Chat MQTT Universitaria

Implementazione Paradigma PUB/SUB con Mosquitto in C

Mattioli Simone

Caratteristiche Principali:

- Sistema di messaging MQTT per ambiente universitario
- Gerarchia di ruoli con permessi differenziati
- Implementazione in C con libreria Mosquitto
- Analisi traffico con Wireshark
- Build automation con Makefile

22 luglio 2025

Indice

1 Introduzione

Informazione

Cos'è MQTT?
MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero progettato per dispositivi con risorse limitate e reti con larghezza di banda ridotta. Utilizza il paradigma publish/subscribe per la comunicazione asincrona.

Questo progetto implementa un sistema di chat universitaria basato su MQTT che simula la comunicazione tra diversi ruoli accademici: Rettore, Segreteria, Docenti e Studenti. Il sistema utilizza una gerarchia di topic strutturata per gestire i permessi di accesso e la distribuzione dei messaggi.

1.1 Obiettivi del Progetto

- Implementare il paradigma PUB/SUB usando Mosquitto
- Creare una gerarchia di permessi basata sui ruoli
- Dimostrare la gestione delle connessioni e dei messaggi MQTT
- Fornire un'interfaccia utente interattiva
- Permettere l'analisi del traffico con Wireshark

2 Architettura del Sistema

2.1 Gerarchia dei Topic

Il sistema utilizza una struttura gerarchica di topic MQTT per gestire i permessi e il routing dei messaggi:

```
universita/
          personale/
                docenti/
                      [username]
                studenti/
                      [username]
                segreteria/
                    [username]
          amministrazione/
              rettore/
                  [username]
```

Listing 1: Struttura Topic MQTT

2.2 Ruoli e Permessi

2.2.1 RETTORE

- **Sottoscrizione:** `universita/#` (accesso completo)

- **Pubblicazione:** `universita/amministrazione/rettore/[username]`
- **Privilegi:** Può leggere tutti i messaggi dell'università

2.2.2 SEGRETERIA

- **Sottoscrizione:** `universita/personale/#`
- **Pubblicazione:** `universita/personale/segreteria/[username]`
- **Privilegi:** Accesso a tutto il personale (docenti e studenti)

2.2.3 DOCENTE

- **Sottoscrizione:** `universita/personale/#`
- **Pubblicazione:** `universita/personale/docenti/[username]`
- **Privilegi:** Può comunicare con altri docenti e studenti

2.2.4 STUDENTE

- **Sottoscrizione:** `universita/personale/studenti/#`
- **Pubblicazione:** `universita/personale/studenti/[username]`
- **Privilegi:** Può comunicare solo con altri studenti

2.3 Flusso dei Messaggi

Mittente	Destinatari	Topic
Studente	Altri Studenti, Docenti, Segreteria, Rettore	universita/personale/studenti/[username]
Docente	Altri Docenti, Segreteria, Rettore	universita/personale/docenti/[username]
Segreteria	Altre Segreterie, Rettore	universita/personale/segreteria/[username]
Rettore	Tutti	universita/amministrazione/rettore/[username]

3 Installazione e Configurazione

3.1 Prerequisiti

- macOS con Homebrew installato
- Compilatore GCC
- Accesso terminale

3.2 Installazione Dipendenze

```
# Installa Mosquitto broker e librerie di sviluppo
brew install mosquitto
brew install mosquitto-dev

# Verifica installazione
mosquitto --help
mosquitto_pub --help
mosquitto_sub --help
```

Listing 2: Installazione Mosquitto

3.3 Struttura del Progetto

```
mqtt-chat/
          mqtt_chat.c          # Codice sorgente principale
          Makefile             # Build automation
          README.md            # Documentazione base
```

Listing 3: Struttura Directory

3.4 Compilazione

```
# Compilazione standard
make

# Compilazione con debug
make debug

# Pulizia file compilati
make clean

# Installazione dipendenze
make install-deps
```

Listing 4: Comandi di Build

4 Guida all'Utilizzo

4.1 Avvio del Sistema

4.1.1 1. Avviare il Broker Mosquitto

```
# Avvio del servizio
make start-broker
# oppure
brew services start mosquitto

# Verifica stato
make status-broker
# oppure
brew services list | grep mosquitto
```

4.1.2 2. Avviare l'Applicazione

```
./mqtt_chat
```

4.2 Interfaccia Utente

4.2.1 Selezione Ruolo

All'avvio, il programma richiede di scegliere il ruolo:

```
=== SCEGLI IL TUO RUOLO ===
1. Rettore (legge tutto)
2. Segreteria (legge personale)
3. Docente (legge personale)
4. Studente (legge solo studenti)
Scelta [1-4]:
```

4.2.2 Comandi Disponibili

Comando	Descrizione
`/help`	Mostra l'elenco dei comandi disponibili
`/quit`	Esce dall'applicazione
`/status`	Mostra lo stato della connessione e il ruolo attivo
`/topic`	Visualizza i topic di sottoscrizione e pubblicazione
`[messaggio]`	Qualsiasi altro testo viene inviato come messaggio

4.3 Scenario di Test

Per replicare uno scenario completo, aprire 6 terminali:

```
# Terminal 1 - Rettore
./mqtt_chat
    Scegli ruolo: 1
    Username: rettore

# Terminal 2 - Segreteria 1
./mqtt_chat
    Scegli ruolo: 2
    Username: segretaria1

# Terminal 3 - Segreteria 2
./mqtt_chat
    Scegli ruolo: 2
    Username: segretaria2

# Terminal 4 - Docente
./mqtt_chat
    Scegli ruolo: 3
    Username: prof_rossi

# Terminal 5 - Studente 1
./mqtt_chat
    Scegli ruolo: 4
```

```
    Username: mario_rossi

# Terminal 6 - Studente 2
./mqtt_chat
    Scegli ruolo: 4
    Username: giulia_bianchi
```

Listing 5: Setup Multi-Utente

5 Analisi del Codice Sorgente

5.1 Strutture Dati Principali

Il file `mqtt_chat.c` definisce le seguenti strutture fondamentali:

5.1.1 Enumerazione Ruoli

```
typedef enum {
    ROLE_RETTORE,
    ROLE_SEGRETERIA,
    ROLE_DOCENTE,
    ROLE_STUDENTE
} user_role_t;
```

Listing 6: Definizione Ruoli Utente

5.1.2 Struttura Client Chat

```
typedef struct {
    struct mosquitto *mosq;              // Istanza client Mosquitto
    user_role_t role;                    // Ruolo dell'utente
    char username[MAX_USERNAME_LEN];     // Nome utente
    char subscribe_topic[MAX_TOPIC_LEN]; // Topic sottoscrizione
    char publish_base_topic[MAX_TOPIC_LEN]; // Topic pubblicazione base
    int is_connected;                    // Stato connessione
} mqtt_chat_client_t;
```

Listing 7: Struttura Client MQTT

5.2 Funzioni Callback

5.2.1 Callback di Connessione

```
void on_connect(struct mosquitto *mosq, void *userdata, int result) {
    mqtt_chat_client_t *client = (mqtt_chat_client_t *)userdata;

    if (result == 0) {
        printf("    Connesso al broker MQTT come %s\n", client->username
    );
        client->is_connected = 1;

        // Subscribe al topic appropriato
```

```
        mosquitto_subscribe(mosq, NULL, client->subscribe_topic, 0);
        printf("     Sottoscritto al topic: %s\n", client->
    subscribe_topic);
    } else {
        printf("    Errore di connessione: %d\n", result);
    }
}
```

Listing 8: Gestione Connessione

Questa funzione:

- Gestisce il risultato della connessione al broker
- Effettua automaticamente la sottoscrizione al topic appropriato
- Aggiorna lo stato di connessione del client
- Fornisce feedback visivo all'utente

5.2.2 Callback di Messaggio

```
void on_message(struct mosquitto *mosq, void *userdata,
                const struct mosquitto_message *message) {
    mqtt_chat_client_t *client = (mqtt_chat_client_t *)userdata;

    if (message->payloadlen) {
        // Ottieni timestamp
        time_t now = time(NULL);
        struct tm *tm_info = localtime(&now);
        char timestamp[20];
        strftime(timestamp, sizeof(timestamp), "%H:%M:%S", tm_info);

        // Stampa il messaggio ricevuto
        printf("\ n    [%s] Topic: %s\n", timestamp, message->topic);
        printf("   Messaggio: %s\n", (char *)message->payload);
        printf(">> ");
        fflush(stdout);
    }
}
```

Listing 9: Ricezione Messaggi

Funzionalità implementate:

- Riceve e processa i messaggi in arrivo
- Formatta l'output con timestamp e informazioni topic
- Gestisce la visualizzazione asincrona dei messaggi
- Mantiene il prompt utente sempre visibile

5.3 Inizializzazione Client

```
void init_client_by_role(mqtt_chat_client_t *client, user_role_t role,
                         const char *username) {
    client->role = role;
    strncpy(client->username, username, MAX_USERNAME_LEN - 1);
    client->is_connected = 0;

    switch (role) {
        case ROLE_RETTORE:
            strcpy(client->subscribe_topic, "universita/#");
            strcpy(client->publish_base_topic, "universita/
    amministrazione/rettore");
            break;

        case ROLE_SEGRETERIA:
            strcpy(client->subscribe_topic, "universita/personale/#");
            strcpy(client->publish_base_topic, "universita/personale/
    segreteria");
            break;

        case ROLE_DOCENTE:
            strcpy(client->subscribe_topic, "universita/personale/#");
            strcpy(client->publish_base_topic, "universita/personale/
    docenti");
            break;

        case ROLE_STUDENTE:
            strcpy(client->subscribe_topic, "universita/personale/
    studenti/#");
            strcpy(client->publish_base_topic, "universita/personale/
    studenti");
            break;
    }
}
```

Listing 10: Configurazione Ruolo

Questa funzione implementa la logica di controllo accesso configurando i topic appropriati per ogni ruolo secondo la gerarchia definita.

5.4 Funzione di Pubblicazione

```
void publish_message(mqtt_chat_client_t *client, const char *message) {
    if (!client->is_connected) {
        printf("    Non connesso al broker!\n");
        return;
    }

    char full_topic[MAX_TOPIC_LEN];
    char full_message[MAX_MESSAGE_LEN];

    // Crea il topic completo con l'username
    snprintf(full_topic, sizeof(full_topic), "%s/%s",
             client->publish_base_topic, client->username);

    // Crea il messaggio completo con nome utente
```

```
    snprintf(full_message, sizeof(full_message), "%s: %s",
             client->username, message);

    int result = mosquitto_publish(client->mosq, NULL, full_topic,
                                   strlen(full_message), full_message, 0,
    false);

    if (result == MOSQ_ERR_SUCCESS) {
        printf("     Messaggio inviato!\n");
    } else {
        printf("    Errore invio messaggio: %d\n", result);
    }
}
```

Listing 11: Invio Messaggi

Caratteristiche implementative:

- Verifica stato connessione prima dell'invio
- Costruisce topic completo includendo username
- Formatta il messaggio con identificazione mittente
- Gestisce errori di pubblicazione con feedback utente
- Utilizza QoS 0 per massime performance

5.5 Loop Principale

Il `main()` implementa un ciclo di gestione input che:

- Inizializza la libreria Mosquitto
- Gestisce selezione ruolo e username
- Configura callback e connessione
- Avvia loop network in background
- Processa comandi utente in loop sincrono
- Gestisce cleanup e disconnessione pulita

6 Analisi del Makefile

Il Makefile fornisce un sistema di build automation completo:

6.1 Variabili di Configurazione

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99 -pedantic
LIBS = -lmosquitto
TARGET = mqtt_chat
SOURCE = mqtt_chat.c
```

Listing 12: Configurazione Build

6.2 Target Principali

6.2.1 Compilazione

```
$(TARGET): $(SOURCE)
  $(CC) $(CFLAGS) -o $(TARGET) $(SOURCE) $(LIBS)

debug: CFLAGS += -g -DDEBUG
debug: $(TARGET)
```

6.2.2 Gestione Dipendenze

```
install-deps:
  @echo "Installazione dipendenze per macOS..."
  brew install mosquitto
  brew install mosquitto-dev
```

6.2.3 Gestione Broker

```
start-broker:
  @echo "Avvio broker Mosquitto..."
  brew services start mosquitto

stop-broker:
  @echo "Arresto broker Mosquitto..."
  brew services stop mosquitto

status-broker:
  @echo "Stato broker Mosquitto:"
  brew services list | grep mosquitto
```

Il Makefile automatizza completamente il workflow di sviluppo, dalla gestione delle dipendenze al controllo del broker MQTT.

7 Testing e Debug

7.1 Analisi con Wireshark

7.1.1 Filtri Utili

Filtro	**Descrizione**
`mqtt`	Mostra tutto il traffico MQTT
`mqtt.topic contains "chat"`	Filtra per topic contenenti "chat"
`mqtt and ip.addr == localhost`	Traffico MQTT verso/da localhost
`mqtt.msgtype == 3`	Solo messaggi PUBLISH

7.1.2 Osservazioni sul Traffico

- **Keep Alive:** Pacchetti PING regolari per mantenere connessione NAT
- **Publisher Pattern:** Connessione → Pubblicazione → Disconnessione
- **Subscriber Pattern:** Connessione persistente con keep-alive
- **QoS Levels:** Il sistema usa QoS 0 (at most once delivery)

7.2 Debug e Logging

7.2.1 Compilazione Debug

```
make debug
```

7.2.2 Debugging con GDB

```
gdb ./mqtt_chat
(gdb) run
(gdb) break on_message
(gdb) continue
```

7.3 Test di Funzionalità

7.3.1 Test di Connessione

1. Avviare il broker Mosquitto
2. Lanciare il client
3. Verificare messaggio "Connesso al broker MQTT"
4. Controllare sottoscrizione automatica

7.3.2 Test di Comunicazione

1. Avviare due istanze con ruoli compatibili
2. Inviare messaggio dalla prima istanza
3. Verificare ricezione nella seconda istanza
4. Controllare timestamp e formattazione

8 Considerazioni sulla Sicurezza

8.1 Implementazioni di Sicurezza Attuali

- **Controllo Accesso Logico:** Basato su topic hierarchy
- **Isolamento Ruoli:** Ogni ruolo ha permessi specifici
- **Validazione Input:** Controllo lunghezza messaggi e topic
- **Gestione Errori:** Handling sicuro delle connessioni

8.2 Vulnerabilità Identificate

Attenzione

Messaggi in Chiaro
I messaggi viaggiano non crittografati. Un attaccante con accesso alla rete può intercettare e leggere tutti i contenuti.
Mancanza di Autenticazione
Non c'è verifica dell'identità degli utenti. Chiunque può assumere qualsiasi ruolo.
Topic Spoofing
Un client può potenzialmente pubblicare su topic non autorizzati modificando il codice.

8.3 Miglioramenti Suggeriti

1. **TLS/SSL:** Crittografia del trasporto
2. **Autenticazione:** Username/password o certificati
3. **ACL Broker:** Controllo accesso server-side
4. **Message Encryption:** Crittografia end-to-end
5. **Rate Limiting:** Prevenzione spam/DoS
6. **Audit Logging:** Log delle attività utente

9 Risoluzione Problemi

9.1 Problemi Comuni

9.1.1 Errore: "Broker non raggiungibile"

```
# Verifica se Mosquitto    attivo
brew services list | grep mosquitto

# Se non attivo, avvialo
brew services start mosquitto

# Testa connessione manuale
```

```
mosquitto_sub -h localhost -t test/topic
```

9.1.2 Errore di Compilazione: "mosquitto.h not found"

```
# Installa librerie di sviluppo
brew install mosquitto-dev

# Verifica path include
pkg-config --cflags libmosquitto
```

9.1.3 Messaggi Non Ricevuti

1. Verifica topic di sottoscrizione con `/topic`
2. Controlla che i ruoli abbiano permessi compatibili
3. Testa con client da linea di comando:

```
mosquitto_sub -h localhost -t "universita/#"

```

9.2 Diagnostica Avanzata

9.2.1 Log del Broker

```
# Avvia broker in modalit  verbose
mosquitto -v

# Log su file
mosquitto -v > mosquitto.log 2>&1
```

9.2.2 Monitor Traffico di Rete

```
# Con netstat
netstat -an | grep 1883

# Con lsof
lsof -i :1883
```

Informazione

Suggerimento per il Debug
Usa sempre Wireshark per analizzare il traffico MQTT. È il modo più efficace per capire cosa succede a livello di protocollo e identificare problemi di comunicazione.

10 Possibili Estensioni

10.1 Funzionalità Avanzate

- **Interfaccia Grafica:** GUI con GTK+ o Qt

- **Persistenza Messaggi:** Database SQLite per storico chat
- **Notifiche Push:** Integrazione con sistemi di notifica OS
- **File Transfer:** Invio di allegati tramite MQTT
- **Presenza Utenti:** Indicator online/offline
- **Gruppi Privati:** Chat room tematiche

10.2 Integrazione Enterprise

- **LDAP Integration:** Autenticazione aziendale
- **SSO Support:** Single Sign-On
- **Compliance:** GDPR, logging audit
- **Scalabilità:** Clustering broker, load balancing

11 Conclusioni

Il progetto Chat MQTT Universitaria rappresenta un'implementazione completa e funzionale del paradigma publish/subscribe utilizzando il protocollo MQTT. L'architettura basata sulla gerarchia di topic permette un controllo granulare dei permessi di accesso, simulando realisticamente l'ambiente comunicativo di un'istituzione universitaria.

11.1 Obiettivi Raggiunti

- Implementazione completa del pattern PUB/SUB
- Sistema di ruoli e permessi funzionante
- Interfaccia utente intuitiva e interattiva
- Gestione robusta delle connessioni e disconnessioni
- Analisi del traffico di rete tramite Wireshark integrata
- Build system automatizzato con Makefile completo

11.2 Lezioni Apprese

- **Paradigma PUB/SUB:** Efficacia nella disaccoppiamento dei componenti
- **Topic Hierarchy:** Importanza cruciale nella progettazione dei permessi
- **Gestione Asincrona:** Complessità nella sincronizzazione I/O e UI
- **Error Handling:** Necessità di gestione robusta degli stati di errore
- **Debugging Distribuito:** Utilità fondamentale degli strumenti di analisi di rete

11.3 Limitazioni Identificate

- Mancanza di persistenza dei messaggi offline
- Assenza di crittografia end-to-end
- Controllo accesso basato solo su logica client-side
- Scalabilità limitata per grandi numeri di utenti
- Gestione basilare degli errori di rete

11.4 Applicabilità Reale

Il sistema implementato fornisce una base solida per applicazioni IoT e sistemi di messaging distribuiti. Le competenze acquisite sono direttamente trasferibili a:

- Sistemi di monitoraggio industriale
- Applicazioni IoT smart home
- Piattaforme di telemetria veicolare
- Sistemi di notifica enterprise
- Architetture microservizi event-driven

12 Appendici

12.1 Appendice A: Codice Completo

Il codice sorgente completo è disponibile nei file del progetto. Le sezioni principali implementano:

- Inizializzazione e configurazione client MQTT
- Gestione callback per eventi di rete
- Loop principale con input utente non-bloccante
- Sistema di comandi interattivi
- Gestione sicura della memoria e cleanup

12.2 Appendice B: Configurazioni Avanzate

12.2.1 Configurazione Broker Mosquitto

Per ambienti di produzione, modificare il file `/opt/homebrew/etc/mosquitto/mosquitto.conf`:

```
# Porta di ascolto
port 1883

# Autenticazione
allow_anonymous false
password_file /opt/homebrew/etc/mosquitto/passwd

# ACL (Access Control List)
acl_file /opt/homebrew/etc/mosquitto/acl

# Persistenza
persistence true
persistence_location /opt/homebrew/var/lib/mosquitto/

# Log
log_dest file /opt/homebrew/var/log/mosquitto/mosquitto.log
log_type all

# Security
max_connections 1000
max_inflight_messages 20
max_queued_messages 1000
```

Listing 13: mosquitto.conf Avanzato

12.2.2 Setup ACL di Produzione

```
# ACL per ruoli universitari
user rettore
topic readwrite universita/#

user segreteria_#
topic readwrite universita/personale/#

user docente_#
topic readwrite universita/personale/docenti/#
topic read universita/personale/studenti/#

user studente_#
topic readwrite universita/personale/studenti/#
```

Listing 14: File ACL Esempio

12.3 Appendice C: Script di Deployment

```
#!/bin/bash
# deploy.sh - Script di deployment automatico

set -e

echo "=== Deploy Chat MQTT Universitaria ==="

# Verifica prerequisiti
check_dependencies() {
    echo "Verifica dipendenze..."
```

```
    command -v gcc >/dev/null 2>&1 || { echo "GCC richiesto"; exit 1; }
    command -v mosquitto >/dev/null 2>&1 || { echo "Mosquitto richiesto"
    ; exit 1; }
    echo "    Dipendenze verificate"
}

# Compilazione
build_project() {
    echo "Compilazione progetto..."
    make clean
    make
    echo "    Compilazione completata"
}

# Setup servizi
setup_services() {
    echo "Configurazione servizi..."
    brew services start mosquitto
    echo "    Broker MQTT avviato"
}

# Test sistema
run_tests() {
    echo "Esecuzione test..."
    # Test connessione broker
    timeout 5 mosquitto_pub -h localhost -t test -m "test" || {
        echo "    Test broker fallito"
        exit 1
    }
    echo "    Test completati"
}

# Main
main() {
    check_dependencies
    build_project
    setup_services
    run_tests
    echo "     Deploy completato con successo!"
}

main "$@"
```

Listing 15: Deploy Script

12.4 Appendice D: Metriche e Monitoring

12.4.1 Script di Monitoraggio

```
#!/bin/bash
# monitor.sh - Monitoraggio sistema MQTT

while true; do
    echo "=== MQTT System Status $(date) ==="

    # Status broker
    echo "Broker Status:"
```

```
    brew services list | grep mosquitto

    # Connessioni attive
    echo "Active Connections:"
    netstat -an | grep 1883 | wc -l

    # Utilizzo risorse
    echo "Resource Usage:"
    ps aux | grep mosquitto | grep -v grep

    sleep 30
done
```

Listing 16: Monitoring Script

13 Bibliografia e Riferimenti

13.1 Documentazione Tecnica

- Eclipse Mosquitto Documentation: `https://mosquitto.org/documentation/`
- MQTT Protocol Specification v5.0: `https://docs.oasis-open.org/mqtt/mqtt/v5.0/`
- Eclipse Paho C Client Library: `https://github.com/eclipse/paho.mqtt.c`
- Wireshark MQTT Dissector Guide: `https://wiki.wireshark.org/MQTT`

13.2 Risorse Aggiuntive

- HiveMQ MQTT Essentials: `https://www.hivemq.com/mqtt-essentials/`
- MQTT Security Best Practices: `https://www.hivemq.com/blog/mqtt-security-fundamentals`
- IoT Communication Protocols Comparison
- Real-time Messaging Patterns in Distributed Systems

13.3 Standard e RFC

- RFC 6455 - The WebSocket Protocol
- RFC 7693 - The BLAKE2 Cryptographic Hash and Message Authentication Code
- ISO/IEC 20922:2016 - Information technology – Message Queuing Telemetry Transport (MQTT)

Informazione

Nota Finale
Questa documentazione rappresenta una guida completa per l'implementazione, il deployment e la manutenzione del sistema Chat MQTT Universitaria. Per aggiornamenti e contributi al progetto, consultare il repository Git associato.

Fine Documentazione

Chat MQTT Universitaria v1.0