BROKER MQTT v.3.1.1

Henrique de Carvalho

IMPLEMENTAÇÃO

Código Fonte

A lógica principal do broker está ditada em 3 blocos de funções, dividida nos arquivos:

a. ***broker.c*** - contém o loop principal do broker
b. ***session.c*** - contém as funções que manejam os dados internos de uma sessão entre o cliente e o broker
c. ***handlers.c*** - contém as funções que manejam as mensagens entre cliente e o broker

Funções

As funções encontradas nesses arquivos permitem entender a estrutura básica do programa. As funções podem ser agrupadas por finalidade. Tem-se as funções de:

- **Inicialização dos dados** `{struct}_init`
- **Deserialização de um pacote** `deserialize_{parte de um pacote}`
- **Validação de uma mensagem** `validate_{parte de um pacote}`
- **Manejo de pacotes** `handle_{pacote}`
- **Envio de uma mensagem** `send_{mensagem}`
- **Verificação de um dado** `verify_{dado de uma sessão}`

Detalhe de implementação. O controle do fluxo entre as funções é realizado por meio de um **código de retorno** (`rc_t`). É dessa forma que as camadas externas do programa sabem como lidar com o controle de fluxo.

Estruturas de Dados

O fluxo de dados é realizado com o auxílio de `session_data` e `control_packet`. Essas estruturas armazenam informações que são essenciais a todas as funções de manejo da sessão e das mensagens.

```c
struct session_data {
    /* Dados da conexão */
    int                 connfd;
    bool                connected;  /* se um MQTT CONNECT foi estabelecido */
    bool                session_present;

    /* Dados do tópico em que se publica ou em que se inscreve */
    FILE                *topic;
    char                *topicfp;
    char                *topicname;
    uint16_t            topicnamelen;

    struct              control_packet *packet;
};
```

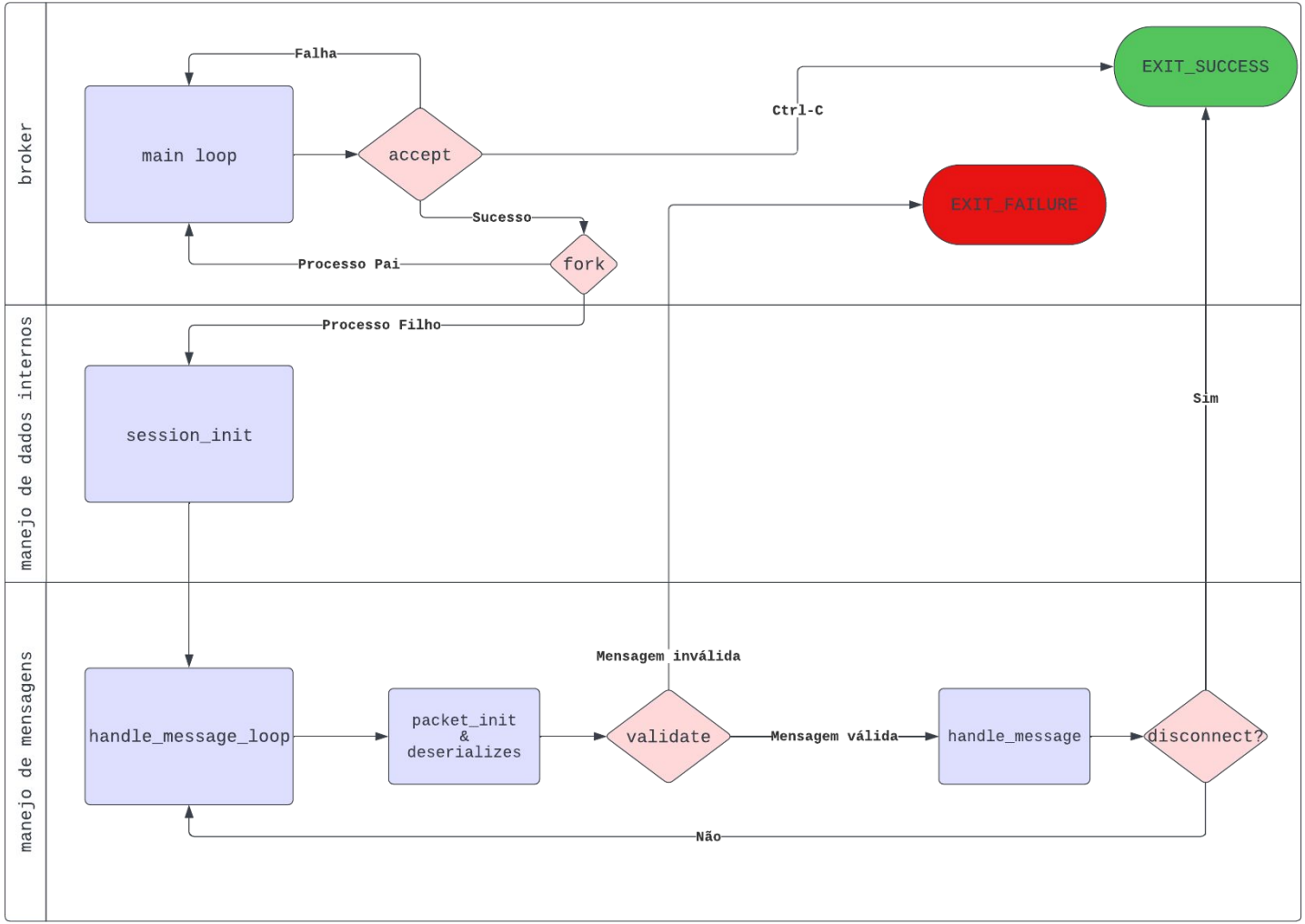
```c
struct control_packet {
    uint8_t type;
    uint8_t fixed_header_flags;
    ssize_t remaininglen;
    ssize_t payloadlen;

    /* Flags de connect */
    uint8_t dup;
    uint8_t qos;
    uint8_t retain;
    uint8_t clean_session;

    int8_t    *fixed_header;
    int8_t    *variable_header;
    int8_t    *payload;
    char      *payloadmsg;
};
```

FLUXO DO PROGRAMA

Fluxo do programa

Laço principal. Em toda conexão com o broker, este:

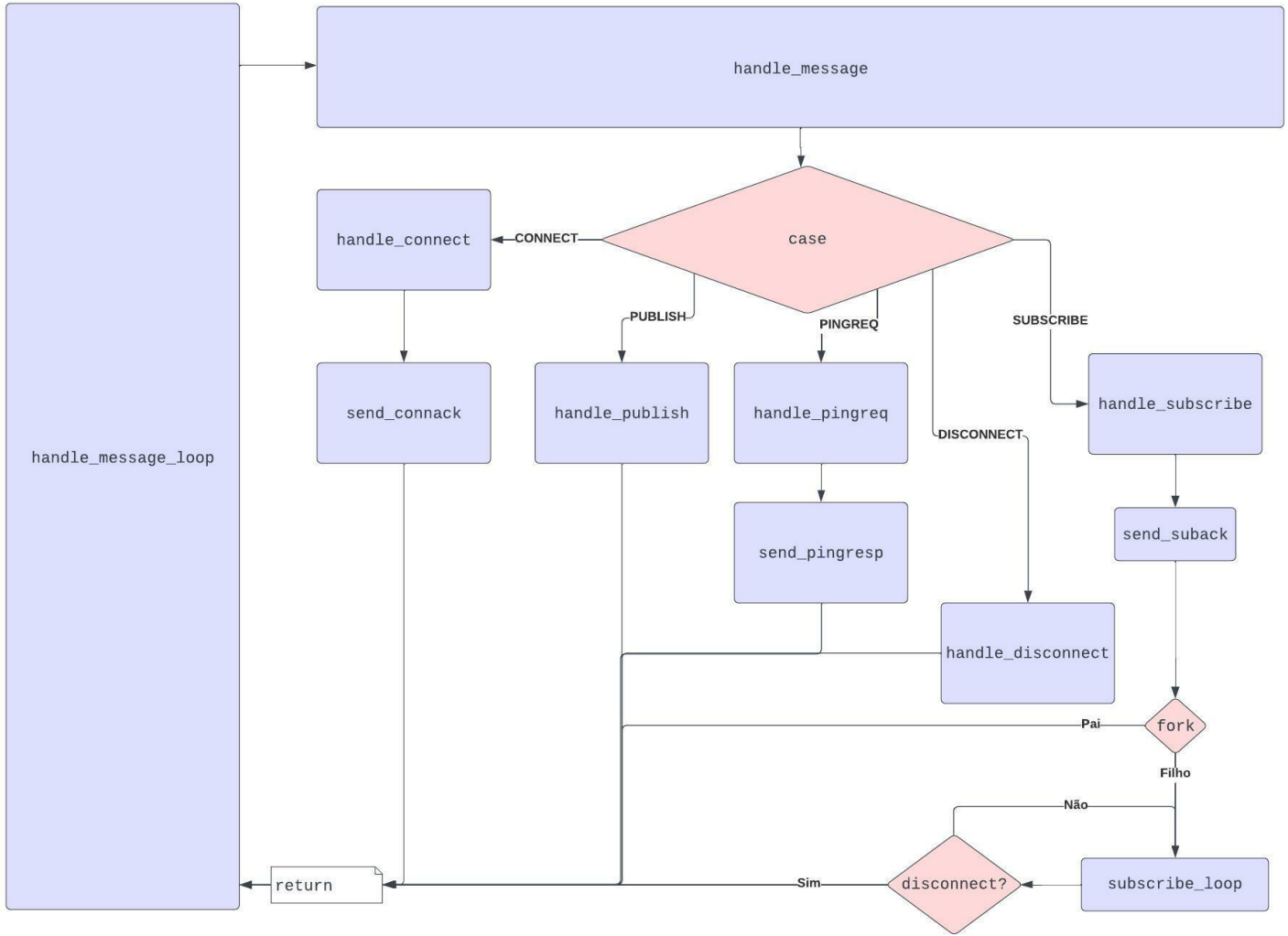
1. **cria um processo filho** responsável pela **sessão**;
2. este processo entra no **loop** de manejo de mensagens;
3. **deserializa** uma mensagem recebida em um pacote e a valida;
4. realiza **manejo** sobre a mensagem.



Fluxo de manejo de mensagens

Manejo. O sistema de manejo de mensagens lida com CONNECT, PINGREQ, DISCONNECT, SUBSCRIBE e PUBLISH.

Inscrição e publicação. O sistema SUBSCRIBE PUBLISH funciona por meio da escrita e leitura de arquivos. Nesta implementação, **um tópico é um arquivo em uma pasta temporária** (ver LEIAME).



ANÁLISE DE PERFORMANCE

Análise de Performance

A análise de performance foi dividida em:

a. **Análise do consumo de CPU:** a coleta de dados para consumo da CPU foi realizada por meio do comando `ps`, que retorna o valor %CPU, que se refere à quantidade de utilização de tempo da CPU dividido pelo tempo.

b. **Análise do consumo de rede:** a coleta de dados para consumo de rede foi realizada por meio do programa `nethogs`, que fornece a taxa de transferência dos dados recebidos e enviados em Kb/s. Como as medidas foram realizadas a cada segundo, pode-se olhar para esses dados como se fossem a quantidade de bytes transferidos naquele segundo.

Análise de Performance

As coletas dos dados foram realizadas em 3 cenários, 20 vezes cada (entende-se que 20 vezes é uma quantidade razoável de medidas que possibilita generalizar as conclusões obtidas):

a. **0 *subscribers* e 0 *publishers*:** a coleta foi realizada com o servidor aceitando conexões por 1 minuto, sem haver qualquer conexão.

b. **100 clientes: 50 *subscribers* e 50 *publishers*:** a coleta foi realizada com o broker aberto para conexões por meio da internet e os clientes foram criados em outro computador, por meio de forks sucessivos, divididos em **5 tópicos**. Primeiro foram criados os *subscribers* e depois foram criados os *publishers*.

c. **1000 clientes: 500 *subscribers* e 500 *publishers*:** realizada da mesma forma que o item acima (**b.**)

Análise de Performance

O computador que rodou o broker tem a seguinte configuração:

- **CPU:** AMD Ryzen 7 5800H with Radeon Graphics

- **Interface de Ethernet:** RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller. Realtek Semiconductor Co., Ltd.

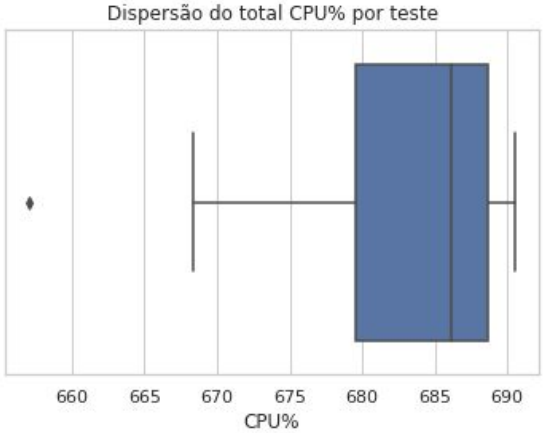
1. Zero clientes - 0 inscritos e 0 publicadores

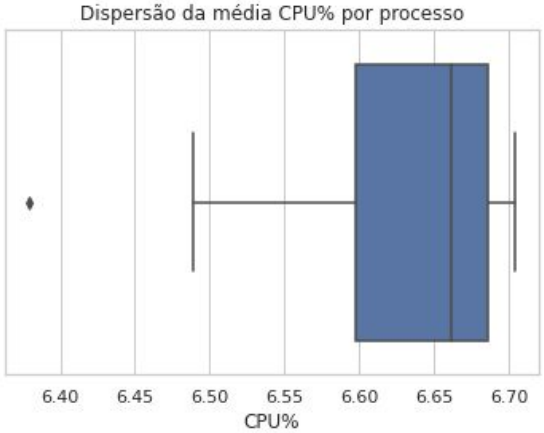
CPU. O primeiro cenário não resultou em um consumo de CPU% maior que 0.0, de modo que pode-se considerar que o broker performa bem quando não recebe requisições. Como a soma de todo o consumo de CPU foi 0.0, não há dados para visualizar.

Rede. Também, como se esperava, não há consumo de rede quando não existem conexões ao broker e, portanto, não há dados para visualizar.

2.1. Cem clientes - 50 inscritos e 50 publicadores

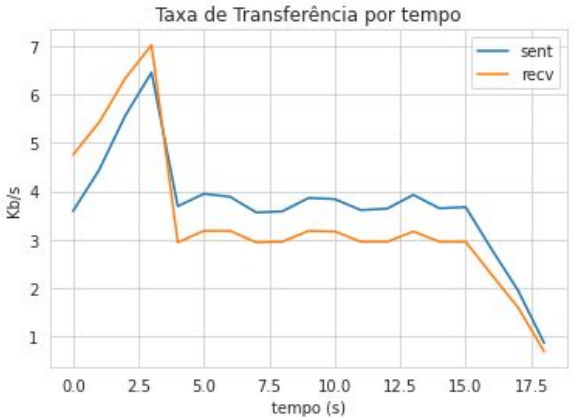
CPU. O primeiro gráfico representa a dispersão, para todos os testes, do consumo total de CPU% por execução e o segundo representa a dispersão, para todos os testes, do consumo médio de CPU% por processo. **Apesar de poucos clientes, o consumo de CPU parece substancial. Acredita-se que isso se deve à grande quantidade de processos lidando com leitura e escrita de arquivos tópicos. Nota-se que há pouca variabilidade (respectivamente, σ=9.04 e σ=0.08).**

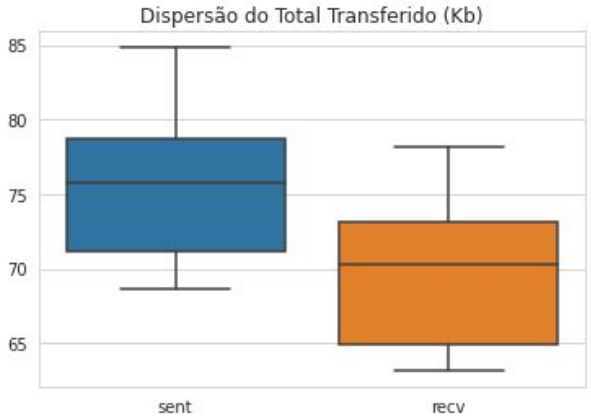


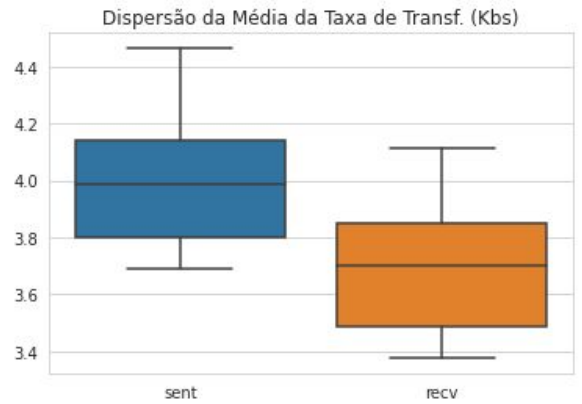


2.2. Cem clientes - 50 inscritos e 50 publicadores

Rede. Os gráficos abaixo comprovam que **a carga na rede desse cenário é baixa**, até porque a quantidade de dados transferida foi baixa – foram transferidos pequenos textos para poucos inscritos. Nota-se que **a quantidade de bytes enviados é, em geral, maior que a recebida**, principalmente quando o broker passa a enviar as mensagens recebidas aos inscritos.

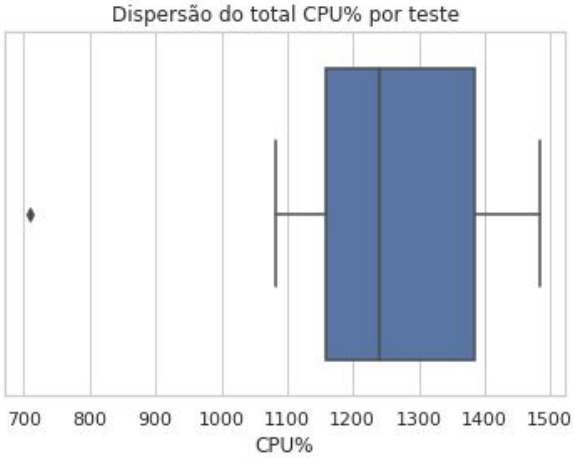


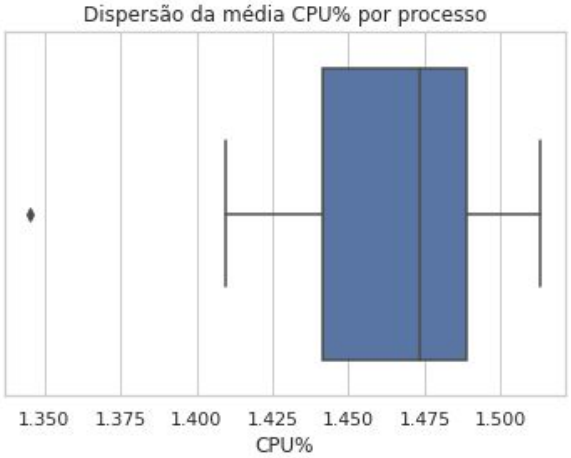




3.1. Mil clientes - 500 inscritos e 500 publicadores

CPU. Como explicado acima, para cada subscribers, 2 processos são criados para administrar a sessão, o que explica os gráficos de dispersão serem similares. Entende-se que os gráficos comprovam que **existe um consumo alto de CPU e que ele se deve às leituras sucessivas realizadas pelo processo que lê o tópico para enviá-lo ao cliente. Igualmente aqui, nota-se que há pouca variabilidade (respectivamente, σ=264.653723 e σ=0.058490).**





3.2. Mil clientes - 500 inscritos e 500 publicadores

CPU. Da mesma forma que no cenário anterior, **a média taxa de transferência de envio é maior que a de recebimento**, mas isso começa a ocorrer apenas quando os publicadores passam a dominar o envio. A quantidade total de bytes enviados também é maior. Nota-se que, pela maior quantidade de bytes transferidos aqui, **há uma maior diferença entre as médias de bytes enviados e recebidos**. Apesar disso, **a carga na rede continua sendo baixa**, se considerada a quantidade de bytes enviados e recebidos e a taxa de transferência tomada.