MQTT BROKER

MAC0352 - Redes de Computadores

11796378 - João Henri Carrenho Rocha

Implementação:

- Foi usada a função **mmap** para criar vetores acessíveis por todos os subprocessos do broker. Ele permite que todos esses processos leiam e escrevam nesses vetores.

- São os vetores abaixo, que serão detalhados no slide seguinte:
  - names:
  - messages
  - messages_length
  - current_offset

Implementação:

- **names[topic_id]**:
  - names[topic_id] contém o nome do tópico de ID ***topic_id.***
- **messages[topic_id][offset]**:
  - contém as últimas 100 mensagens de cada tópico.
- **messages_length[topic_id][offset]**:
  - contém o tamanho da respectiva mensagem no tópico **messages.**
- **current_offset[topic_id]**:
  - há uma lógica para que os clientes saibam qual foi a última mensagem enviada dentro das 100 mensagens em **messages.**

kubernetes

Implementação:

- Foram definidos alguns limites para facilitar a implementação:
  - Há no máximo **100** tópicos;
  - O nome de cada tópico não pode ultrapassar **129** caracteres;
  - Cada tópico só armazena as últimas **100** mensagens;
  - Os pacotes não podem ultrapassar **129** caracteres unsigned;
  - Cada cliente só pode se inscrever em **1** tópico;

kubernetes

Implementação:

- **SUBSCRIBE:**
  - É checado se o tópico enviado existe. Se não for, o broker tenta o criar em um dos 100 slots.
  - Se o tópico existir ou tiver sido criado, é enviado o **SUBACK** com código **0x00**. Caso contrário é retornado o código de erro **0x80**.
  - É iniciada uma subrotina (processo filho) que ficará infinitamente checando a cada 100 milisegundos se o seu ***offset*** é diferente do ***current_offset*** do tópico. Se for, a rotina incrementa a sua própria ***offset*** e publica a mensagem para seu respectivo *subscriber.*

kubernetes

Implementação:

- Note, então, que cada uma dessas rotinas tem seu próprio ***offset*** para comparar com o servidor:

```
void start_listener_child_process(int topic_id, int connfd) {
    if ((fork()) == 0) {
        int client_offset = topics.current_offset[topic_id];

        while (1) {
            if (client_offset != topics.current_offset[topic_id]) {
                client_offset = (client_offset + 1) % TOPIC_MESSAGE_RETENTION_QUANTITY;
                write(connfd,
                      topics.messages[topic_id][client_offset],
                      topics.messages_length[topic_id][client_offset]);
            }
            usleep(100);
        }
    }
}
```

- Uma limitação óbvia é que, se 100 mensagens forem enviadas em 100 ms, algum subscriber pode perder mensagens.

kubernetes

Implementação:

- Os ***offsets*** são atualizados da seguinte maneira:

```
#define TOPIC_MESSAGE_RETENTION_QUANTITY 100

int new_offset = (topics.current_offset[topic_id] + 1) % TOPIC_MESSAGE_RETENTION_QUANTITY;
```

- Dessa maneira, se a última mensagem de um tópico estiver em ***messages[topic_id][99]***, a próxima mensagem fará o broker sobrescrever a mensagem em ***messages[topic_id][0]***.

kubernetes

Implementação:

- **PUBLISH:**
  - É checado se o tópico enviado existe. Se não for, a mensagem não é salva e/ou enviada. Isso funciona pois só trabalhamos com **QoS = 0**.

  - Se existir, os vetores ***message***, ***message_length*** e ***current_offset*** são atualizados de acordo com a lógica apresentada nos slides anteriores.

Teste:

- Os testes foram feitos utilizando Docker. Os contêineres rodam numa subrede própria e podem se comunicar entre si através da mesma:

```
$ ip -4 addr

...

4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

kubernetes

Teste:

- É possível inspecionar essa rede com o comando ***docker inspect network bridge:***

Teste:

- O broker é *containerizado* como ***ep1-joao-henri***, roda na porta **1883** e como sobe primeiro, recebe sempre o IP **172.17.0.2** dessa sub-rede.

- Foi criado um container ***ep1-joao-henri-clients*** que lança X subscribers.
  Ver **scripts/run_mqtt_subscriber.sh**

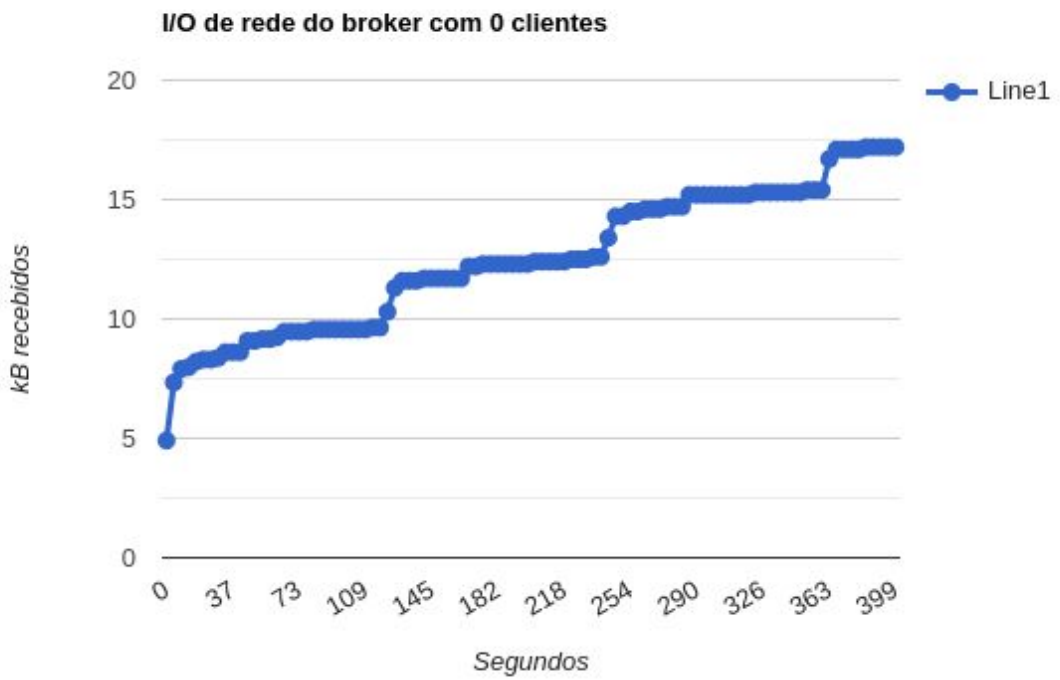
- Foi criado um container ***ep1-joao-henri-publisher*** que envia uma mensagem “mensagem” no tópico “topico” por segundo.
  Ver **scripts/publish_messages.sh**

- Foi criado um script que continuamente pega o uso de CPU e I/O de rede através do comando ***docker stats***. Esses dados foram jogados para a pasta **test-outputs** e depois gráficos foram gerados através desse site.

- Foi usado um computador com 8 CPUs e 16 GB RAM.

kubernetes

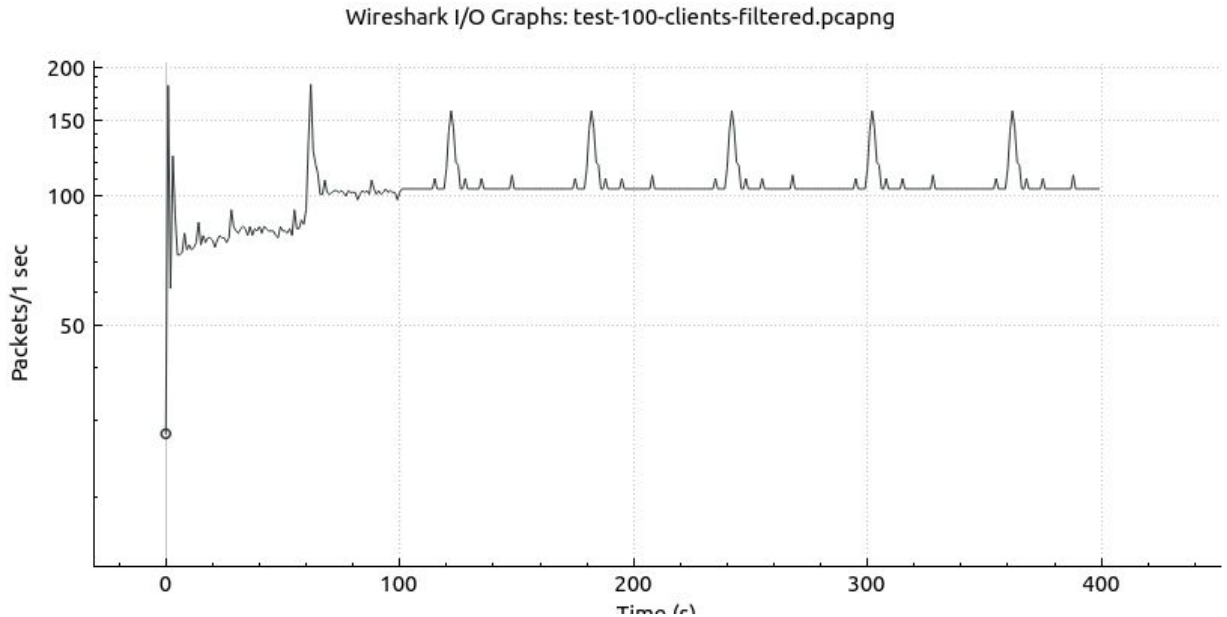
Resultados - 0 clientes:

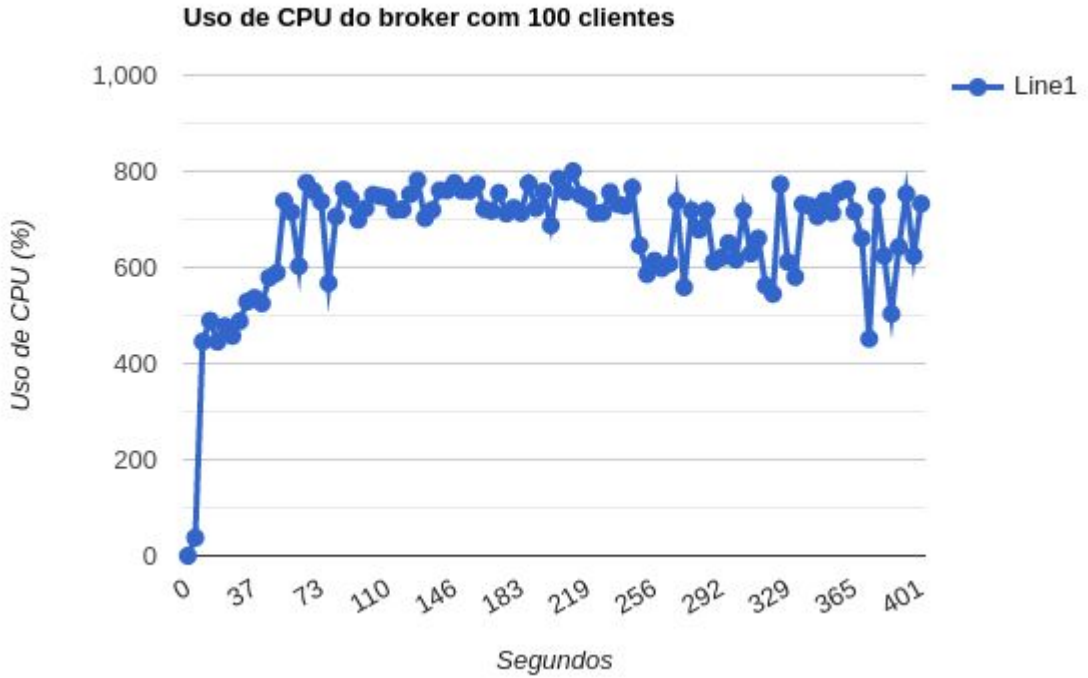


O comando docker stats usa uma API HTTP, por isso da crescente entrada de dados.

kubernetes

Resultados - 100 clientes:





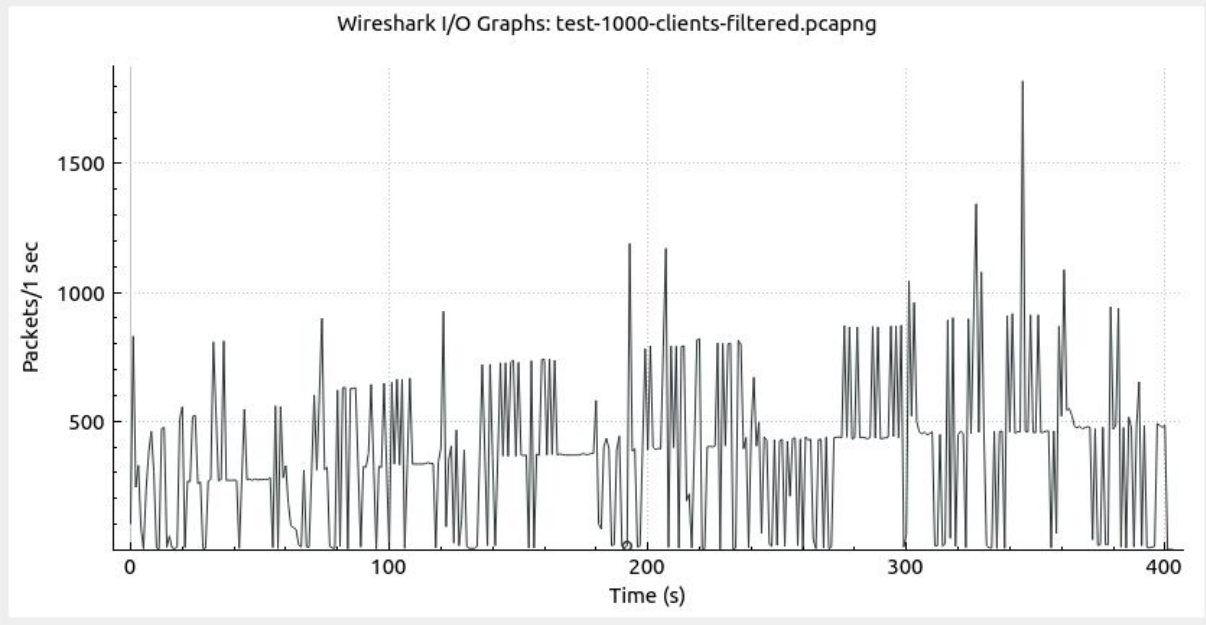
A CPU chega a 800% pois são usados 8 cores.

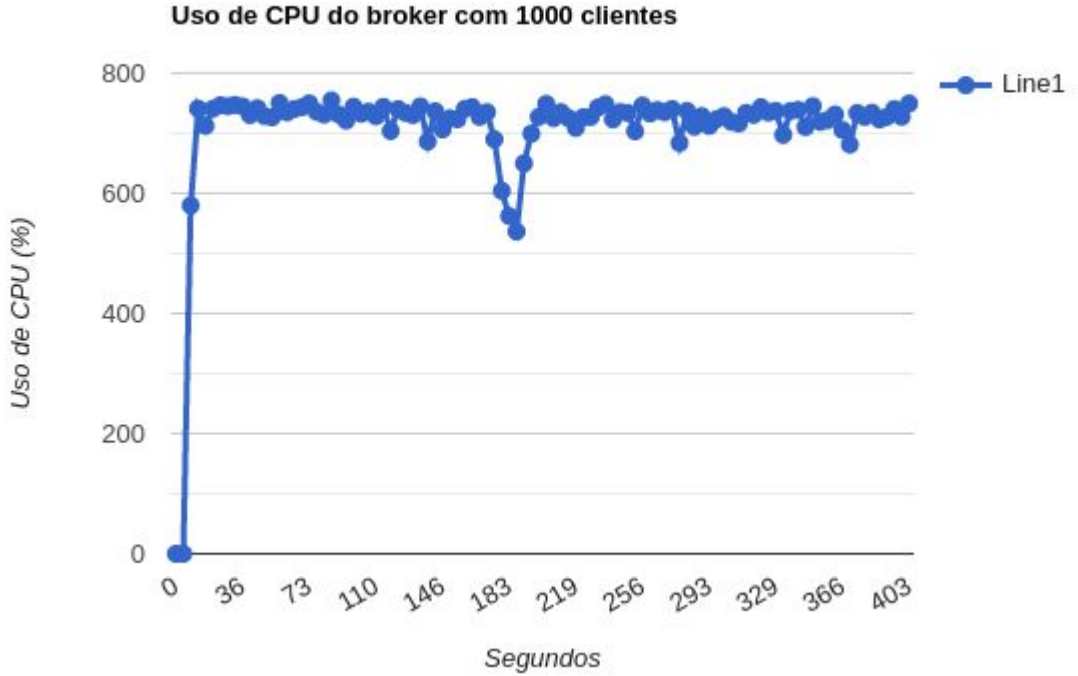
Ethernet | IPv4 · 2 | IPv6 | TCP · 516 | UDP

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
172.17.0.2	172.17.0.3	1,600	124 k	400	28 k	1,200	95 k	0.000000	399.5226	576	
172.17.0.2	172.17.0.4	39,817	3.401 k	39,018	3.343 k	799	58 k	0.342585	399.1840	67 k	

kubernetes

Resultados - 1000 clientes:



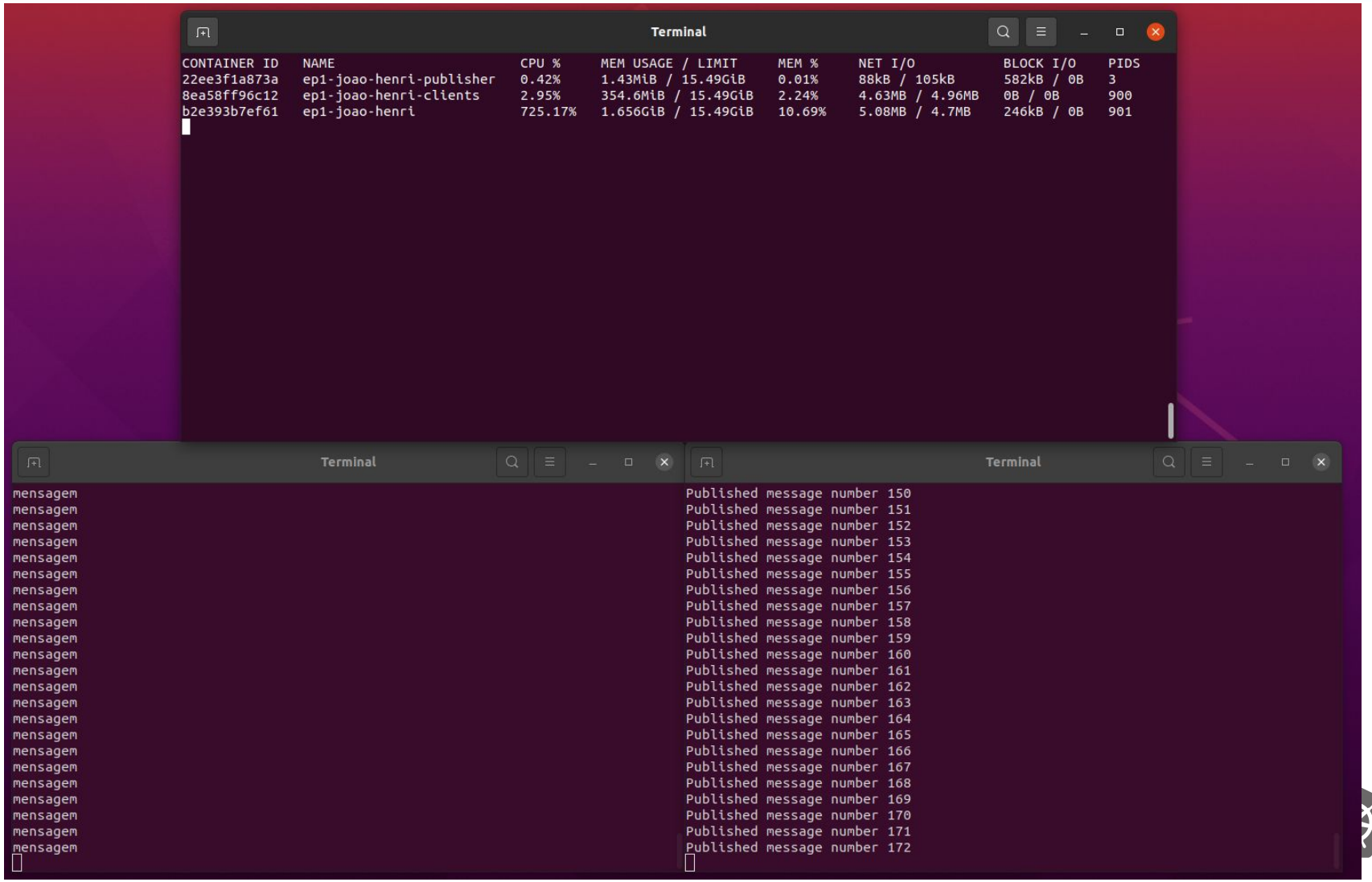


Ethernet | IPv4 · 2 | IPv6 | TCP · 2868 | UDP

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
172.17.0.2	172.17.0.3	150,592	12 M	145,386	12 M	5,206	399 k	0.000000	403.0797	247 k	
172.17.0.2	172.17.0.4	1,541	119 k	385	27 k	1,156	91 k	1.375749	398.7290	556	

kubernetes

Teste em execução:



kubernetes