



EP 1 - MAC 0352

REDES DE COMPUTADORES E

SISTEMAS DISTRIBUÍDOS

Bruno Mazetti Saito

11221838



Roteiro

- Estrutura do programa e decisões de projeto gerais;
- Leitura dos pacotes
- Implementação dos pacotes
- Testes
- Gráficos
- Conclusões



Estrutura do programa e decisões de projetos gerais

- Foi utilizado como base o programa fornecido no e-disciplinas que não usava a implementação com pipes;
- Biblioteca adicional (*auxiliar.h*) com pequenas funções úteis durante o programa:
 - *void toBit(char c, int v)* - Transforma o caractere *c* em um byte e armazena no vetor *v*;
 - *void calculaBit(int byte[], int limite)* - Lê os elementos do vetor de bits *byte* e retorna o seu respectivo valor numérico;
 - *void leStringPacote(int ini, int fim, char recvline[], char *topico)* - Lê os bits de um intervalo do pacote recebido e devolve a string interpretada;
- *Switch case* separando a implementação para cada tipo de pacote recebido;
- Para cada tópico foi criado um arquivo com mesmo nome na pasta */tmp/*;
- Erros inesperados ocorrem algumas vezes para tópicos com mais de 110 caracteres no nome.



Leitura dos pacotes

- Os pacotes foram lidos pela função *read* e armazenados em um vetor;
- Leitura do cabeçalho fixo de cada pacote:
 - A partir do primeiro byte lido, assim como é informado no site da OASIS, torna-se possível identificar qual tipo de pacote está sendo enviado pelo cliente;
 - Com a interpretação do segundo byte podemos descobrir o tamanho inteiro do pacote recebido;



Implementações para os pacotes e decisões de projeto específicas

- CONNECT;
- PUBLISH;
- SUBSCRIBE;
- DISCONNECT



CONNECT

- Envio do pacote CONNACK sendo a única ação necessária a ser feita;
- Pacote Constituído de 4 bytes:
 - O primeiro byte tem os 4 primeiros bits para identificar o pacote CONNACK e os últimos bits são reservados por padrão, portanto, o byte é constante (0010 0000);
 - Segundo byte contém a quantidade de bytes restantes. Para este programa, assume-se que não haverá inscrição/publicação em mais de um tópico, por tal motivo, este byte sempre vale 00000010 (indicando 2 bytes restantes);
 - Terceiro byte é reservado, com o último bit indicando a flag '*Session Present*' igual a 0, já que não foi considerado o caso que a flag seja verdadeira (00000000);
 - Quarto byte é um código de retorno de sucesso para a tentativa de conexão;
- Por fim o header descrito acima é enviado para o cliente pelo comando `write()`.



PUBLISH

- Primeiramente é identificado o tópico da publicação por meio da função *leStringPacote()* da biblioteca auxiliar;
- Com a string contendo o tópico já interpretado, a função *open()* abre o respectivo arquivo, ou o cria caso não exista, para escrita no final deste;
- Após a abertura do arquivo é feita uma trava para que outros processos não possam escrever;
- Depois do processo de escrita, o arquivo é desbloqueado para futuras leituras ou escritas;
- O pacote PUBACK não é enviado por que não foi considerado o caso em que o nível QoS é maior que zero, similarmente, não foram tratados os casos em que as flags DUP e RETAIN são diferentes de 0.



SUBSCRIBE

- Primeiramente é montado o pacote SUBACK analogamente ao CONNACK
 - Primeiro byte constante -> 1001 0000 (Identificação do pacote)
 - Segundo byte constante -> 00000011 (3 bytes restantes)
 - Terceiro e quarto bytes são apenas copiados dos respectivos índices do pacote SUBSCRIBE recebido (indicam o comprimento da string do tópico);
 - Quinto byte é um código de retorno indicando o sucesso para cada tópico especificado pelo cliente (apenas um código retornado devido a suposição de associação de cada cliente a um único tópico);
- Em seguida o tópico em questão é identificado e o respectivo arquivo é aberto/criado para futuras leituras;
- O processo fica em um loop infinito esperando por futuras escritas, até que o pacote escrito no arquivo pelo processo PUBLISH é lido e enviado ao cliente.



DISCONNECT

- Para este pacote a única ação feita durante o programa foi o fechamento do socket do cliente.



Testes

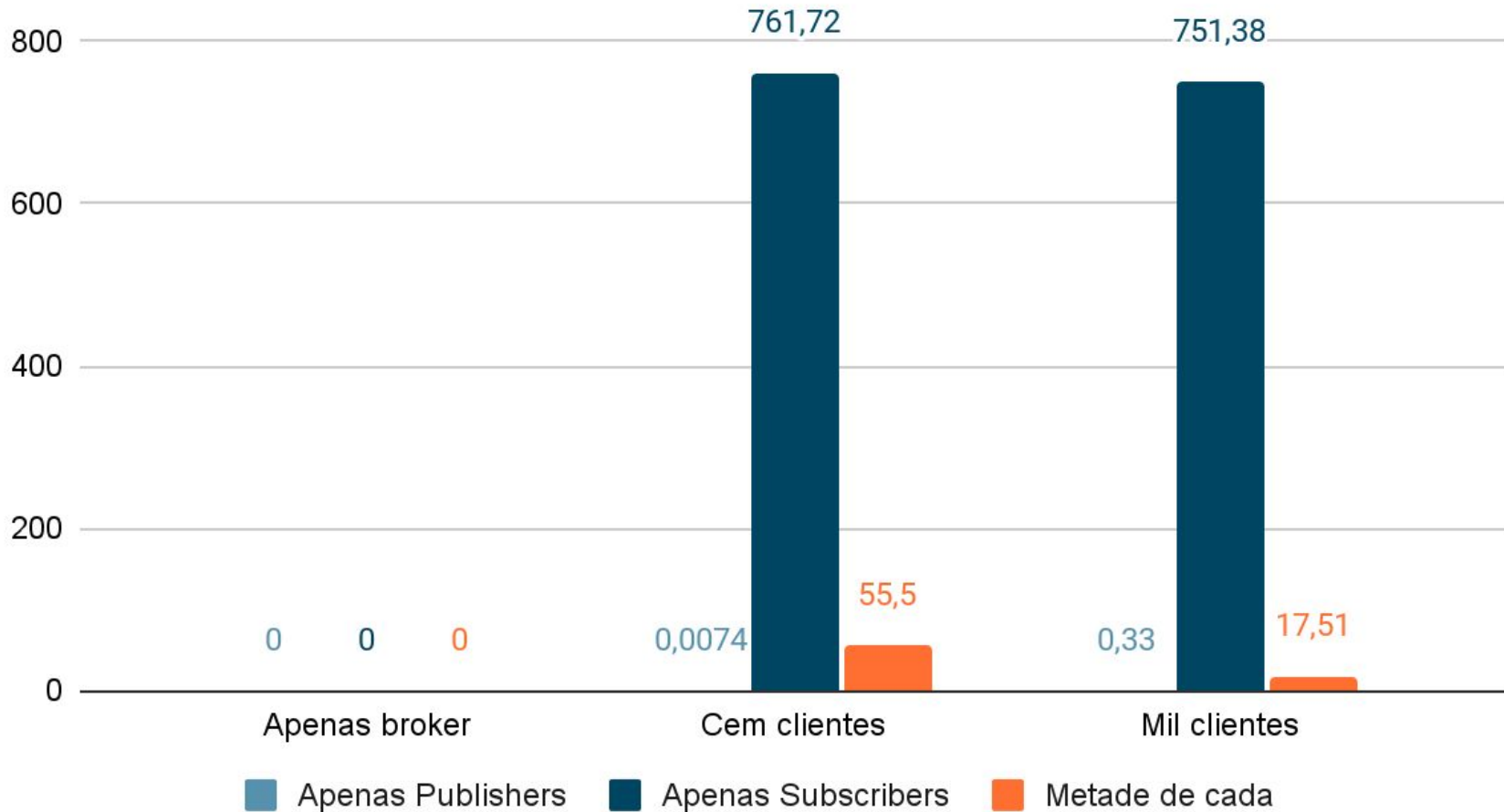
- Os testes de uso de rede e de CPU foram feitos nos seguintes cenários:
 - Apenas o broker está rodando sem nenhum cliente conectado;
 - Com clientes conectados ao broker simultaneamente em três cenários:
 - 100 subscribers;
 - 100 publishers;
 - 50 subscribers e 50 publishers;
 - Mil clientes conectados ao broker simultaneamente em três cenários:
 - 1000 subscribers;
 - 1000 publishers;
 - 500 subscribers e 500 publishers.



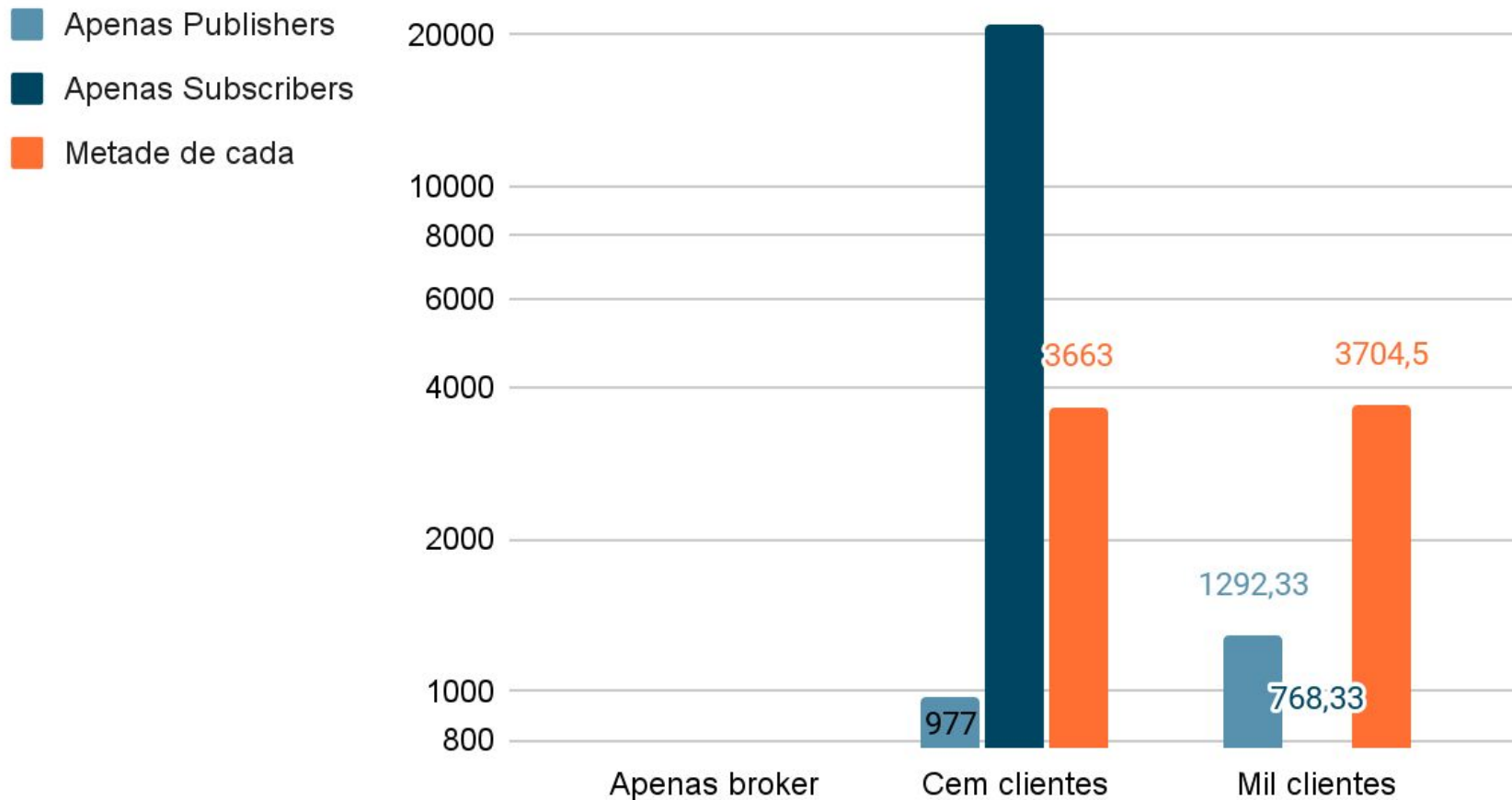
Testes

- Cada teste foi simulado 10 vezes;
- A simulação foi feita rodando o broker no seguinte ambiente:
 - Processador i7-8550U: 4 cores com 8 threads;
 - Placa de rede Realtek RTL810xE PCI Express Fast Ethernet;
- Os clientes foram executados em containers docker por meio de um script;

Consumo de CPU (%)

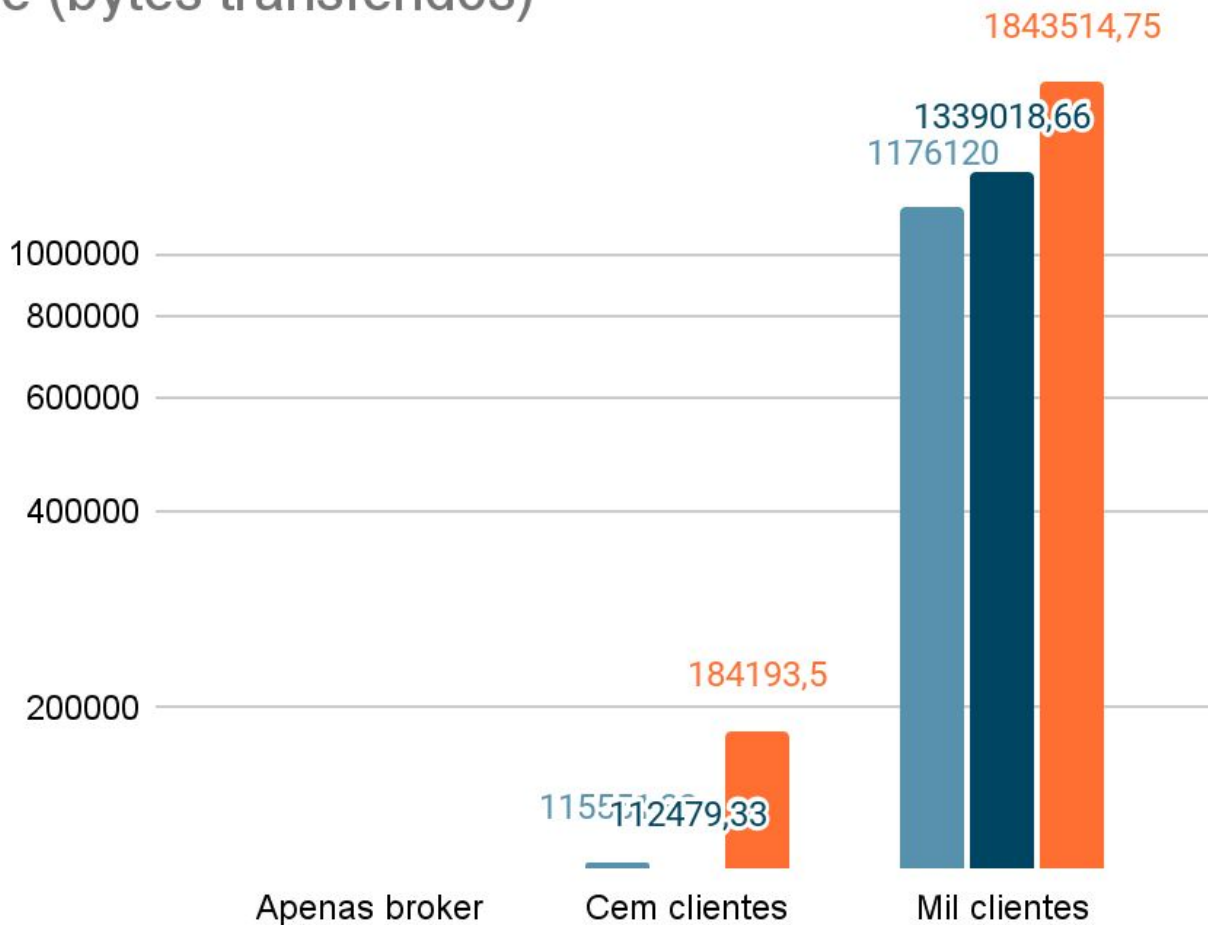


Consumo de rede (bytes/s)



Consumo de rede (bytes transferidos)

- Apenas Publishers
- Apenas Subscribers
- Metade de cada





Desvio Padrão - Cem clientes

	Apenas clientes	Apenas subscribers	Metade de cada cliente
Consumo de CPU	0.005	7.68	3.75
Consumo de rede (bytes/s)	139.06	88.6	100.51
Consumo de rede (bytes)	7451.79	3397.79	3951.29



Desvio Padrão - Mil clientes

	Apenas publishers	Apenas subscribers	Metade de cada cliente
Consumo de CPU	0.04	2.28	3.45
Consumo de rede (bytes/s)	31.21	5057.12	19472.19
Consumo de rede (bytes)	6107.29	135.55	59.10



Conclusão

- A partir dos gráficos podemos tirar as seguintes conclusões:
 - O consumo de CPU por parte dos *publishers* é praticamente insignificante comparado com os *subscribers*;
 - A sobrecarga da CPU na execução dos publishers se dá pelo loop infinito que sempre tenta ler um pacote;
 - Em relação à transferência de bytes, os resultados coincidem com o esperado, que mostram a execução de ambos os clientes tendo uma maior quantidade de bytes transferidos.