine di poter eseguire degli **esperimenti nel dettaglio**.
- Sono stati implementati tutti i pacchetti più importanti offerti da MQTT.
- Per il **trasporto** dei pacchetti è stata utilizzata la libreria *twisted*.
- **Implementazione** del pacchetto *publish*.

```
def publish(self, topic, message, dup=False, qos=0, retain=False, messageId=None):
    print(self.sentPacketColor + " PACKET SENT => PUBLISH [QoS: "+ str(qos) +", id: "+ str(messageId) +", payload: "+ str(message) +"]" + self.endColor)
    header = bytearray() # fix header
    varHeader = bytearray() # variable header
    payload = bytearray() # payload

    header.append(0x03 << 4 | dup << 3 | qos << 1 | retain) # campi del fix header (tipo pacchetto, duplicate flag, QoS, retain)

    varHeader.extend(_encodeString(topic.encode('utf-8'))) # topic nel var header


    if qos > 0:
        if messageId is None:
            varHeader.extend(_encodeValue(random.randint(1, 65535)))
        else:
            varHeader.extend(_encodeValue(messageId))

    payload.extend(_encodeString(message.encode("utf-8")))  # messaggio del  pacchetto
    header.extend(_encodeLength(len(varHeader) + len(payload))) # variable header + payload

    # trasporto del pacchetto
    self.transport.write(header)
    self.transport.write(varHeader)
    self.transport.write(payload)
```

Selezionare per scrivere o eliminare il sottotitolo

Selezionare per scrivere o eliminare il sottotitolo

Selezionare per scrivere o eliminare il sottotitolo



Selezionare per scrivere o eliminare il sottotitolo