Documentație pentru serverul MQTT, implementat la disciplina RC-P

Cebotaroș Emil

Chiriac Ion

Munteanu Letiția-Ioana

grupa 1307A, Facultatea de Automatică și Calculatoare
Universitatea Tehnică "Gheorghe Asachi", Iași, România

October 29, 2020

Cuprins

Capitolul 1

Introducere

MQTT este un protocol de comunicare M2M (Machine 2 Machine), utilizat frecvent în aplicații IoT. Aplicația noastră implementează versiunea 5 a protocolului.

1.1 Documentație

În cadrul etapei de proiectare, au fost identificate 22 clase necesare pentru a asigura ușurința procesului de dezvoltare și modularitatea aplicației.

1.1.1 Clase de date

Clasa BinaryData

MQTT codifică anumite date det tip binar într-o formă ușoară pentru procesare. Respectiv, această clasă va aveam câmpul `length` de tip int, și `data` de tip `bytes`.

VariableByte

MQTT codifică anumite date de tip întreg într-un număr variabil de octeți. Respectiv, această clasă va avea câmpul `data`, de tip `bytes`. Metodele statice implementate de această clasă vor fi :
`encode(nr : int)->bytes`
`decode(byteseq : bytes)->int`.

CustomUTF8

MQTT codifică anumite date de tip string în format Unicode. Python3 consideră în mod implicit obiectele de tip string ca fiind în format Unicode. Totuși, formatarea Python nu corespunde cu specificațiile implementării MQTT. Din această cauză este necesară implementarea clasei CustomUTF8. Aceasta va

avea câmpul `data`, de tip `bytes`. Metodele implementate de această clasă vor fi statice, cu următoarele definiții
`encode(text : str)->bytes`
`decode(byteseq : bytes)->str`.

1.1.2 Configurare

Pentru configurarea serverului, am decis să avem un fișier de configurare *config/broker.cfg*, asemănător după structură cu fișierul de configurare pentru serviciul SSH. Acesta va conține setările pe mediul de stocare intern al gazdei pe care rulează brokerul. Pentru a asigura că aceste setări sunt mereu actuale, a fost creată clasa Config. Câmpurile acesteia sunt:

```
configurations['AllowPublicAccess'] -> bool
configurations['AuthenticationMethods'] -> list(str)
configurations['KnownClientsPath'] -> str
configurations['HistoryMessageQueueSize'] -> int
configurations['RetainedMessagesPath'] -> str
configurations['DefaultKeepAlive'] -> int #specificat în secunde
configurations['TopicHierarchyListPath'] -> str
configurations['MaxPacketSize'] -> int
configurations['MaxQoS1QueueSize'] -> int
configurations['MaxQoS2QueueSize'] -> int
configurations['MaxSupportedTopics'] -> int
configurations['SessionsPath'] -> str
configurations['AllowWildcard'] -> bool
configurations['configPath'] -> str
__init__(self, confPath : str)
reload(self) -> None
```

1.1.3 Clasa MQTTPacket

Deoarece MQTT este un protocol bazat pe pachete, am considerat ca fiind imperativă crearea unei clase generice care să modeleze trăsăturile comune ale unui pachet MQTT. Această clasă va conține 3 câmpuri de bază, având câte o metodă de parsare pentru fiecare.

Implementare generală în Python

```
packet['fixed'] -> bytes
packet['variable'] -> bytes
packet['payload']-> bytes
#Funcțiile de parsare vor adăuga câmpuri noi
#în interiorul claselor.
#Acestea vor fi extinse în subclase.
parseFixedHeader(self) -> None
parseVariableHeader(self) -> None
```

```
parsePayloadHeader(self) -> None
```

1.1.4 Subclasele MQTTPacket

Clasa ConnectPacket

Clasa ConnectPacket descrie un pachet MQTT de tip CONNECT. Aceasta va moșteni din clasa generică *MQTTPacket* și va suprascrie metodele *parseFixedHeader* , *parseVariableHeader* și *parsePayloadHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'protocolName' : str,
'protocolLevel':int,
'flags' : dict(
'username':bool,
'password':bool,
'willRetain':bool,
'willQoS':int,
'willFlag':bool,
'cleanStart':bool
),
'keepAlive':int,
'propertyLength':int,
properties:dict(
'sessionExpiry':int,
'receiveMaximum':int,
'maximumPacketSize':int,
'topicAliasMaximum':int,
'requestResponseInformation':bool,
'requestProblemInformation':bool,
'userProperty':list(str),
'authenticationMethod':str,
'authenticationData':BinaryData)
)
def parsePayloadHeader(self) -> dict(
'clientId':str,
'willProperties':dict(
'propertyLength':int,
'willDelay':int,
'payloadFormatIndicator' : int,
```

```
'messageExpiryInterval':int,
'contentType' : str,
'responseTopic':str,
'correlationData':BinaryData,
'userProperty':list(str),
'willTopic':str,
'willPayload':BinaryData,
'username':str,
'password':BinaryData
)
)
```

Clasa ConnackPacket

Clasa ConnectPacket descrie un pachet MQTT de tip CONNACK. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader* și *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int)
)
def parseVariableHeader(self) -> dict(
'sessionPresent':bool,
'reasonCode':int,
'propertyLength':int,
'properties':dict(
'sessionExpiry':int,
'receiveMaximum':int,
'maximumQoS':bool,
'retainAvailable':bool,
'maximumPacketSize':int,
'assignedClientID':str,
'topicAliasMaximum':int,
'reasonString':str,
'wildcardSubscriptionAvailable':bool,
'subscriptionIdentifierAvailable':bool,
'sharedSubscriptionAvailable':bool,
'serverKeepAlive':int,
'responseInformation':str,
'serverReference':str,
'authenticationMethod':str,
'authenticationData':BinaryData
```

```
)
)
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip CONNACK nu au payload.

Clasa PublishPacket

Clasa ConnectPacket descrie un pachet MQTT de tip PUBLISH. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader* și *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'dup':bool,
'QoS':int,
'retain':bool,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'topicName':str,
'packetIdentifier':int,
'propertyLength':int,
'properties':dict(
'payloadFormatIndicator':int,
'messageExpiryInterval':int,
'topicAlias':int,
'responseTopic':str,
'userProperty':list(str),
'subscriptionID':int,
'contentType':str)
)
}
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PUBLISH nu au payload.

Clasa PubackPacket

Clasa ConnectPacket descrie un pachet MQTT de tip PUBACK. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*, *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'packetIdentifier':int,
'reasonCode':int,
'propertyLength':int,
'properties':dict(
'reasonString': str,
'userProperty':list(str)
)
)
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PUBLISH nu au payload.

Clasa PubrecPacket

Clasa ConnectPacket descrie un pachet MQTT de tip PUBREC. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixed-Header*, *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'packetIdentifier':int,
'reasonCode':int,
'propertyLength':int,
'properties':dict(
'reasonString': str,
'userProperty':list(str)
)
)
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PUBLISH nu au payload.

Clasa PubrelPacket

Clasa PubrelPacket descrie un pachet MQTT de tip PUBREL. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixed-*

Header si *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'reasonCode' : list(int),
'properties' : dict(
'propertyLength' : int,
'reasonString': str,
'userProperty' : list(str)
)
)
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PUBREL nu au payload.

Clasa PubcompPacket

Clasa PubcompPacket descrie un pachet MQTT de tip PUBCOMP. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader* si *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'reasonCode' : list(int),
'propertyLength' : int,
'properties' : dict(
'reasonString': str,
'userProperty' : list(str)
)
)
}
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PUBCOMP nu au payload.

Clasa SubscribePacket

Clasa SubscribePacket descrie un pachet MQTT de tip SUBSCRIBE. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*, *parseVariableHeader* și *parsePayloadHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int)
)
def parseVariableHeader(self) -> dict(
'propertyLength':int,
'properties' : dict(
'subscriptionIdentifier' : int,
'userProperty' : list(str)
)
)
def parsePayloadHeader(self) -> dict(
'topicFilters' : list(str)
)
```

Clasa SubackPacket

Clasa SubackPacket descrie un pachet MQTT de tip SUBACK. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*, *parseVariableHeader* și *parsePayloadHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
def parseVariableHeader(self) -> dict(
'propertyLength':int,
'properties' : dict(
'reasonString' : str,
'userProperty' : list(str)
)
)
def parsePayloadHeader(self) -> dict(
'subscribeReasonCodes' : list(int)
)
```

Clasa UnsubscribePacket

Clasa UnsubscribePacket descrie un pachet MQTT de tip UNSUBSCRIBE. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*, *parseVariableHeader* și *parsePayloadHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int)
)
def parseVariableHeader(self) -> dict(
'propertyLength':int,
'properties' : dict(
'userProperty' : list(str)
)
)
def parsePayloadHeader(self) -> dict(
'topicFilters' : list(str)
)
```

Clasa UnsubackPacket

Clasa UnsubackPacket descrie un pachet MQTT de tip UNSUBACK. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*, *parseVariableHeader* și *parsePayloadHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int)
)
    def parseVariableHeader(self) -> dict(
     'packetIdentifier': int,
     'propertyLength': int,
     'properties':dict(
     'reasonString':str,
     'userProperty':list(str)
     )
     )
def parsePayloadHeader(self) -> dict(
'reasonCode':list(int)
)
}
```

Clasa PingreqPacket

Clasa PingreqPacket descrie un pachet MQTT de tip PINGREQ. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
)
```

Metodele *parseVariableHeader* și *parsePayloadHeader* nu vor avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PINGREQ nu au nici header variabil, nici payload.

Clasa PingrespPacket

Clasa PingrespPacket descrie un pachet MQTT de tip PINGRESP. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
)
```

Metodele *parseVariableHeader* și *parsePayloadHeader* nu vor avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip PINGRESP nu au nici header variabil, nici payload.

Clasa AuthPacket

Clasa AuthPacket descrie un pachet MQTT de tip AUTH. Aceasta va moșteni din clasa generică MQTTPacket și va suprascrie metodele *parseFixedHeader* si *parseVariableHeader*. În urma execuției metodelor, vor fi generate dicționare care conțin datele parsate în format human-readable. Astfel, definițiile metodelor vor deveni următoarele :

```
def parseFixedHeader(self) -> dict(
'controlType' : int,
'remainingLength' : int
)
    def parseVariableHeader(self) -> dict(
     'reasonCode':int,
     'propertyLength':int,
```

```
     'properties':dict(
     'authenticationMethod':str,
     'authenticationData':BinaryData,
     'reasonString':str,
     'userProperty':list(str)
     )
    )
}
```

Metoda *parsePayloadHeader* nu va avea nevoie de suprascriere pentru că în specificație este menționat faptul că pachetele de tip AUTH nu au payload.

Clasa MalformedPacket

Clasa MalformedPacket descrie un pachet MQTT malformat. Aceasta nu va suprascrie funcții și va fi utilizată pentru marca un request greșit.

1.1.5 Clasa Session

Clasa Session a fost inclusă în design pentru a stoca datele care țin de o sesiune de comunicare client-broker. Aceasta include și metode pentru comunicarea cu clienții, gestionând automat generarea de pachete de răspuns prin intermediul metodei `handlePacket`.

Structură generală în Python

```
session_state['clientId'] -> str
session_state['subscribedTopics'] -> dict(
'subscriptId' : str,
'topic':str
)
session_state['end_time'] -> date
session_state['QoS1packets'] -> Queue(MQTTPacket)
session_state['QoS2packets'] -> Queue(MQTTPacket)
session_state['last_will'] -> dict
classifyData(data : bytes) -> MQTTPacket
handlePacket(packet : MQTTPacket) -> packet : MQTTPacket or None
sendResponse(packet : MQTTPacket) -> None
store() -> None
forceDisconnect() -> None
```

1.1.6 Clasa Keeper

Clasa Keeper a fost inclusă în design pentru a reține istoricul pachetelor. Conform cerințelor de implementare, această clasă este responsabilă pentru stocarea ultimelor *HistoryMessageQueueSize* mesaje. În plus, aceasta stochează pachetele ce aveau bitul de Retain setat. Aceastea sunt stocate într-un fișier,

calea căruia este specificată în fișierul de configurare la valoarea *RetainedMessagesPath*

Structură generală în Python

```
retained : dict(topic : str, msg : MQTTPacket)
updateMessageOnTopic(topic : str, packet : MQTTPacket)
history : dict(topic : str, queue : Queue(MQTTPacket))
```