



UNIVERSITATEA TEHNICĂ "GH ASACHI" IAȘI
FACULTATEA AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

DISCIPLINA REțele DE CALCULATOARE PROIECT

Message Queuing Telemetry Transport MQTT server

Coordonator,

Ș.l.dr.ing. Nicolae-Alexandru Botezatu

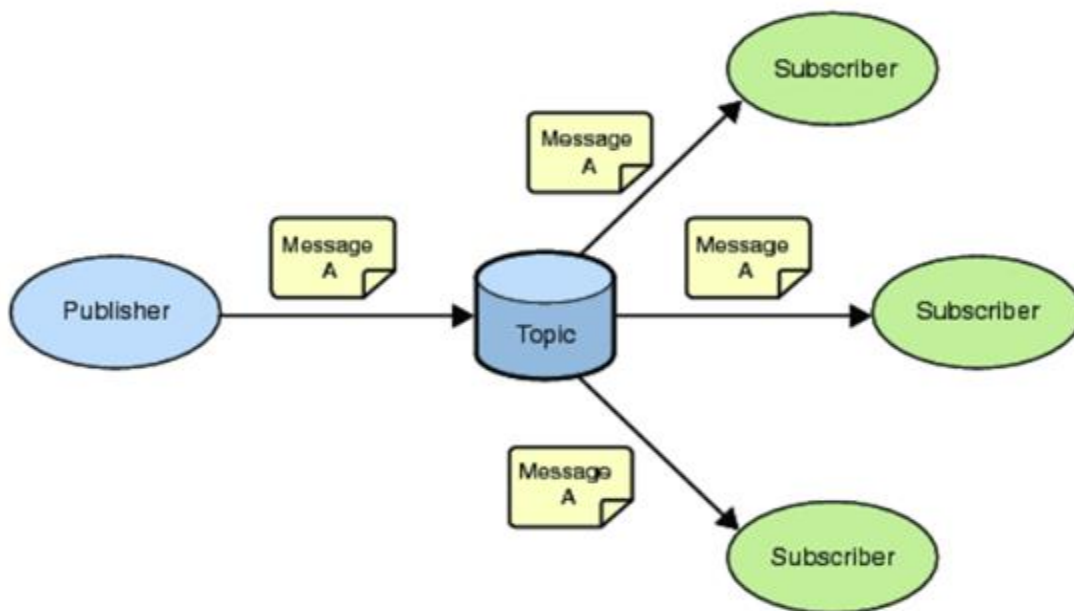
Studenti,

Ichim Paula-Madalina
Sopca Stefania

Mqtt este un standard de comunicare prin internet care definește unul dintre cele mai importante protocoale de transport al mesajelor prin internet între server și clienți. Acesta a fost inventat și dezvoltat în 1999 de IBM drept un protocol simplu și flexibil, necesitând un minim de resurse, util pentru conexiunile la distanță și aplicațiile ce fac parte din “Internet of Things”, fie că sunt automatizări de consum (Consumer **IoT**) sau automatizări industriale (IIoT).

În 2014 a fost declarat ca standard de OASIS (Organization for the Advancement of Structured Information Standards), iar în 2016 ca standard ISO (International Organization of Standardization).

Modelul descris de **MQTT** se bazează pe conexiunea dintre server (“broker”) și client, acesta din urmă își publică mesajele cu o etichetă numită “topic”. De asemenea, alți clienți se abonează, ”subscribe” la server la anumite topicuri. Brokerul are misiunea de a trimite mesaje unui anumit topic clienților ce s-au abonat la acel topic.



Un protocol de mesagerie “Publisher- Subscriber” permite publicarea unui mesaj o singură dată și primirea acestuia de mai mulți consumatori (aplicații /dispozitive) oferind decuplarea între producător și consumatori.

Informațiile sunt organizate într-o ierarhie de topicuri. Atunci când un editor(publisher) are un nou element de date de distribuit, acesta trimite un mesaj de control cu datele către brokerul conectat. Brokerul distribuie apoi informațiile tuturor clienților care s-au abonat(“make a subscription”) la acel topic.

Astfel un server / broker de mesaje “matches publications to subscriptions” în următorul mod:

- Dacă nu se potrivește, mesajul este eliminat
- Dacă este una sau mai multe potriviri, mesajul este livrat fiecărui abonat / consumator.

Publisherul nu trebuie să dețină date despre numărul sau locațiile abonaților, iar abonații, la rândul lor, nu trebuie să fie configurați cu niciun fel de date despre publisheri.

Un abonament („a subscription”) poate fi durabil sau nedurabil:

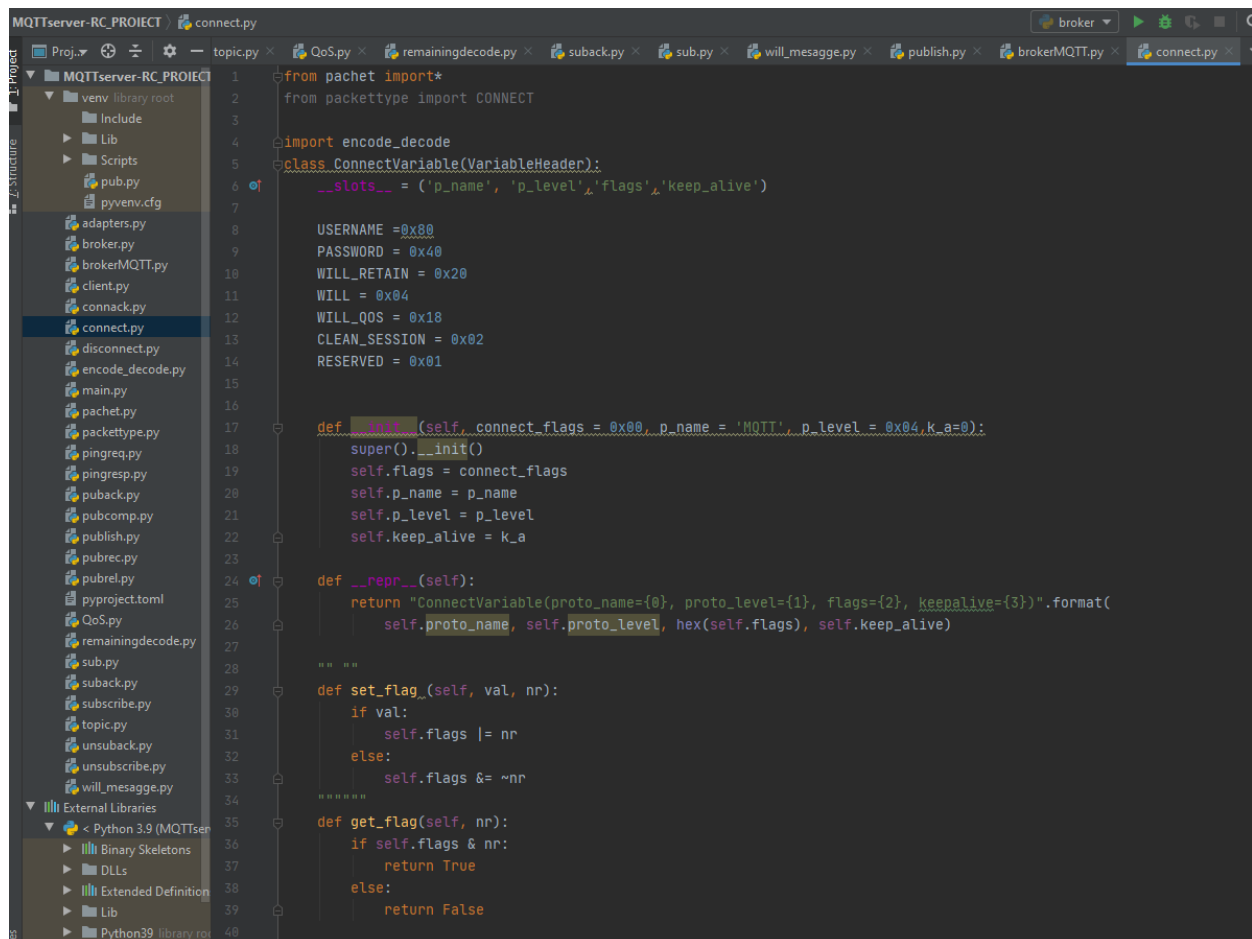
- Durabil: Conform abonamentului (subscription), un broker va transmite mesajele potrivite către abonat, imediat dacă abonatul

este conectat, dacă abonatul nu este conectat, mesajele sunt stocate pe server / broker până la următoarea dată în care abonatul se conectează.

- Nedurabil: durata de viață a abonamentului (subscription) este aceeași cu cea a abonatului conectat la server / broker.

Există patru pachete de control MQTT primare pe care un client și un server le pot folosi pentru a comunica:

- Conectare (“Connect”)



```
1 from packet import*
2 from packettype import CONNECT
3
4 import encode_decode
5 class ConnectVariable(VariableHeader):
6     __slots__ = ('p_name', 'p_level', 'flags', 'keep_alive')
7
8     USERNAME = 0x80
9     PASSWORD = 0x40
10    WILL_RETAIN = 0x20
11    WILL = 0x04
12    WILL_QOS = 0x18
13    CLEAN_SESSION = 0x02
14    RESERVED = 0x01
15
16
17    def __init__(self, connect_flags = 0x00, p_name = 'MQTT', p_level = 0x04, k_a=0):
18        super().__init__()
19        self.flags = connect_flags
20        self.p_name = p_name
21        self.p_level = p_level
22        self.keep_alive = k_a
23
24    def __repr__(self):
25        return "ConnectVariable(proto_name={0}, proto_level={1}, flags={2}, keepalive={3}).format(
26            self.proto_name, self.proto_level, hex(self.flags), self.keep_alive)
27
28    == ==
29    def set_flag(self, val, nr):
30        if val:
31            self.flags |= nr
32        else:
33            self.flags &= ~nr
34
35    *****
36    def get_flag(self, nr):
37        if self.flags & nr:
38            return True
39        else:
40            return False
```

— Primul pachet trimis de la client la server trebuie să fie un pachet de conectare pentru a stabili o conexiune.

Serverul TREBUIE să valideze faptul că “reserved flag” din pachetul de control CONNECT este setat la zero astfel deconectează clientul dacă nu este zero.

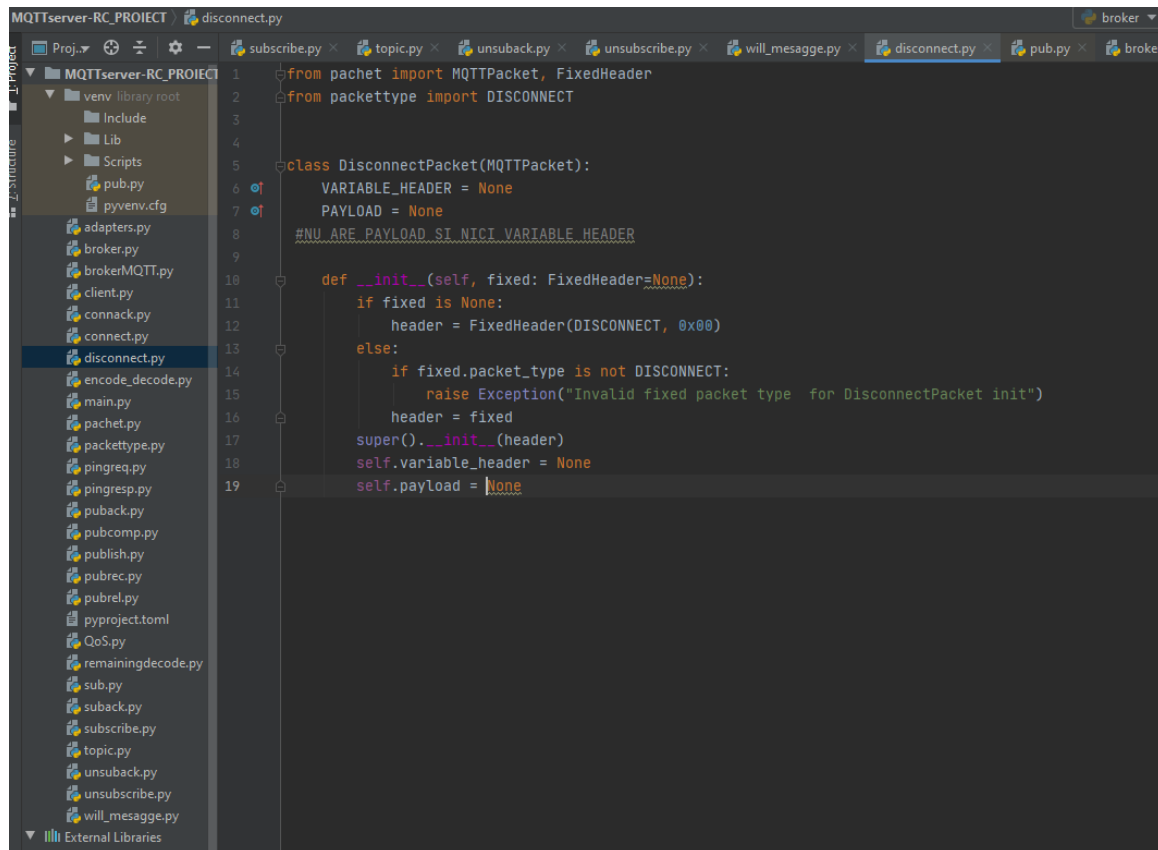
Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
byte 2...	Remaining Length							

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

Pentru a primi mesaje de la un broker MQTT, un client se conectează la broker și creează abonamente la topicurile de care este interesat. Dacă conexiunea dintre client și broker este întreruptă în timpul unei sesiuni non-persistente, aceste subiecte se pierd și clientul trebuie să se aboneze din nou la reconectare. Re-abonarea de fiecare dată când conexiunea este întreruptă este o povară pentru clienții constrânși cu resurse limitate. Pentru a evita această problemă, clientul poate solicita o sesiune persistentă atunci când se conectează la broker. Sesiunile persistente salvează toate informațiile relevante pentru client pe broker. Id-ul clientului oferit atunci când stabilește conexiunea cu brokerul identifică sesiunea

Când clientul se conectează la broker, acesta poate solicita o sesiune persistentă. Clientul folosește un flag “cleanSession” pentru a spune brokerului de ce tip de sesiune are nevoie: Când flagul “cleanSession” este setat pe ‘1’, clientul nu dorește o sesiune persistentă. Dacă clientul se deconectează din orice motiv, toate informațiile și mesajele aflate în coadă dintr-o sesiune persistentă anterioară se pierd. Când flagul “cleanSession” este setat pe ‘0’, brokerul creează o sesiune persistentă pentru client. Toate informațiile și mesajele sunt păstrate până la următoarea dată când clientul solicită o “cleanSession”. Dacă flagul “cleanSession” este setat pe ‘0’ și brokerul are deja o sesiune disponibilă pentru client, acesta folosește sesiunea existentă și livrează mesajele la coadă anterior clientului.

-Deconectare(“Disconnect”) - Pachetul final trimis de la client la server care indică de ce conexiunea este închisă.



```
1 from packet import MQTTPacket, FixedHeader
2 from packettype import DISCONNECT
3
4
5 class DisconnectPacket(MQTTPacket):
6     VARIABLE_HEADER = None
7     PAYLOAD = None
8     #NU ARE PAYLOAD SI NICI VARIABLE HEADER
9
10    def __init__(self, fixed: FixedHeader=None):
11        if fixed is None:
12            header = FixedHeader(DISCONNECT, 0x00)
13        else:
14            if fixed.packet_type is not DISCONNECT:
15                raise Exception("Invalid fixed packet type for DisconnectPacket init")
16            header = fixed
17        super().__init__(header)
18        self.variable_header = None
19        self.payload = None
```

După trimiterea unui pachet “DISCONNECT” Clientul:

- TREBUIE să închidă conexiunea la rețea
- NU TREBUIE să mai trimit pachete de control pe acea conexiune de rețea

La primirea DECONNECTĂRII serverului:

- TREBUIE să renunțe la orice „Will message” asociat cu conexiunea curentă fără a-l publica.
- TREBUIE să închidă conexiunea la rețea dacă Clientul nu a făcut deja acest lucru.

Serverul TREBUIE să valideze faptul că biții rezervați(„reserved”) sunt setați la zero și deconectează Clientul dacă nu sunt zero.

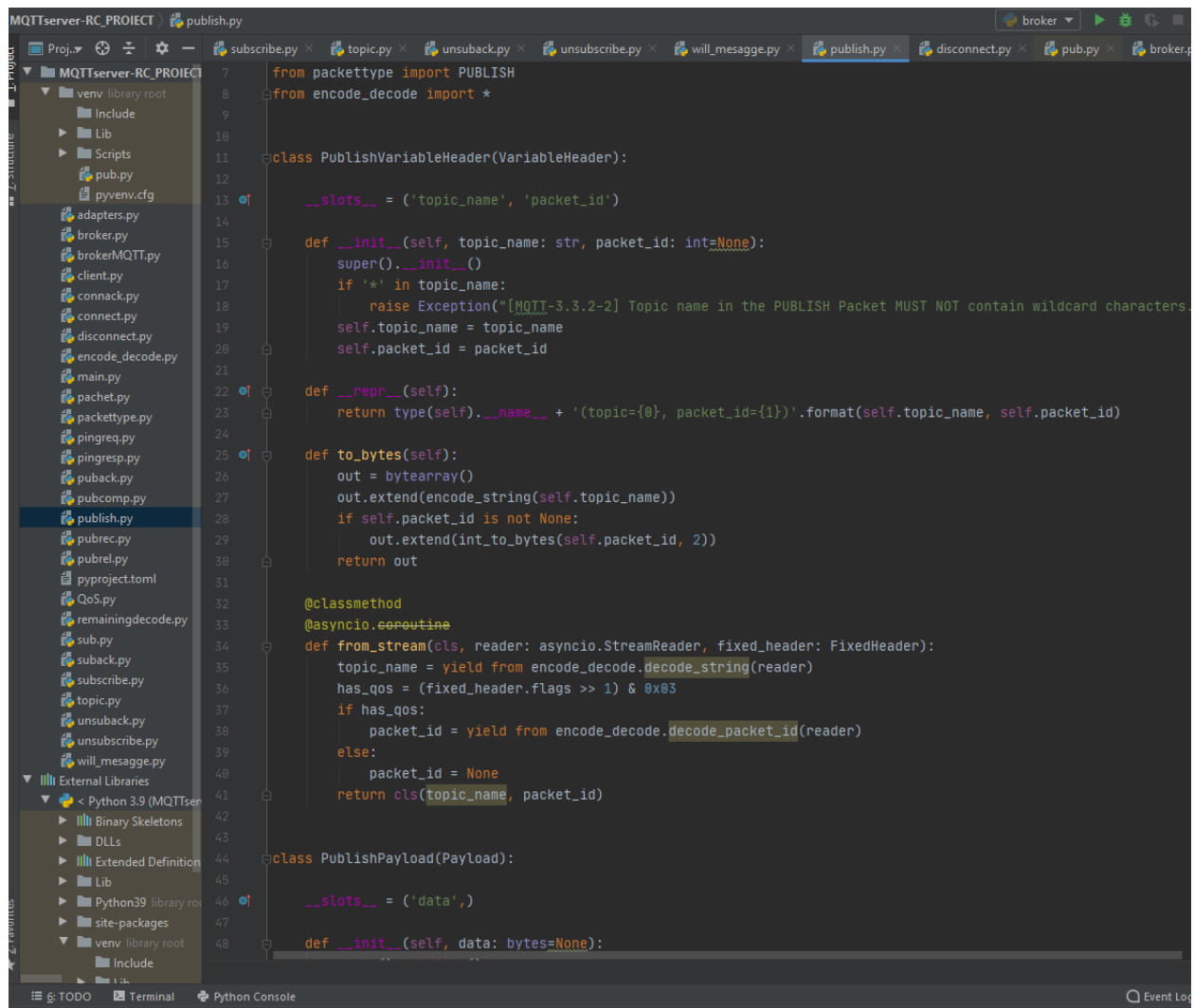
-Abonare(“Subscribe”)

— Un pachet de abonare este întotdeauna trimis de la client la server pentru a crea unul sau mai multe abonamente la topicuri.

```
7
8
9 class SubscribePayload(Payload):
10
11     __slots__ = ('topics',)
12
13     def __init__(self, topics=[]):
14         super().__init__()
15         self.topics = topics
16
17     def to_bytes(self, fixed_header: FixedHeader, variable_header: VariableHeader):
18         out = b''
19         for topic in self.topics:
20             out += encode_string(topic[0])
21             out += int_to_bytes(topic[1], 1)
22         return out
23
24     @classmethod
25     @asyncio.coroutine
26     def from_stream(cls, reader: asyncio.StreamReader, fixed_header: FixedHeader,
27                     variable_header: VariableHeader):
28         topics = []
29         payload_length = fixed_header.remaining_length - variable_header.bytes_length
30         read_bytes = 0
31         while read_bytes < payload_length:
32
33             topic = yield from encode_string.decode_string(reader)
34             qos_byte = yield from encode_string.read_or_raise(reader, 1)
35             qos = bytes_to_int(qos_byte)
36             topics.append((topic, qos))
37             read_bytes += 2 + len(topic.encode('utf-8')) + 1
38
39         return cls(topics)
40
41     def __repr__(self):
42         return type(self).__name__ + '(topics={!r})'.format(self.topics)
43
44
45 class SubscribePacket(MQTTPacket):
46     VARIABLE_HEADER = PacketIdVariableHeader
47     PAYLOAD = SubscribePayload
48
49     SubscribePayload -> from_stream()
```

-Publicare(“Publish”)

— Un pachet de publicare poate fi trimis de la client la server, transportând de obicei un mesaj de aplicație sau de la server la un client care s-a abonat la subiectul respectiv.



Cerintele proiectului:

Interfață grafică care să permită demonstrarea tuturor modurilor de funcționare



Listă de topic-uri configurabilă(creare/stergere din GUI, fisier config.).

Tratarea clienților care vor să se aboneze la un topic inexistent.

Clienții vor primi un mesaj corespunzător.

Vizualizare clienți conectați și abonați, deconectare forțată client.

Se va afișa lista clienților salvați drept “conectați/abonați”.

Vizualizarea istoricului pentru ultimele 10 valori publicate/topic.

Se va afișa ultimele topicuri publicate.

Autentificare clienți.

Clientii se pot autentifica printr-un username/email si o parola.

Implementare mecanism KeepAlive.

Keep Alive este un interval de timp măsurat în secunde. Exprimat ca un cuvânt pe 16 biți, este maxim 539 intervale de timp permise să treacă între momentul în care clientul termină transmiterea unui mesaj și momentul în care începe să trimită următorul.

Dacă valoarea Keep Alive este diferită de zero și Serverul nu primește un pachet de control de la client în decurs de o perioadă și jumătate de timp Keep Alive, TREBUIE să deconecteze conexiunea de rețea-Client ca și cum rețeaua ar fi eșuat.

O valoare Keep Alive de zero (0) are ca efect oprirea mecanismului Keep Alive. Aceasta înseamnă că Serverul nu este obligat să deconecteze Clientul din motive de inactivitate, dar o poate face pentru că îi este permis să deconecteze un client pe care îl consideră inactiv sau care nu răspunde în orice moment, indiferent de valoarea Keep Alive furnizată de clientul respectiv.

Implementare QoS 0,1,2.

Quality of Service 0 : publicarea mesajelor fara asteptarea unui feedback din partea serverului.

Quality of Service 1 : publicarea mesajului in mod repetat pana la o confirmare a serverului.

Quality of Service 2 : publicarea mesajului, asteptarea confirmarii primirii mesajului, confirmarea ca mesajul poate fi procesat , iar apoi confirmarea incheierii transmisiunii.

```

MQTTServer-RC_PROJECT QoS.py
1 qos_0 = 0x00
2 qos_1 = 0x01
3 qos_2 = 0x02

```

```

MQTTServer-RC_PROJECT publish.py
72 class PublishPacket(MQTTPacket):
73     VARIABLE_HEADER = PublishVariableHeader
74     PAYLOAD = PublishPayload
75
76     DUP_FLAG = 0x08
77     RETAIN_FLAG = 0x01
78     QOS_FLAG = 0x06
79
80     def __init__(self, fixed: FixedHeader=None, variable_header: PublishVariableHeader=None, payload=None):
81         if fixed is None:
82             header = packet.MQTTFixedHeader(PUBLISH, 0x00)
83         else:
84             if fixed.packet_type is not PUBLISH:
85                 raise Exception("Invalid fixed packet type %s for PublishPacket init")
86             header = fixed
87
88         super().__init__(header)
89         self.variable_header = variable_header
90         self.payload = payload
91
92     def set_flags(self, dup_flag=False, qos=0, retain_flag=False):
93         self.dup_flag = dup_flag
94         self.retain_flag = retain_flag
95         self.qos = qos
96

```

O utilizare a Quality of Service este pachetul "PUBLISH", intrucat acesta contine pe bit 1 si bit 2 numarul in binar pentru QoS.

715 **3.3 PUBLISH – Publish message**

716 A PUBLISH Control Packet is sent from a Client to a Server or from Server to a Client to transport an
 717 Application Message.

718 **3.3.1 Fixed header**

719 Figure 3.10 – PUBLISH Packet fixed header illustrates the fixed header format:

720 **Figure 3.10 – PUBLISH Packet fixed header**

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

721

Implementare mecanism LastWill

În MQTT, se utilizează caracteristica ” Last Will and Testament”(LWT) pentru a anunța alți clienți despre un client deconectat. Fiecare client își poate specifica ultimul mesaj de testament atunci când se conectează la un broker. Ultimul mesaj de testare este un mesaj MQTT normal cu un subiect, flagul mesajului reținut, QoS și ”payload”-ul respectiv. Brokerul stochează mesajul până când detectează că clientul s-a deconectat forțat. Ca răspuns la deconectarea lipsită de grație, brokerul trimite ultimul mesaj către toți clienții abonați la ultimul topic. Dacă clientul se deconectează grațios cu un mesaj corect DECONNECTARE, brokerul renunță la mesajul LWT stocat.