

Trabalho 1 - Transmissão de Dados

Simulação de rede de sensores utilizando protocolo MQTT

Rosana Santos Ribeiro, 16/0047269 - rosanasrib@gmail.com
<https://github.com/rosantori/TrabalhosUnB.git>

¹Dep. Ciência da Computação - Universidade de Brasília (UnB)
CiC - Transmissão de Dados - Turma A

1. Objetivos

Permitir que o aluno(a) pratique os conhecimentos obtidos durante o semestre em sala de aula, utilizando um ambiente virtual (IoT) e implementando a comunicação entre um servidor e diversos clientes, a partir do protocolo *Message Queuing Telemetry Transport*, ou MQTT.

2. Introdução

Message Queuing Telemetry Transport, mais conhecido como MQTT é um protocolo da camada de aplicação, baseado no TCP e na arquitetura *publisher/subscriber*.

A comunicação ocorre por intermédio de um *broker*, que interage com os clientes, *publisher* e *subscriber*.

O *publisher* envia mensagens ao *broker* com tópicos definidos e os *subscribers* se conectam ao *broker* e se inscrevem em tópicos de interesse. De tal forma que o *broker* é quem realiza a organização das mensagens. É importante ressaltar que todas as interações são procedidas de mensagens de confirmação. Há mensagens de confirmação:

- após ocorrer a conexão do *broker* com o cliente;
- após ser enviado o pedido de inscrição pelo *subscriber*;
- após ser publicada e transmitida uma mensagem.

Além desses, há as mensagens para testar se os clientes estão conectados ao servidor ainda.

Diferente do protocolo HTTP, que também influencia na camada de aplicação, o protocolo MQTT abre uma conexão entre o servidor e o cliente e de acordo com a necessidade, ocorre a troca de mensagens. A conexão só é fechada quando é de interesse do cliente ou do *broker*. O protocolo HTTP não trabalha dessa forma. Basicamente, o cliente faz um requerimento ao servidor desejado e este responde, encerrando a conexão logo após o requerimento estiver entregue.

Como ilustra a Figura 1, a comunicação entre o broker e os clientes se dá mediante mensagens de confirmação.

3. Métodos

De acordo com o roteiro fornecido, a primeira etapa é entender o funcionamento do protocolo MQTT. Sua arquitetura é baseada em um padrão *publisher/subscriber*, de tal forma que a comunicação ocorre por intermédio de um servidor *broker*. É importante ressaltar que é um protocolo da camada de aplicação que utiliza o protocolo TDP na camada de transporte. Basicamente, o *publisher* envia uma mensagem com um certo tópico ao *broker*, que transmite essa mensagem a todos os *subscribers* desse tópico. Caso a mensagem enviada não tenha um tópico, ou algum cliente inscrito nesse tópico, ela é descartada. Caso um cliente esteja inscrito

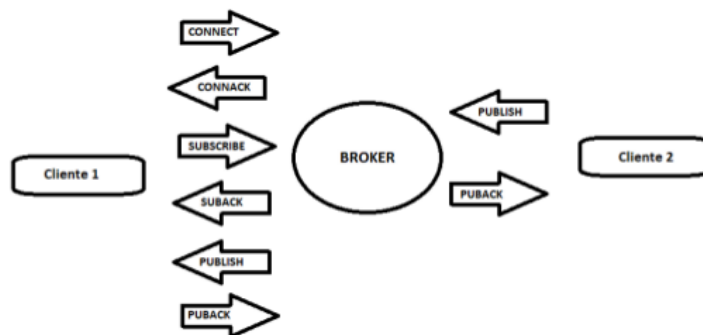


Figure 1. Arquitetura do MQTT.

em um tópico que não há nenhum *publisher* publicando mensagens, o cliente apenas ficará sem receber nada.

Além disso, desde a conexão do cliente ao servidor até o recebimento de uma mensagem transmitida pelo *broker*, há mensagens de confirmações, conhecidas como *acks*. Toda troca de mensagem é procedida por um *ack*.

A implementação foi feita usando a versão 3.5.2 de *Python*, com o auxílio das bibliotecas *Paho MQTT* e *Socket*, principalmente.

3.1. Clients

Utilizando a biblioteca *paho.mqtt.client* foi possível implementar os clientes, tanto os *publishers* quanto os *subscribers*. Neste trabalho, cada cliente só poderá ter um papel, ou o de receber as mensagens ou o de publicá-las.

A conexão com o *broker* é feita utilizando o endereço de IP do computador, que poderá ser obtido com o comando *ifconfig* no terminal GNU/Linux, como mostra a Figura 2.

```

rosaninhagaleidoscopio:~/Documents/TrabalhosUns/105 ifconfig
enp2s0  Link encap:Ethernet  HWaddr 98:83:89:74:01:24
        UP BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:15168 errors:0 dropped:0 overruns:0 frame:0
        TX packets:15168 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:997289 (997.2 KB)  TX bytes:997289 (997.2 KB)

wlp1s0  Link encap:Ethernet  HWaddr 98:83:89:8d:f2:6f
        inet addr:192.168.1.143  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::b6a:750b:8037:1678/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:440814 errors:0 dropped:0 overruns:0 frame:0
        TX packets:92395 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:579098349 (579.0 MB)  TX bytes:14042095 (14.0 MB)

```

Figure 2. Comando *ifconfig* no terminal e o IP utilizado no trabalho.

3.1.1. Publishers

Neste trabalho, serão publicadas mensagens com dois tópicos distintos, sendo eles: "UMIDADE" e "TEMPERATURA". A simulação do sensor utilizado é feita com dados que serão obtidos a partir de um arquivo teste provido pelo professor.

Os dados de leitura estão padronizados em linhas de 46 caracteres, sendo o 47º a quebra de linha, onde os caracteres 11 a 16 representam o valor da umidade, em porcentagem, e os caracteres de 32 a 37 representam o valor da temperatura. Somente esses caracteres serão transmitidos ao *broker*, como mostra a Figura 3.

```
while True:
    msg = f.readline()
    if not msg: break
    client.publish(topic = "UMIDADE", payload = msg[10:15], qos = 0, retain = False)
    client.publish(topic = "TEMPERATURA", payload = msg[31:36], qos = 0, retain = False)
    time.sleep(0.1)
f.close()
```

Figure 3. Fragmento do código do *publisher*.

3.1.2. Subscribers

O *subscriber*, ao conectar-se ao *broker*, entra em um *loop* esperando receber alguma mensagem com o tópico em que está inscrito. Na implementação da Figura 4 ocorre um exemplo de um *subscriber* cujo tópico que é inscrito é definido por código, sendo "UMIDADE". Nota-se que não é implementado nenhuma resposta *ack*, tampouco uma forma de desconectar-se do servidor.

3.2. Servidor - Broker

O *broker* realiza o intermédio da comunicação entre os clientes. Ele é responsável por toda a organização das mensagens, de acordo com os tópicos. Em um primeiro momento, para testes, foi utilizado o *Eclipse Mosquitto*, que utiliza o protocolo MQTT na transmissão das mensagens. O objetivo era a implementação de um servidor com a biblioteca *socket*. O funcionamento deve cumprir os seguintes requisitos:

- Permitir a conexão com múltiplos clientes;
- Enviar mensagens de confirmação para cada interação que é iniciada por um cliente;
- Organizar os tópicos das mensagens e transmiti-las de acordo com os seus *subscribers*.

4. Resultados

4.1. Publisher e Subscriber

Neste trabalho, a implementação dos clientes obteve resultados que não foram esperados.

O *publisher* envia as mensagens para o *broker* corretamente e o *subscriber* recebe as mensagens corretamente. No entanto, não há o envio de mensagens de confirmação ou a implementação dos usuários com senha para entrar no servidor. Os resultados podem ser verificados na Figura 5.

```

import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
    print("Connection status: "+rc)

def on_message(client, userdata, msg):
    print("Message received: "+msg.payload.decode())

broker_host = input("Digite o host:") #default
broker_port = 1883 #default

username = input("Digite o username: ")

client = mqtt.Client()
client.connect(broker_host, broker_port)
client.on_connect = on_connect
client.on_message = on_message
client.subscribe("UMIDADE")

client.loop_forever()

```

Figure 4. Implementação de um cliente *subscriber* no tópico "UMIDADE"

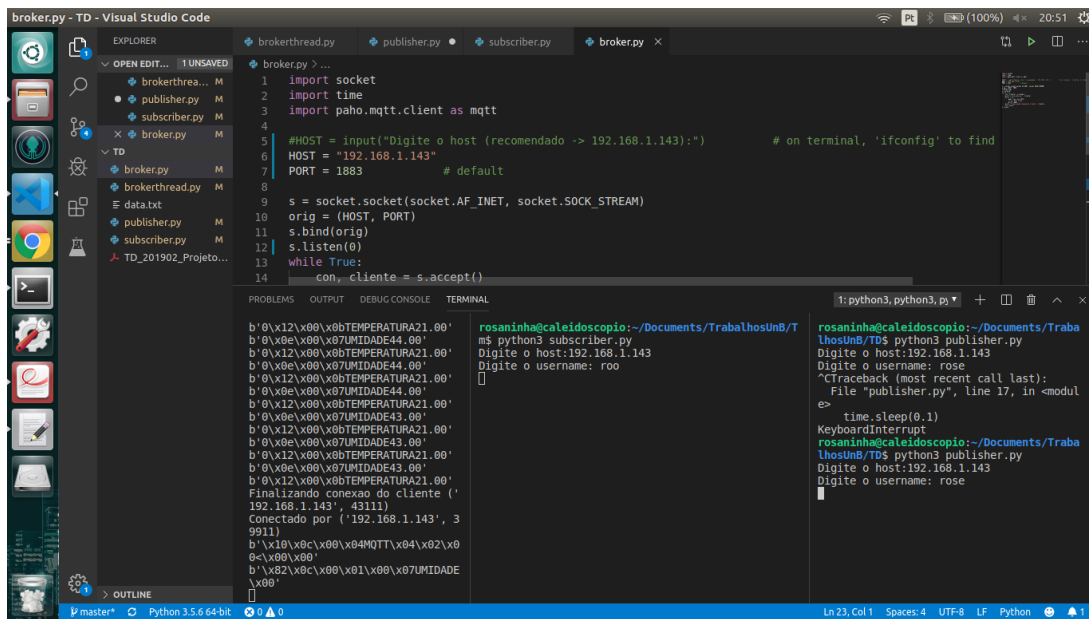
4.2. Servidor - (Broker)

A implementação do servidor não apresentou os resultados desejados, tampouco supriu as demandas do Roteiro.

A criação de um servidor foi possível via *socket*, no entanto, a forma que foi implementado somente permitiu uma conexão por vez, o que impossibilitou o *subscriber* e o *publisher* de estarem conectados ao mesmo tempo. A execução da Figura 5 foi realizada da seguinte maneira:

1. Inicializou-se o *broker*;
2. Definiu-se o *host* como 192.168.1.143;
3. Inicializou-se o *publisher* e realizou a conexão com o servidor, de mesmo endereço e porta;
4. Inicializou-se o *subscriber*, em "UMIDADE";
5. Notou-se que o *subscriber* não recebia nada e encerrou a conexão do *publisher*;
6. Reiniciou a conexão do *publisher* e verificou que nem o servidor recebia as mensagens publicadas, nem o *subscriber*.

Analizando o funcionamento do código, percebeu-se que a função "*s.accept()*" bloqueia o programa, ou seja, enquanto não houver uma conexão para ser aceita, não sairá dessa linha de código.



```
brokerthread.py
1 import socket
2 import time
3 import paho.mqtt.client as mqtt
4
5 #HOST = input("Digite o host (recomendado -> 192.168.1.143):")
6 HOST = "192.168.1.143"
7 PORT = 1883
8
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 orig = (HOST, PORT)
11 s.bind(orig)
12 s.listen(0)
13 while True:
14     con, cliente = s.accept()
```

```
subscriber.py
1 import socket
2 import time
3 import paho.mqtt.client as mqtt
4
5 #HOST = input("Digite o host (recomendado -> 192.168.1.143):")
6 HOST = "192.168.1.143"
7 PORT = 1883
8
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 orig = (HOST, PORT)
11 s.bind(orig)
12 s.listen(0)
13 while True:
14     con, cliente = s.accept()
```

```
publisher.py
1 import socket
2 import time
3 import paho.mqtt.client as mqtt
4
5 #HOST = input("Digite o host (recomendado -> 192.168.1.143):")
6 HOST = "192.168.1.143"
7 PORT = 1883
8
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 orig = (HOST, PORT)
11 s.bind(orig)
12 s.listen(0)
13 while True:
14     con, cliente = s.accept()
```

```
Terminal
1:python3,python3.py
rosaninha@caleidoscopio:~/Documents/TrabalhosUnB/T
m$ python3 subscriber.py
Digite o host:192.168.1.143
Digite o username: roo
[]
```

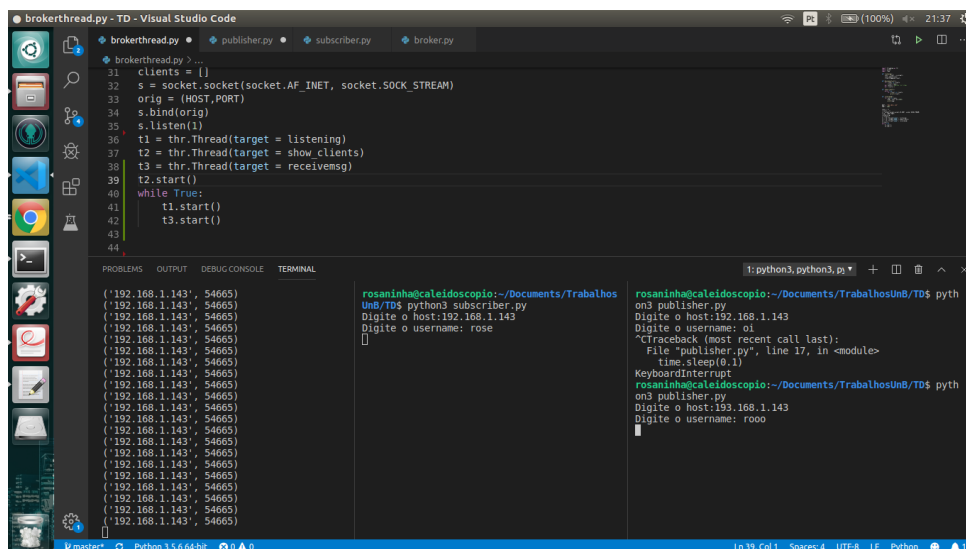
Figure 5. Execução dos 3 códigos.

4.3. Tentativa de correção

Em uma tentativa de corrigir o primeiro problema apresentado, uma solução pensada foi a programação por *threads* em python.

A ideia para a correção era de que a função para aceitar novas conexões estivesse em uma *thread* separada das demais funções do servidor *broker*, de tal forma que, a cada nova conexão, seria adicionado à uma lista o cliente recém conectado.

O início dessa implementação permitiu que o programa não parasse enquanto esperava novas conexões. No entanto, não chegou à nenhum resultado útil para o suprimento das demandas. O resultado obtido é mostrado na Figura 6.



```
brokerthread.py
31 clients = []
32 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33 orig = (HOST, PORT)
34 s.bind(orig)
35 s.listen(1)
36 t1 = thr.Thread(target = listening)
37 t2 = thr.Thread(target = show_clients)
38 t3 = thr.Thread(target = receivmsg)
39 t2.start()
40 while True:
41     t1.start()
42     t3.start()
43
44
```

```
Terminal
1:python3,python3.py
rosaninha@caleidoscopio:~/Documents/TrabalhosUnB/T
m$ python3 subscriber.py
Digite o host:192.168.1.143
Digite o username: rose
[]
```

Figure 6. Execução com threads.

A fim de visualizar o funcionamento dos clientes, foi implementado o *Mosquitto* e obteve-se os resultados da Figura 7. Nota-se que ao executar os três arquivos, o subscriber recebe as

[illegible]

Figure 7. Subscriber e Publisher com Mosquitto

mensagens corretamente. O *broker* funciona como o esperado, por ser um já implementado.

5. Discussão e conclusões

Este trabalho, mesmo que obtendo péssimos resultados, permitiu bastante o aprendizado. A implementação correta do *broker* será continuada, ainda com a tentativa voltada às *threads*. A maior dificuldade esteve na interpretação do roteiro: o que realmente era pedido para ser implementado e o que as bibliotecas já implementavam, impossibilitando as suas utilizações. No entanto, referente ao que se aprendeu sobre o protocolo MQTT, pode-se dizer que o *broker* é como o centro das atividades, pois ele faz toda a organização das mensagens recebidas. Desde a decisão de para quais clientes serão enviadas até se serão descartadas ou mantidas.

References

- [1] HIVE MQ. **Client, broker/server and connection establishment.** Disponível em : <https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/>. Acesso em: 30/11/2019.
- [2] Python.org . **Sockets.** Disponível em: <https://docs.python.org/3/library/socket.html>. Acesso em: 01/12/2019.
- [3] mosquitto. **Eclipse Mosquitto.** Disponível em: <https://mosquitto.org/>. Acesso em 01/12/2019.
- [4] Aprender. **Roteiro Trabalho.** Disponível em: https://aprender.ead.unb.br/pluginfile.php/336130/mod_resource/content/4/TD_201902_Projeto_Final.pdf. Acesso em: 20/11/2019.