EP1 - MAC0352 - Redes de Computadores e Sistemas Distribuídos

Nome: Vinicius Pereira Ximenes Frota NUSP: 11221967

Arquitetura do broker

Funções e *structs* importantes:

- Packet: *struct* que possui informações relevantes do pacote, como:
  - Tipo do pacote
  - Flags do cabeçalho fixo
  - Tamanho restante do pacote (tamanho do payload e tamanho do cabeçalho variável)
  - Cabeçalho variável
  - Payload

Funções e *structs* importantes:

- createPacket: função que recebe *bytes* e devolve um Packet correspondente, preservando todas as informações recebidas.
- createPingresPacket: função que devolve um pacote de resposta para o PINGREQ, ou seja, um pacote PINGRES.
- createConnackPacket: função que devolve um pacote de resposta para o CONNECT, ou seja, um pacote CONNACK.
- createSubackPacket: função que recebe um Packet de SUBSCRIBE e devolve um pacote de resposta, ou seja, um SUBACK. Além disso, ela faz o *parsing* dos tópicos recebidos e para cada um deles chama a função addTopic.

Funções e *structs* importantes:

- addTopic: função que cria um FIFO no sistema de arquivos correspondente ao tópico e ao PID do *Subscriber*. Ela também cria uma thread que recebe as mensagens do tópico correspondente, através do *file descriptor* do FIFO. Essa thread escreve tudo que lê no FIFO no socket do cliente.
- createNonePacket: cria um Packet sem nenhuma informação. Não tem relação com o protocolo MQTT e serve apenas para manter o fluxo do código.
- convertPacketToMessage: recebe um Packet e devolve os *bytes* correspondentes.
- publish: recebe um Packet de PUBLISH e repassa ele para os tópicos correspondentes, abrindo o diretório do tópico e escrevendo os *bytes* do PUBLISH nos arquivos de FIFO.

Fluxo do programa

Para cada nova conexão, um processo novo é criado.

Para cada sequência de *bytes* recebida, o programa cria um Packet correspondente aos *bytes* e um Packet para ser enviado ao cliente (que começa sendo um Packet vazio).

- CONNECT: se o Packet correspondente aos bytes recebidos for do tipo CONNECT, o Packet do cliente se torna um packet do tipo CONNACK.
- PUBLISH: se o Packet correspondente aos bytes recebidos for do tipo PUBLISH, o Packet do cliente se mantém vazio. O Packet PUBLISH é enviado para a função publish, que lida com as publicações conforme descrito nos slides anteriores.
- SUBSCRIBE: se o Packet correspondente aos bytes recebidos for do tipo SUBSCRIBE, o Packet do cliente se torna um packet do tipo SUBACK. Neste caso, a função createSubackPacket, responsável por lidar com cada tópico inscrito conforme descrito nos slides anteriores, é chamada.

Fluxo do programa

- DISCONNECT: se o Packet correspondente aos bytes recebidos for do tipo DISCONNECT, o Packet do cliente se mantém vazio.
- PINGREQ: se o Packet correspondente aos bytes recebidos for do tipo PINGREQ, o Packet do cliente se torna um packet do tipo PINGRESP.

  Depois de lidar com os pacotes para cada caso, o programa converte o Packet do cliente para uma sequência de bytes. Se essa sequência de bytes for uma resposta válida, ou seja, não vazia, ela é escrita no socket do cliente. Depois, o programa fica pronto para ler uma nova sequência de bytes.

Observações:

- Os arquivos de FIFO utilizados usam o seguinte caminho: /tmp/ep1/<nome do tópico>/<pid do subscriber>.falcon
- Uma lista ligada para armazenar os tópicos inscritos por um *subscriber* foi utilizada. Quando o *subscriber* é desconectado, a lista ligada é percorrida e os arquivos de **FIFO** são apagados.

Experimentos

Experimentos

Os experimentos foram feitos em uma máquina com processador Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz com 4 cores e 2 threads por core e 8GB de memória RAM.

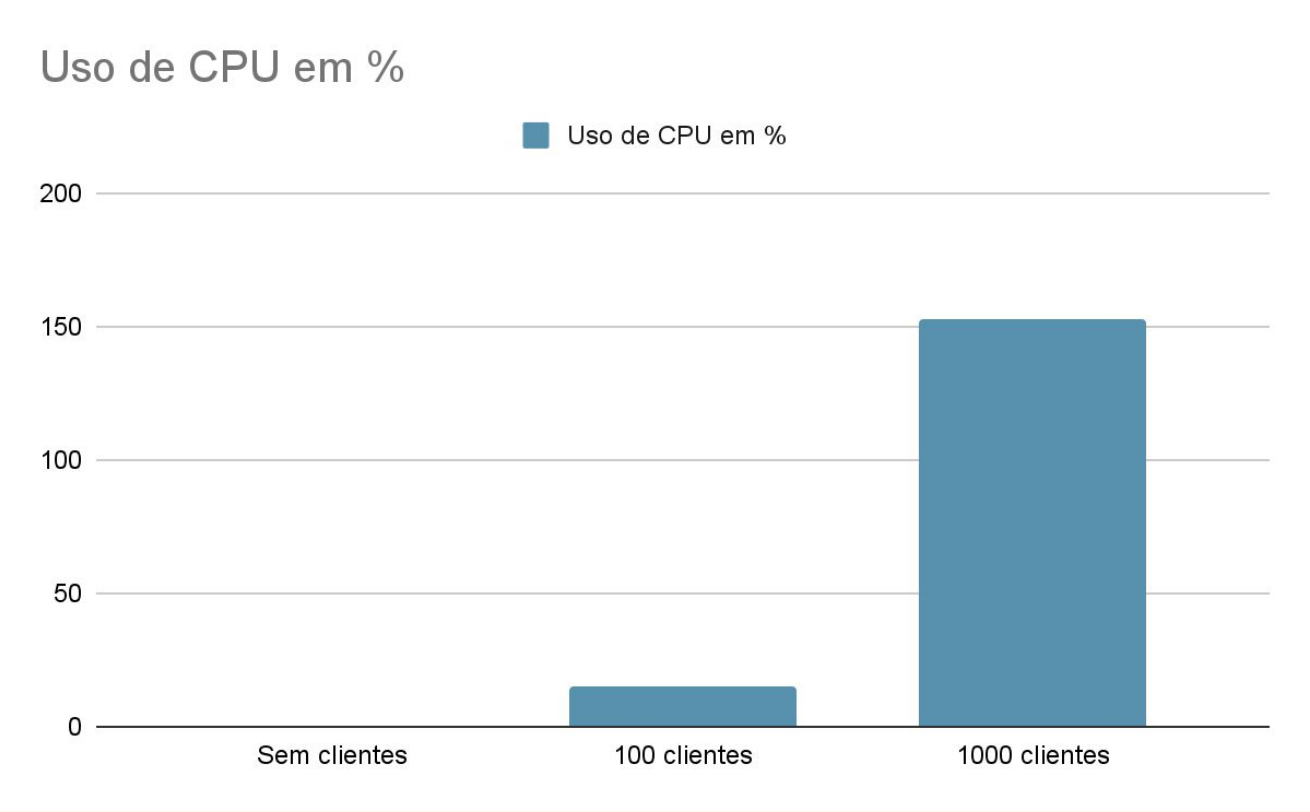
Os *publishers* e *subscribers* foram criados na máquina, enquanto o broker rodava dentro de um container do Docker. Os dados foram coletados através do comando **docker stats**. Para cada número de clientes (0, 100, 1000), foram realizadas 30 medições.

Os slides a seguir supõem que os testes formam uma distribuição normal.

O uso do CPU pode passar os 100% porque mais de um núcleo do processador pode ser usado. Pode haver transferência de Bytes para 0 clientes, por conta da comunicação da rede do Docker com a máquina.

Para a medição de **n** clientes, **n** *subscribers* foram criados paralelamente e depois **n** publishers publicaram sequencialmente.

Uso de CPU (uso máximo em cada medição)



Uso de CPU em % (uso máximo em cada medição)

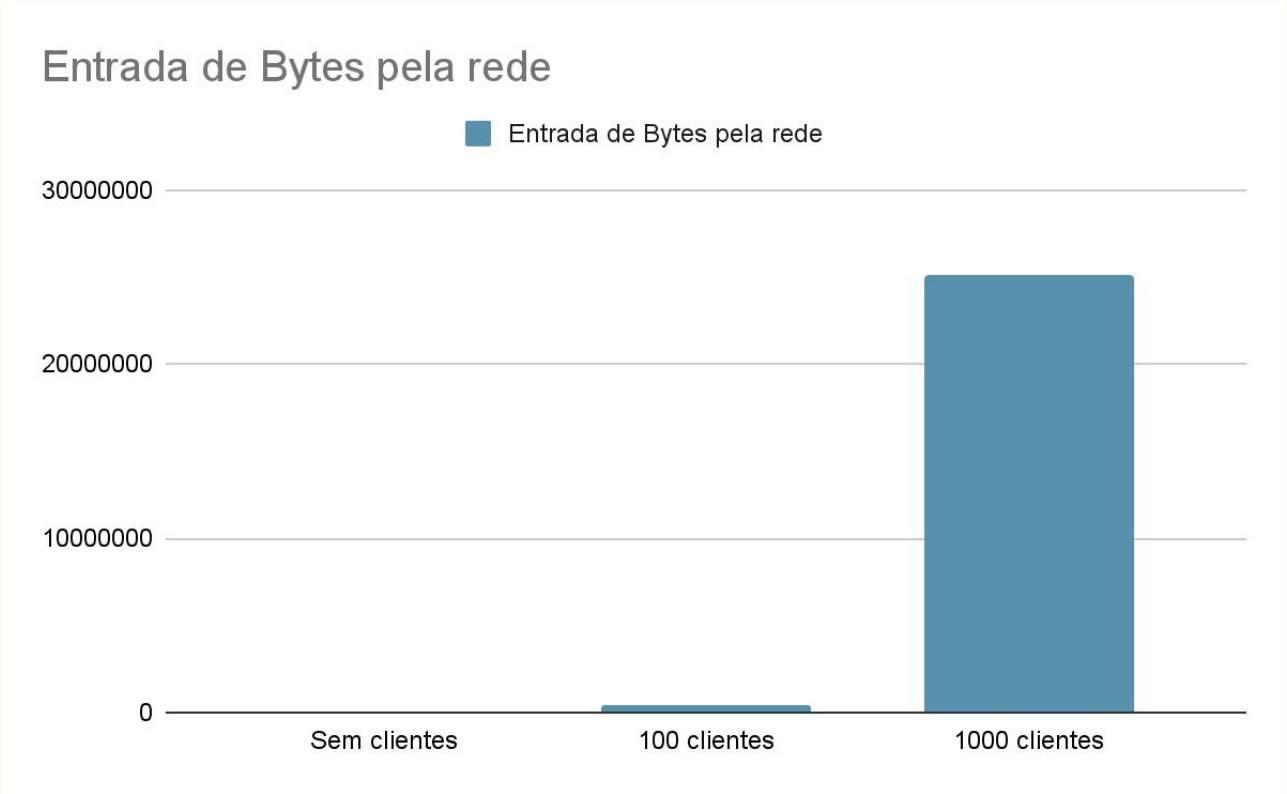
Intervalo de confiança (95%) e média:

Sem clientes - Intervalo: [0.0, 0.0], Média: 0.0

100 clientes - Intervalo: [11.519326, 18.000674], Média: 14.76%

1000 clientes - Intervalo: [146.360449, 158.913551], Média: 152.637%

Entrada de Bytes pela rede



Entrada de Bytes pela rede

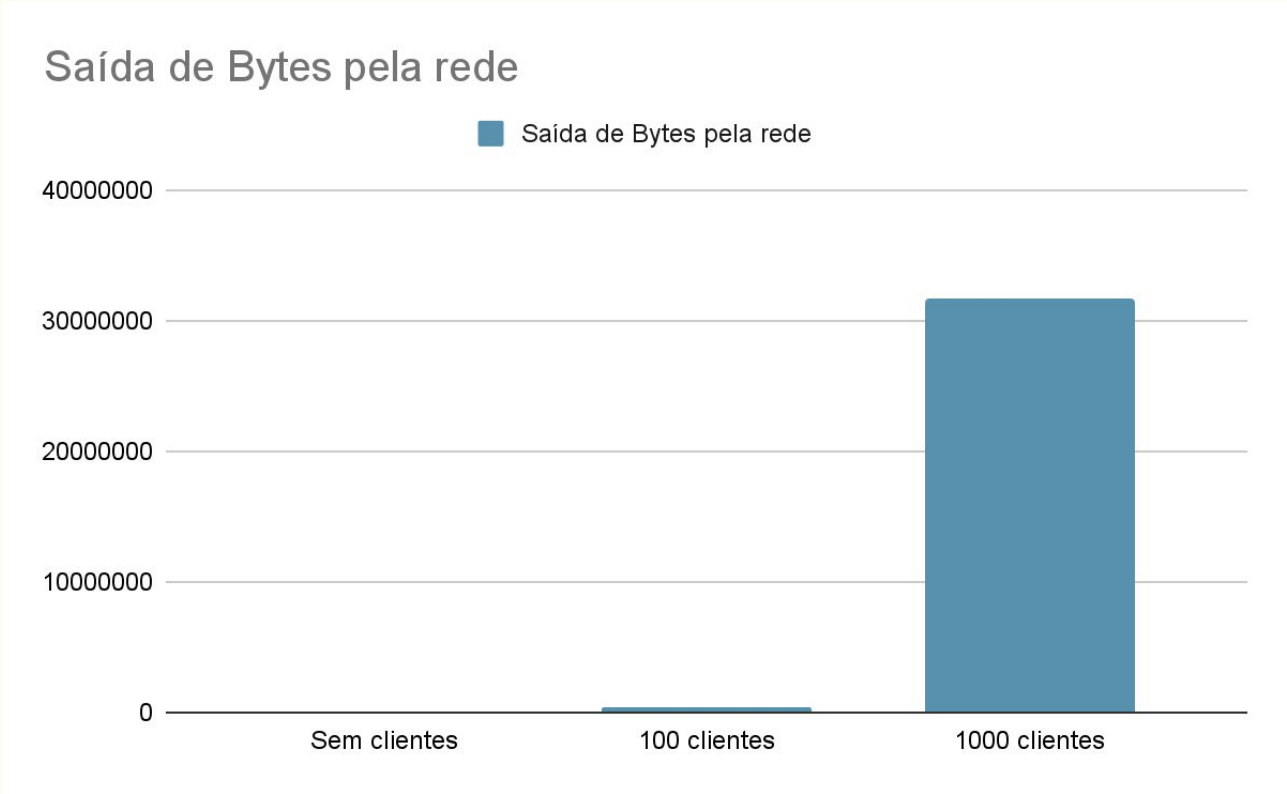
Intervalo de confiança (95%) e média:

Sem clientes - Intervalo: [8.381345, 8.953322], Média: 8.667333kB

100 clientes - Intervalo: [336.303416, 391.763251], Média: 364.033333kB

1000 clientes - Intervalo: [24.142386, 25.984280], Média: 25.063333MB

Saída de Bytes pela rede



Saída de Bytes pela rede

Intervalo de confiança (95%) e média:

Sem clientes - Intervalo: [0.0, 0.0], Média: 0.0kB

100 clientes - Intervalo: [341.733266 , 411.733401], Média: 376.733333kB

1000 clientes - Intervalo: [30.483104, 32.916896], Média: 31.700000MB