

AMELIA

Advanced Message Queueing
Protocol by Sabrina Araujo

Amelia é um servidor de interpretação e o processamento de mensagens da camada de aplicação de um servidor AMQP na versão 0.9.1 do protocolo.

Também é um acrônimo de “Advanced Message Queueing Protocol by Sabrina Araujo” :)

IMPLEMENTAÇÃO DO SERVIDOR



1. Início da conexão

A função **sendProtocolHeader()** é responsável pelo processo de negociação, ela envia o cabeçalho do protocolo e permite o início da conexão.

2. Leitura do frame

Após o processo de negociação, o servidor lê o frame enviado pelo cliente pela função **readAMQPFrame()**. Esse frame contém informações como `type`, `channel`, `size`, `class_id` e `method_id`.

3. Interpretando o frame

A partir das informações do frame, o servidor usa a função **AMQPConnection()** no arquivo **amqp.c** para interpretar o class_id e o method_id e definir qual operação o cliente está solicitando.

```
switch (class_id) {
    case CONNECTION:
        switch (method_id){
            case CONNECTION_START_OK:
                printf("[CLIENT] CONNECTION_START\n");
                read(connfd, recvline, size);
                printf("[SERVER] CONNECTION_START\n");
                write(connfd, PACKET_CONNECT);
                break;
            case CONNECTION_TUNE_OK:
                printf("[CLIENT] CONNECTION_TUNE\n");
                read(connfd, recvline, size);
                printf("[SERVER] CONNECTION_TUNE\n");
                write(connfd, PACKET_TUNE);
                break;
            case CONNECTION_CLOSE:
                printf("[CLIENT] CONNECTION_CLOSE\n");
                read(connfd, recvline, size);
                printf("[SERVER] CONNECTION_CLOSE\n");
                write(connfd, PACKET_CLOSE);
                break;
        }
}
```

4. Envio dos pacotes

Para enviar pacotes de rede (exceto nos casos `QUEUE_DECLARE` e `BASIC PUBLISH`), o código utiliza pacotes predefinidos obtidos a partir do Wireshark. Esses pacotes estão no arquivo `packets.c`.

5. Estrutura de dados

Quando o cliente solicita `QUEUE_DECLARE`, a estrutura de dados **queue** é utilizada para representar informações sobre uma fila no broker.

6. Persistência dos dados

Para a persistência dos dados na estrutura **queue** entre diferentes operações, foram utilizadas funções de alocação e liberação de memória compartilhada.

7. Construção do pacote

O pacote QUEUE_DECLARE_OK, enviado pelo servidor ao cliente, é construído na função **queuePacket()** em packets.c com base nas informações do frame.

8. Publicação da mensagem

Quando o cliente solicita a operação BASIC PUBLISH, o servidor AMQP executa a função **publishMethod()**, que lê a mensagem e a fila correspondente e armazena na estrutura ‘queue’.

10. Round Robin

9. Consumo da mensagem

Quando o cliente solicita a operação BASIC CONSUME, o servidor AMQP executa a função **consumeMethod()**, que lê a fila correspondente, ordena o ‘consumer’ de acordo com o round robin e manda o pacote de ‘consume’ para o cliente.

A função **moveConsumer()** mantém uma ordem circular, garantindo que cada ‘consumer’ tenha sua vez de processar mensagens.

11. Envio da mensagem

A função **consumePacket()** em **packets.c** monta o pacote que contém os dados da mensagem e envia ao ‘consumer’.

ANÁLISE DE DESEMPENHO



Ambiente computacional

- Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
- Ubuntu 20.04.4 LTS

Rede

- Ethernet controller: Intel Corporation Ethernet Connection (4) I219-LM (rev 21)

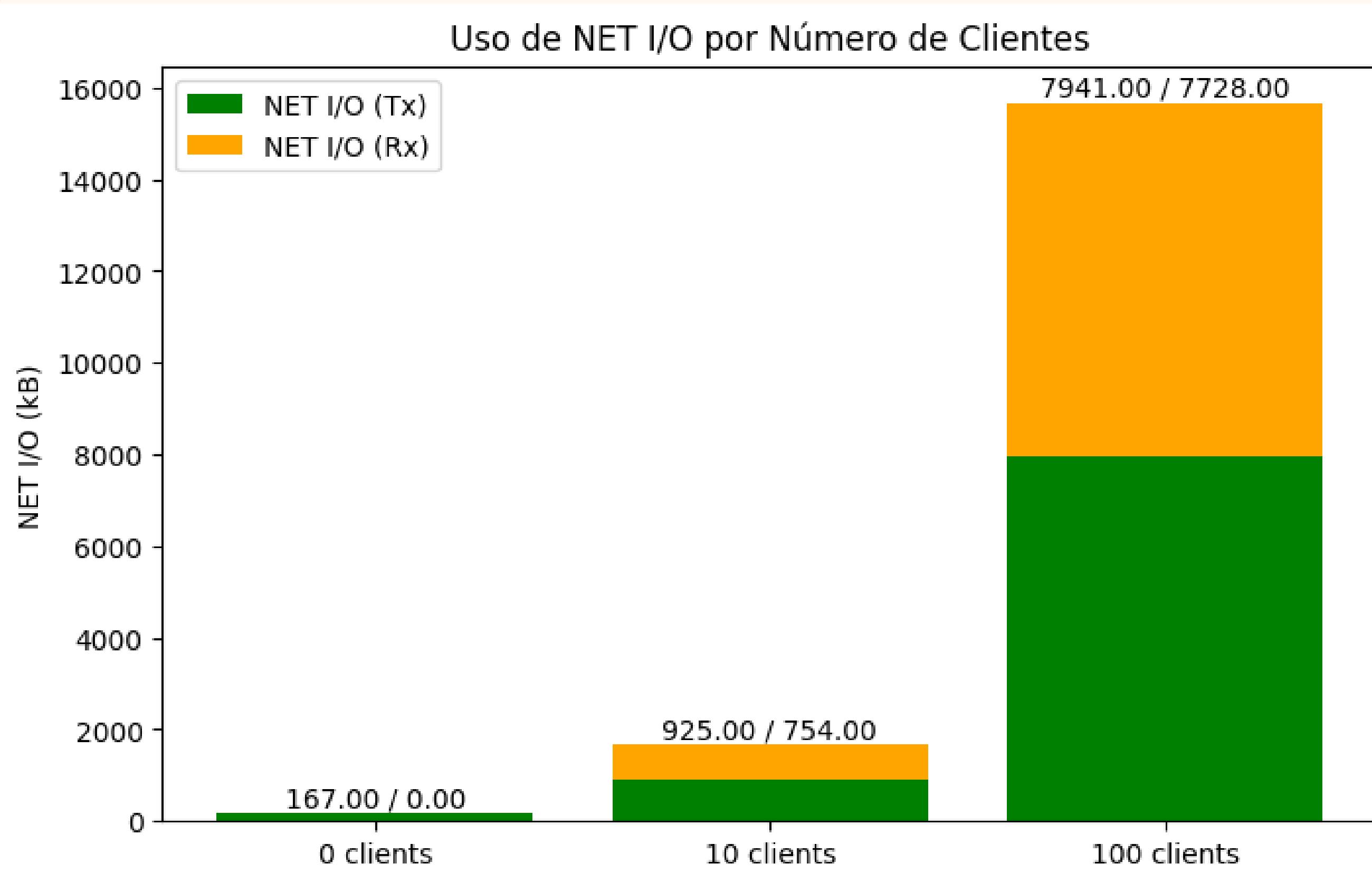
Experimentos

Os experimentos dos 3 cenários foram feitos utilizando uma rede virtual em um contêiner do Docker.

Medições

As medições do uso de rede e de CPU foram feitas pelo docker stats.

USO DE REDE



O aumento no número de clientes resulta em um aumento proporcional no tráfego de rede, com uma ênfase na transmissão de dados do servidor para os clientes.

USO DE CPU

Os resultados mostram que o uso de CPU aumentou à medida que o número de clientes aumentou

