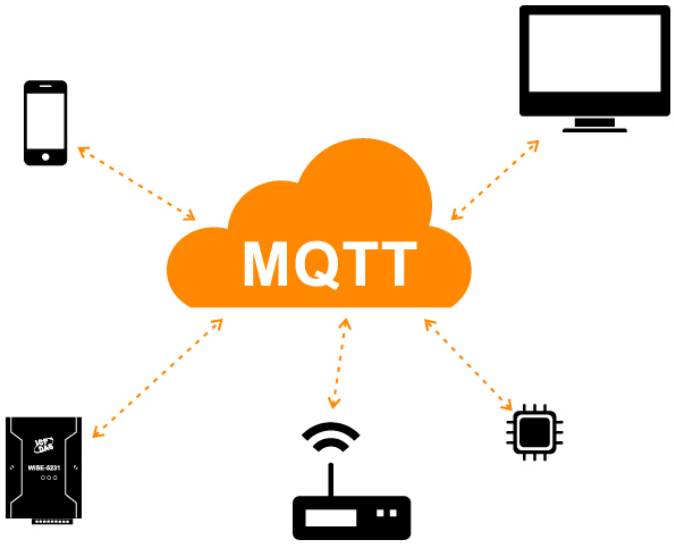
Universitatea Tehnică "Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare
Rețele de Calculatoare - Proiect

Client MQTT - Documentație

Baltariu Ionuț-Alexandru
Grupa 1305A

Bejenariu Răzvan-Marius
Grupa 1306A



1 Scurtă descriere:

MQTT (**Message Queue Telemetry Transport**) este un standard de comunicație prin internet, inventat și dezvoltat de IBM in 1999, care defineste un protocol de transport al mesajelor prin internet (prin TCP/IP în general), între un server ("message broker") și mai mulți clienți, pe modelul "publicare si abonare" ("**publish and subscribe**").

Este un protocol de conectivitate "mașină - mașină" **(M2M) / "Internet of Things**", conceput a fi simplu și flexibil, cu cerințe minime de resurse, util pentru conexiunile cu locații la distanță și perfect pentru toate tipurile de aplicații IoT, fie că sunt automatizări industriale (**IIot**) sau automatizări casnice/rezidențiale/de consum (**Consumer Io**t).

Este, de asemenea ideal și pentru **aplicatii mobile**, datorită dimensiunilor mici, consumului redus de energie, pachetelor de date minimizate și distribuției eficiente a informației către unul sau mai mulți receptori.

2 Cerințe

1. **2 view-uri: unul pentru publicare și altul pentru abonare**

   În momentul în care un client se conectează, acesta se poate abona la topicurile de care este interesat, pentru a primi informații de la toate dispozitivele care furnizează acea informație sau poate publica diferite topicuri brokerului pentru ca acesta să transmită informația la rândul său.

2. **Autentificare cu utilizator și parolă**

   Acest feature va fi posibil prin trimiterea informațiilor respective într-un pachet folosit la conectare. Brokerul va gestiona întregul proces de autentificare la acesta de către publisheri.

3. **Listă de abonare configurabilă**:

   Vom implementa un mecanism prin care clientul să poată să se aboneze/dezaboneze de la anumite topicuri. Acest lucru va fi posibil prin modificarea unui fișier de configurări sau direct prin interfața grafică a aplicației.

4. **Publicare manuală sau automată**:

   Publisherul va avea posibilitatea de a posta informatii atât la cerere, cât și automat la un interval de timp (configurabil), pentru a mentine alți utilizatori informați.

5. **Mecanism KeepAlive**:

   Mecanismul KeepAlive se utilizează pentru a verifica dacă un client este conectat la broker, ambele părți știind de existența celeilalte. În momentul în care clientul se conectează la broker, aceștia pot sta maxim o anumită perioadă fară a interacționa unul cu celălalt.

   Dacă această perioadă este depășită clientul trebuie să trimită un pachet **PINGREQ** brokerului, la care brokerul va trimite la rândul său un pachet **PINGRESP**, iar dacă aceste pachete nu ajung într-un anumit interval de timp, clientul va fi deconectat de la broker.

6. **Implementare QoS 0,1,2**:

   QoS(Quality of Service) se referă la conexiunea dintre client și broker, și reprezintă o modalitate de garantare a livrării mesajului.

   QoS 0: rapid, dar nesigur, mesajul nu este salvat nicăieri, deci, mesajul este pierdut în cazul în care informația nu ajunge la broker.

   QoS 1: mesajele se pot trimite de mai multe ori,chiar dacă nu este nevoie, după trimiterea informației către broker, se așteaptă o confirmare de la acesta(**PUBACK**), în cazul în care clientul primește confirmarea, informația este ștearsă din coadă, în caz contrar, mesajul se trimite până se primește o confirmare de primire de la broker.

   QoS 2: ne garantează livrarea mesajului o singură dată, chiar dacă operațiunea este mai lentă. Se petrece același lucru ca și la QoS 1 cu diferența că se mai întamplă cateva verificări, înainte ca brokerul să trimită informația mai departe.

7. **Mecanismul Last Will**:

   Este o modalitate de a-i înștiința pe ceilalți clienți în momentul în care cineva se deconectează într-un mod neașteptat. Fiecare client își poate specifica mesajul ce va fii trimis în momentul deconectării de către broker, mesaj, ce va fii trimis tuturor clienților abonați la acel topic.În cazul deconectării intenționate apare **MQTT Disconnect**.

3 Implementare - generalități

3.1 Reprezentarea datelor

Tip de dată	Reprezentarea în cadrul MQTT
Întreg - 2 Octeți (**short int**)	Big Endian: Octetul 1 MSB; Octetul 2 LSB
Întreg - 4 Octeți (**int**)	Big Endian: $3 \times MSB$; $1 \times LSB$
String	Lungimea pe 2 Octeți(MSB & LSB) + String codat UTF-8
Date binare	Lungimea pe 2 Octeți(MSB & LSB) + Octeții corespunzători datelor

3.2 Structura pachetelor de control

3.2.1 Fixed Header

Fiecare pachet are un **Fixed Header**, format din:

- **Byte 1**: primii 4 biți pentru **tipul pachetului**
- **Byte 1**: ultimii 4 biți pentru **flaguri** specifice pachetelor
- **Byte 2 -> Byte N**: Remaining Length - **dimensiunea** (în Octeți) câmpurilor următoare

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Figure 1: Structură Fixed Header

3.2.2 Variable Header

Majoritatea pachetelor au un **Variable Header**, mai puțin PINGREQ și PINGRESP, situat între Fixed Header și Payload, acesta fiind diferit de la pachet la pachet.

	Description	7	6	5	4	3	2	1	0
Topic Name									
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
Packet Identifier									
byte 6	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 7	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0
Property Length									
byte 8	No Properties	0	0	0	0	0	0	0	0

Figure 2: Exemplu Variable Header (PUBLISH)

3.2.3 Payload

În acest câmp sunt introduse datele care trebuie să fie trimise efectiv către server. Un exemplu ar fi, pentru pachetul CONNECT, username-ul (atât timp cât flag-ul pentru username este pus pe 1).

Precum Variable Header, câmpul Payload nu este prezent la toate pachetele, ci doar la 6 dintre ele (în cadrul unui pachet fiind chiar opțional).

Description	7	6	5	4	3	2	1	0
Topic Filter								
byte 1	Length MSB							
byte 2	Length LSB							
bytes 3..N	Topic Filter							
Subscription Options								
	Reserved		Retain Handling		RAP	NL	QoS	
byte N+1	0	0	X	X	X	X	X	X

Figure 3: Exemplu Structură Payload (SUBSCRIBE)

3.2.4 Reason Code

Este un octet care indică rezultatul operației.

În cazul pachetelor SUBACK sau UNSUBACK pot exista mai mulți octeți care să conțină Reason Codes.

3.3 Pachetele de control

În **MQTT**, versiunea **5.0**, există 15 pachete de control. Acestea sunt ilustrate în **Figure 4**.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

Figure 4: Pachetele MQTT

În cadrul aplicației, acestea sunt reprezentate de către o **clasă**, în care găsim metoda de ”creare” efectivă a pachetului (serializare) - pack.

Datele din Variable Header și/sau Payload vor fi introduse în pachet prin intermediul constructorului ($\_\_init\_\_$) sau al altor metode auxiliare, iar conținutul pachetului va fi serializat într-un $bytearray$ pentru a putea fi trimis către un server.

3.4 Clientul MQTT

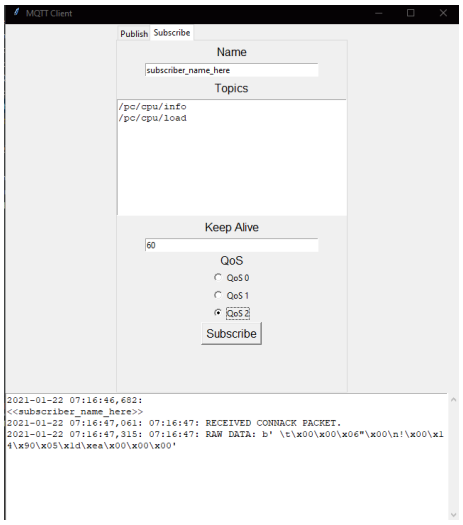
Clientul a fost implementat folosind 3 thread-uri, după cum urmează:

1. **Receiver Thread:** folosit pentru a primi mesaje de la brokerul **mosquitto**
2. **Sender Thread:** folosit pentru a trimite pachete către broker
3. **Keep Alive Thread:** dedicat asigurării persistenței comunicației dintre client și broker

Pentru comunicarea efectivă dintre client(publisher/subscriber) și broker s-a folosit modulul **Socket** din Python, cu ajutorul căruia s-au creat socket-uri TCP pe IP-ul **127.0.0.1** și portul **1883**, specific MQTT.

Clientul poate fi folosit ca **publisher** sau **subscriber**, fără posibilitatea rulării în mod hibrid, și trebuie să se **autentifice** înainte de a putea face orice operațiune folosind date cunoscute de către broker.

(a) Publisher GUI



(b) Subscriber GUI

Figure 5: View-uri

4 Implementarea cerințelor

4.1 Două view-uri: unul pentru publicare, unul pentru abonare

Pentru această parte s-a folosit modulul **Tkinter**, care permite programarea interfețelor grafice.Rezultatul se poate vedea în **Figura 5**.

4.2 Autentificare cu utilizator și parolă

Pentru acest aspect s-a luat în calcul structura pachetului **CONNECT**, în care trebuiau setate pe 1 flag-urile pentru User Name și pentru Password.

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Start	Reserved
byte 8	X	X	X	X	X	X	X	0

Figure 6: Flag-uri CONNECT

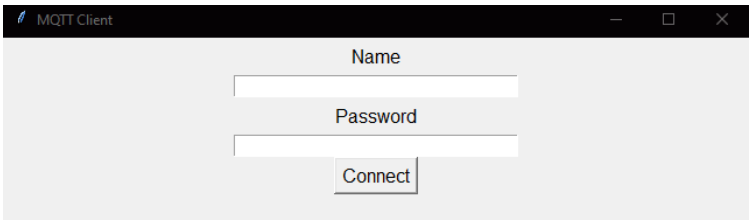


Figure 7: Autentificarea în aplicație

A trebuit, de asemenea, să se facă și niște setări în ceea ce privește broker-ul mosquitto, mai exact:

1. **allow_anonymous false**
2. **password_file [path către fișierul cu parole]**

Pentru criptarea parolelor s-a folosit **mosquitto_passwd**: mosquitto_passwd -U fisier_cu_parole.txt

4.3 Listă de abonare configurabilă (creare/stergere din GUI, fisier config)

Pentru această funcționalitate s-a folosit un widget **Text** din **Tkinter** care permite introducerea unui text pe mai multe linii. Astfel, un utilizator poate introduce topicurile dorite a fi urmărite separate de către o linie nouă.

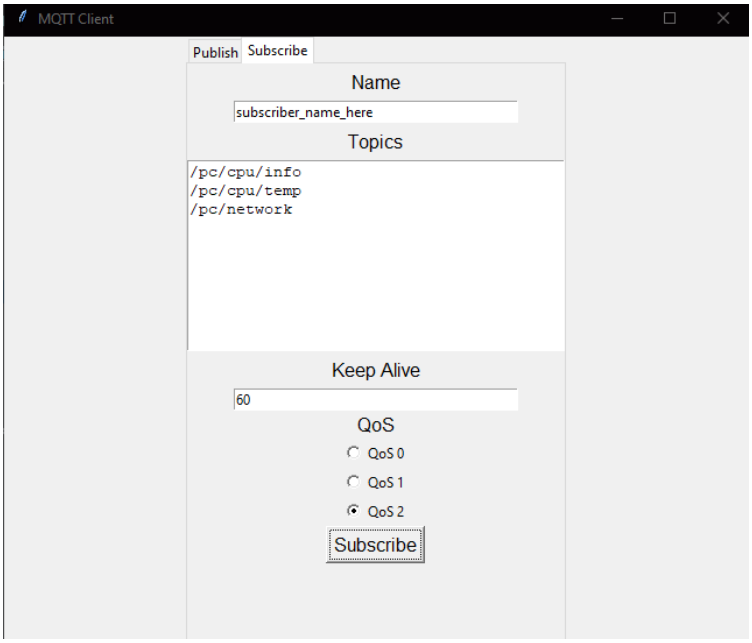


Figure 8: View-ul subscriberului, lista configurabilă

4.4 Publicare manuală (din GUI) sau automată (periodic, configurabil) pentru valorile monitorizate

Practic, publicarea automată a însemnat trimiterea periodică a unui mesaj pe **Sender Thread**, folosindu-ne de **time.sleep** pentru a asigura delay-ul dintre publicări.

```
def run(self):
    global g_manual_publish_flag
    if self.tip == 2://PUBLICARE AUTOMATĂ
        if self.qos == 1 or self.qos == 0:
            while self.running:
                temporary_payload = determine_payload(self.topic)
                self.publish_packet = PublishPacket(self.topic, temporary_payload,
                                                    self.generated_client_id, self.qos)
                self.send(self.publish_packet.pack())
                sleep(self.interval)
```

Figure 9: Codul aferent publicării automate

Cât despre publicarea manuală, s-a folosit o variabilă care se incrementează de atâtea ori cât este apăsat butonul de **PUBLISH**, urmând ca **Sender Thread** să trimite pachetele corespunzătoare până când variabila respectivă ajunge la valoarea 0.

```
if self.qos == 1 or self.qos == 0:
    while self.running:
        while g_manual_publish_flag != 0:
            temporary_payload = determine_payload(self.topic)
            self.publish_packet = PublishPacket(self.topic, temporary_payload, self.generated_client_id,
                                                self.qos)
            self.send(self.publish_packet.pack())
            g_manual_publish_flag -= 1
```

Figure 10: Codul aferent publicării manuale

4.5 Implementare mecanism KeepAlive

S-a folosit un thread separat, menționat anterior, al cărui singur rol este cel de a trimite un pachet **PINGREQ** fix înainte de eventualul timeout care poate fi primit de la broker (care se întâmplă când nu au mai fost trimise pachete pentru $1.5 \times Keep\_Alive$ secunde.

```
def client_keep_alive(self):
    while self.running:
        sleep(int(self.keep_alive * 1.4))
        self.send(PingreqPacket().pack())
```

Figure 11: Codul aferent funcționalității keep alive

4.6 Implementare QoS 0,1,2

S-a ținut cont, atât pe **Sender Thread** cât și pe **Receiver Thread** de QoS-ul selectat din GUI și s-a folosit o variabilă care memora ultimul pachet trimis, pentru a facilita schimbul de pachete specifice QoS 1/2.

```
if self.qos == 2:
    while self.running:
        temporary_payload = determine_payload(self.topic)
        self.publish_packet = PublishPacket(self.topic, temporary_payload, self.generated_client_id,
                                            self.qos)
        if self.last_received_packet == "CONNACK" or self.last_received_packet == "PUBCOMP":
            self.send(self.publish_packet.pack())
        if self.last_received_packet == "PUBREC":
            self.send(PubrelPacket(self.generated_client_id).pack())
        sleep(self.interval)
```

Figure 12: QoS2 pentru Publisher

```
elif self.qos_level == 2:
    while self.running:
        if self.last_received_packet == "PUBLISH":
            self.send(PubrecPacket(g_generated_client_id).pack())
            self.last_received_packet = ""
        elif self.last_received_packet == "PUBREL":
            self.send(PubcompPacket(g_generated_client_id).pack())
            self.last_received_packet = ""
```

Figure 13: QoS2 pentru Subscriber

4.7 Implementare mecanism LastWill

Pentru a preciza brokerului faptul că există un mesaj pentru deconectări neprevăzute (Last Will Message) s-a modificat din nou structura pachetului **CONNECT**, setându-se **Will Flag** pe 1 (Will QoS a fost lăsat pe 0 pentru simplitate).

Resurse

1. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html
2. https://www.hivemq.com/docs/hivemq/4.4/user-guide/introduction.html
3. https://stackoverflow.com/questions/65159409/mosquitto-broker-gives-error-when-receiving-mqtt-connect-packet-python
4. https://stackoverflow.com/questions/65450457/mosquitto-mqtt-broker-wont-send-more-than-20-publish-packets-to-subscriber
5. https://stackoverflow.com/questions/65475163/mosquitto-broker-stops-handling-publisher-and-subscriber-after-receiving-pingreq