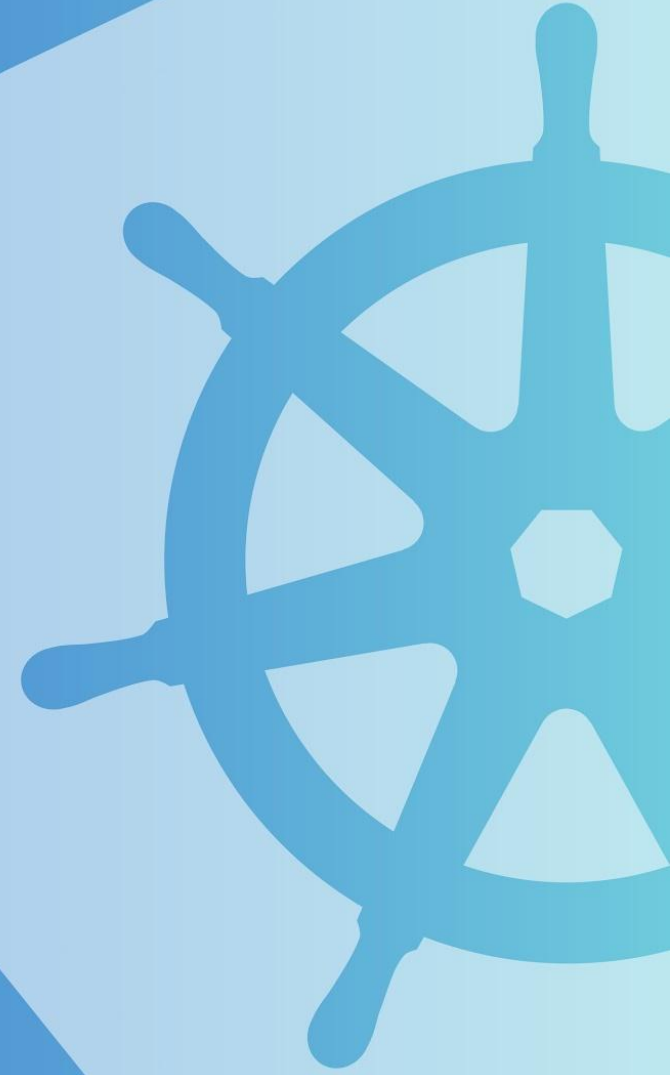


MQTT BROKER

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**.



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;



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*.



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.



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]***.



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
```

Teste:



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

```
[
  {
    "Name": "bridge",
    "Id": "6be7af6513e3871e5ef5fa4a46d42c5688f2fd54bafbb0962c637e39f7b97377",
    "Created": "2022-04-24T19:32:21.596287834-03:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "8131a8e6d0fb75d8dcf31e48eb569bd88c8d5b9595132ec05903dd16ba22d37a": {
        "Name": "ep1-joao-henri-publisher",
        "EndpointID": "e638a013302cfc83842c36da0c5d7d154b9866314509853047016534a66e135c",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16"
      },
      "a1fdd97ec0cd8e92372e22001f5c2d2cb6d995e57911fcdec532c8162270c0fe": {
        "Name": "ep1-joao-henri",
        "EndpointID": "16995feffbc39cc03b85de9639b5597fadd11ee7ec11f2db9fef9046f7c551dd",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16"
      },
      "ddcc8536ad6bf051223977cdc65facdacf1b97a2e7d73c6a97d7d3af9fa8aa34": {
        "Name": "ep1-joao-henri-clients",
        "EndpointID": "c1ec38aa67ae83c550a09fe5be5d4f690851774a3c9c4efaa0c82827e9c355a9",
        "MacAddress": "02:42:ac:11:00:04",
        "IPv4Address": "172.17.0.4/16"
      }
    }
  }
]
```



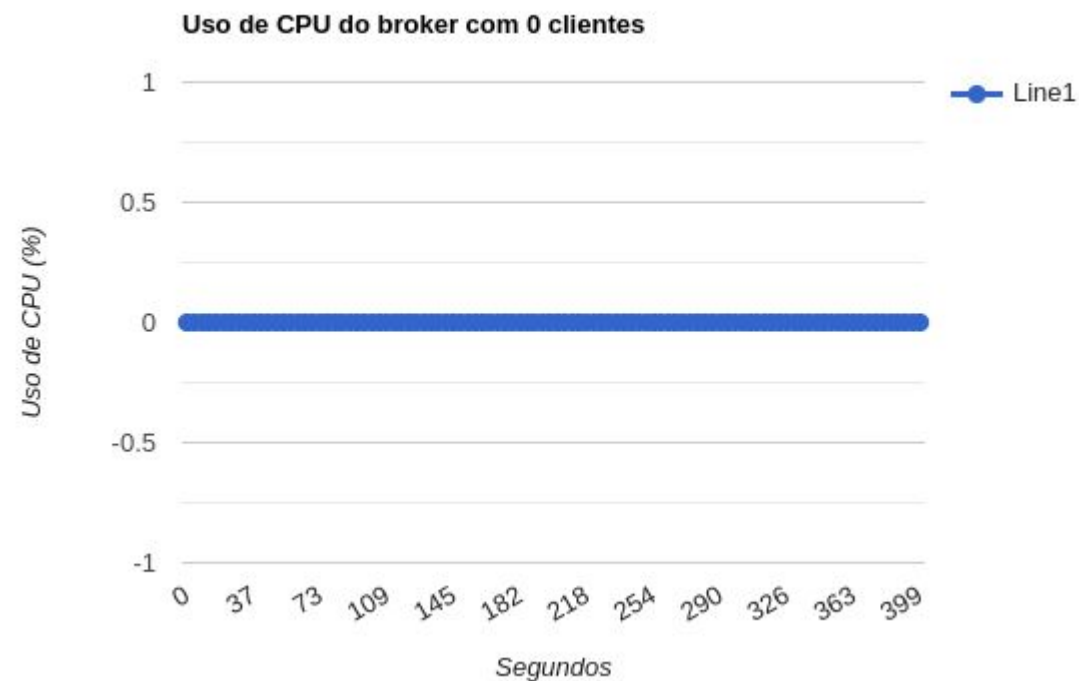
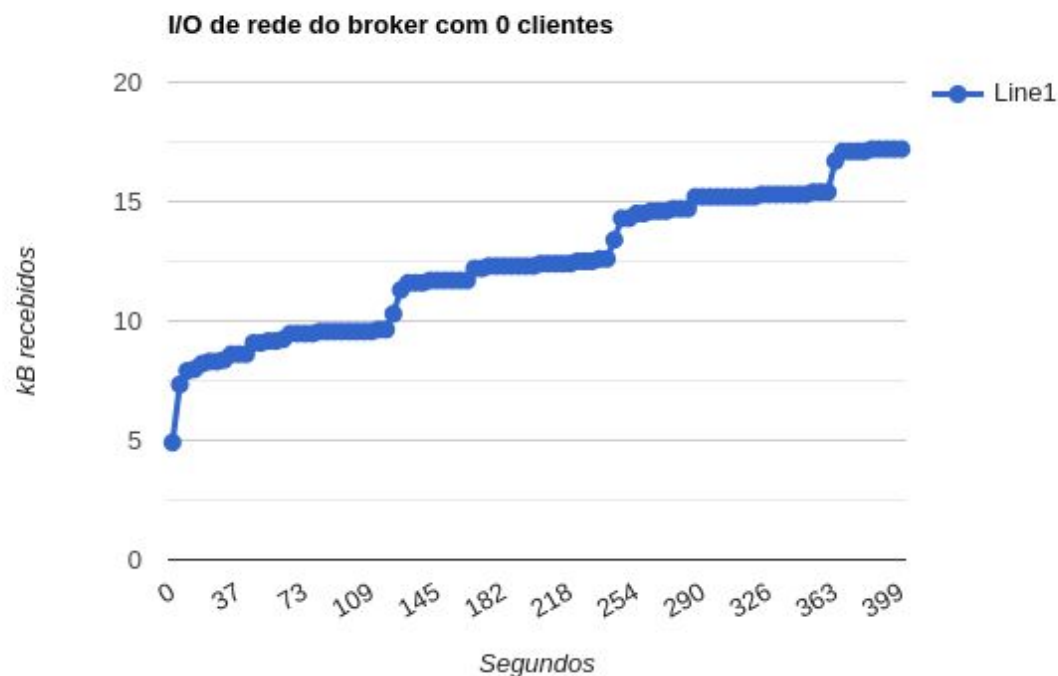
Teste:



- 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.

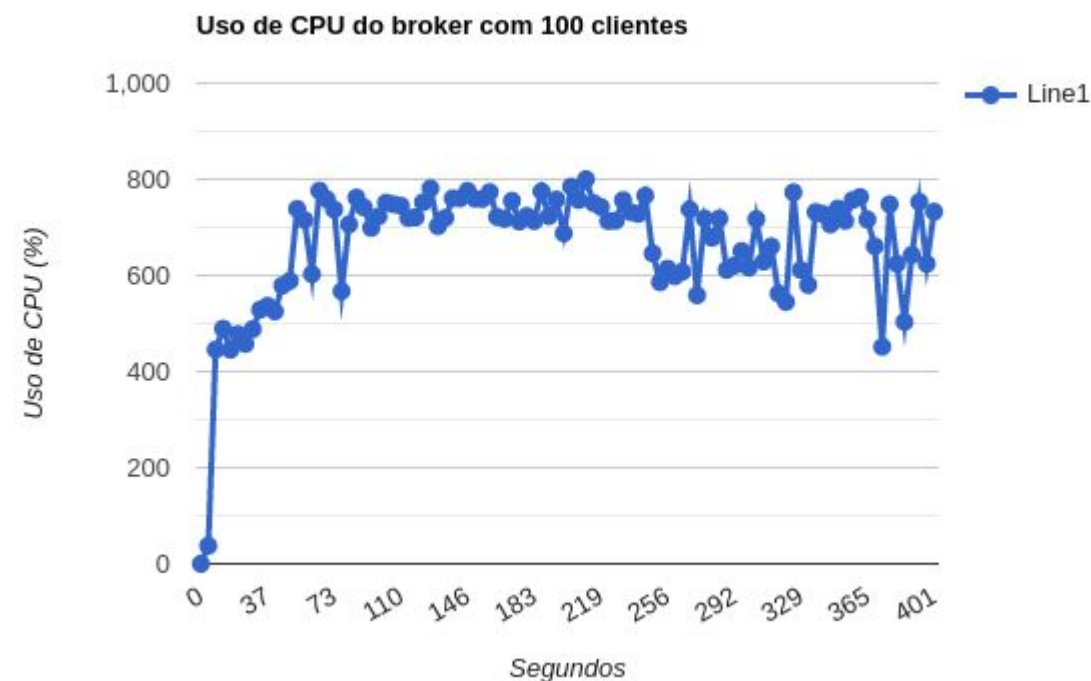
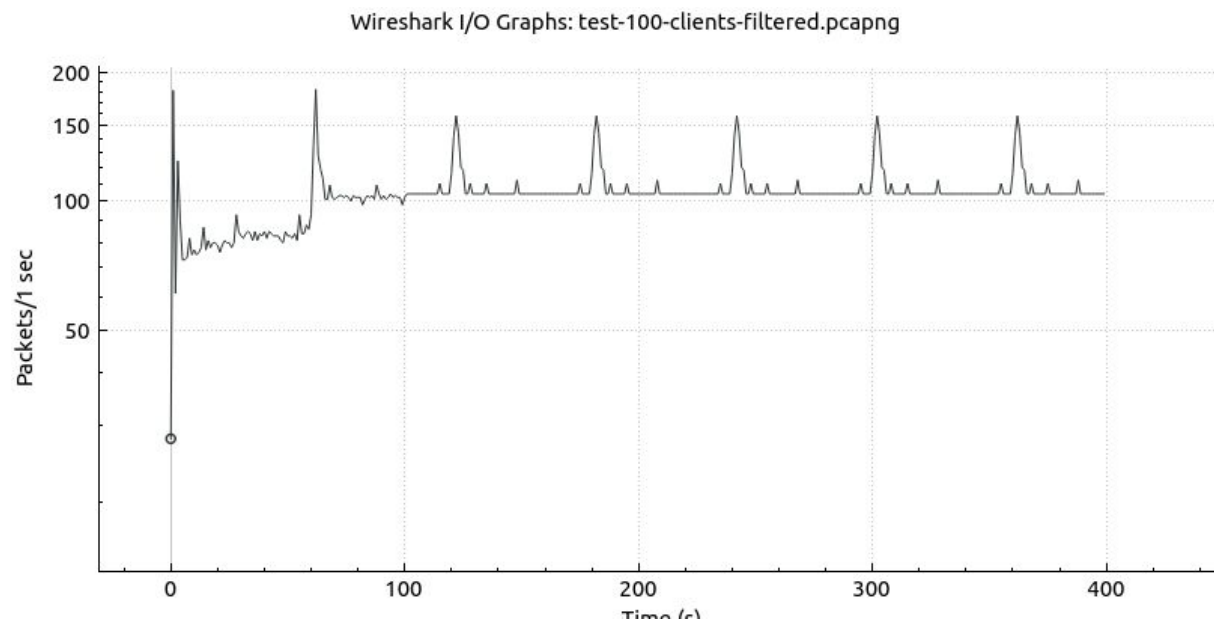


Resultados - 0 clientes:



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

Resultados - 100 clientes:



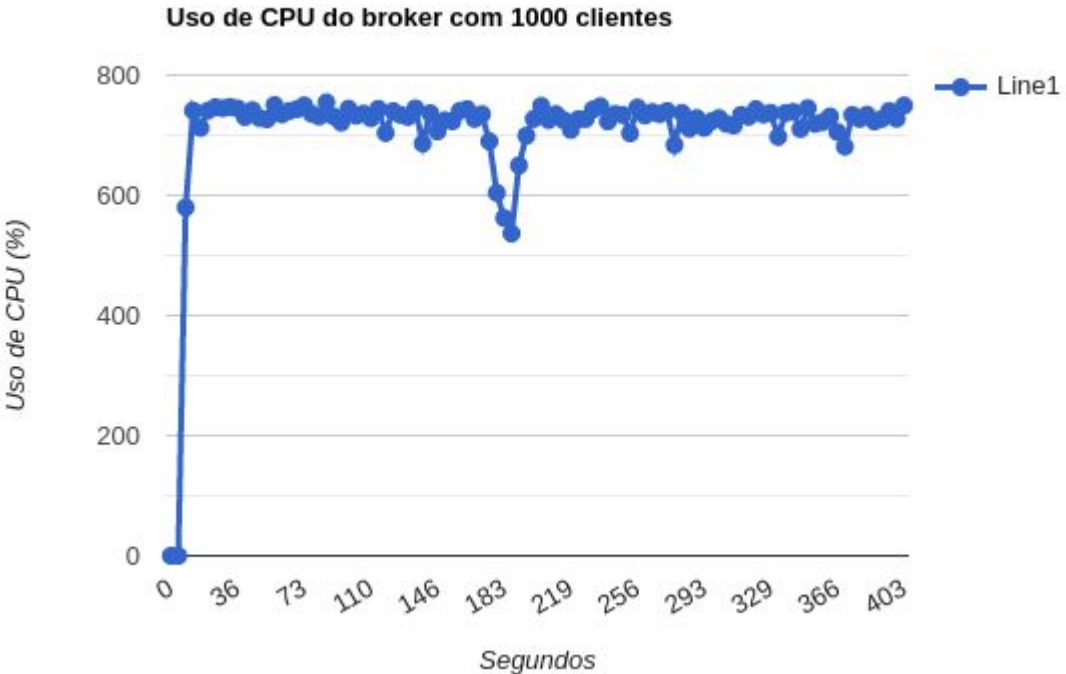
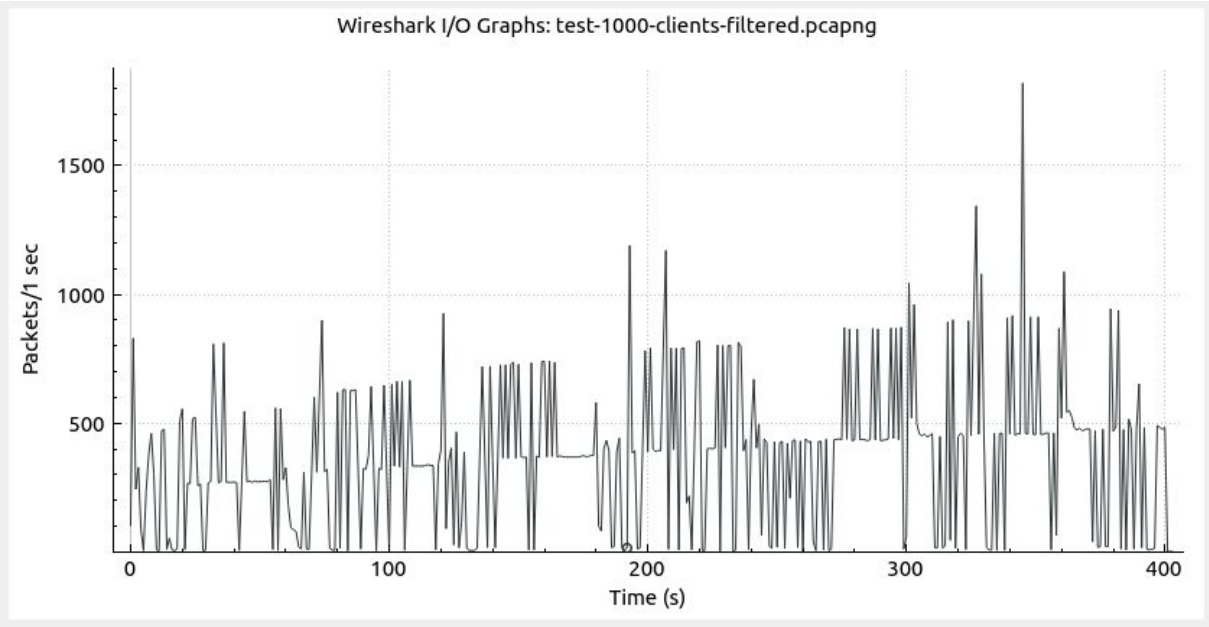
A CPU chega a 800% pois são usados 8 cores.

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	



Terminal							
CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
22ee3f1a87b3a	ep1-joao-henri-publisher	0.42%	1.43MiB / 15.49GiB	0.01%	88kB / 105kB	582kB / 0B	3
8ea58ff96c12	ep1-joao-henri-clients	2.95%	354.6MiB / 15.49GiB	2.24%	4.63MB / 4.96MB	0B / 0B	900
b2e393b7ef61	ep1-joao-henri	725.17%	1.656GiB / 15.49GiB	10.69%	5.08MB / 4.7MB	246kB / 0B	901

[illegible]

Published	message	number	150
Published	message	number	151
Published	message	number	152
Published	message	number	153
Published	message	number	154
Published	message	number	155
Published	message	number	156
Published	message	number	157
Published	message	number	158
Published	message	number	159
Published	message	number	160
Published	message	number	161
Published	message	number	162
Published	message	number	163
Published	message	number	164
Published	message	number	165
Published	message	number	166
Published	message	number	167
Published	message	number	168
Published	message	number	169
Published	message	number	170
Published	message	number	171
Published	message	number	172