EP1 – MAC0352

Ana Yoon Faria de Lima
NUSP: 11795273

Explicitações

Nesse EP, foram aplicados conceitos aprendidos na matéria MAC0352, além dos aprendidos lendo a documentação do protocolo MQTT. Depois de testes feitos com a implementação real do `mosquitto`, foi desenvolvido uma implementação simplificada, o que foi muito útil para entender como comunicações fora da máquina local são feitas. Conforme explicitado no enunciado, não foram feitos diversos tratamentos de falhas nem implementações de autenticação ou criptografia.

Como código base desse EP, foi utilizado o código `mac0352-servidor-exemplo-ep1.c` disponibilizado. As modificações foram feitos dentro do loop de leitura dos packets enviados. Dentro dele, implementamos uma sequência de condicionais que, com base no primeiro byte do recvline, verifica qual foi o comando utilizado.

Para a identificação do pacote, fizemos a conversão de hexadecimal para o sistema decimal. Além disso, mudamos o tipo do recvline de `char` para `unsigned char`, para que eles sejam corretamente identificados.

Connect

Caso o primeiro byte do recvline seja 16, trata-se de um pacote CONNECT.

Primeiro, devemos identificar se o servidor é capaz de processar o pacote CONNECT recebido. Se sim, o código de retorno do CONNACK será nulo. Caso contrário, será não nulo. Para achar o código de retorno no EP, de forma simplificada, verificamos:

- Se o protocolo recebido é o MQTT. Se não, devolve-se o código 1;
- Se for o MQTT, devolve-se o código 0.

Em resposta ao CONNECT, o servidor deve mandar um pacote CONNACK, que possui o seguinte formato:

HEADER	BYTE	CONTEÚDO
FIXED	Byte 1	32
	Byte 2	2
VARIABLE	Byte 1	0 ou 1
	Byte 2	Connect Return Code

No EP, fazemos um write mandando essas informações.

Publish

Caso o primeiro byte do recvline esteja entre 48 e 63, trata-se de um pacote PUBLISH

Primeiro, devemos identificar, no pacote recebido, a mensagem enviada e o tópico em que se quer publicar. Essas informações são pegas do *variable header* e do *payload* do pacote PUBLISH.

Em seguida, precisamos mandar a mensagem enviada para todos os clientes inscritos no tópico. Para isso, nessa implementação, faremos um loop por todos os arquivos dos clientes inscritos no tópico, escrevendo a mensagem enviada.

IMPLEMENTAÇÃO:

Escolhemos guardar as informações de publicações e subscrições em arquivos `FIFO`. Cada subscrição cria um arquivo próprio com o tópico em que se inscreveu e fica lendo desse arquivo para verificar se algo foi publicado. A tarefa do publish, portanto, é identificar todos os arquivos dos clientes inscritos no tópico (esses arquivos terão o tópico no nome) e escrever a mensagem publicada neles.

Subscribe

Caso o primeiro byte do recvline seja 130, trata-se de um pacote SUBSCRIBE

Primeiro, devemos identificar, no pacote recebido, o tópico em que o cliente quer se inscrever. Essa informação é encontrada no *payload* do pacote SUBSCRIBE, que contém tanto o comprimento do tópico quanto o seu nome.

Consideramos somente um tópico por subscribe, conforme explicitado no fórum da disciplina.

Em resposta ao SUBSCRIBE, o servidor deve mandar um pacote SUBACK, que possui o seguinte formato:

HEADER	BYTE	CONTEÚDO
FIXED	Byte 1	144
	Byte 2	Remaining Length
VARIABLE	Byte 1	Packet Identifier MSB
	Byte 2	Packet Identifier LSB
PAYLOAD	Byte 1	Return Code

Subscribe

Para cada cliente inscrito em determinado tópico, vamos criar um arquivo para esse cliente nesse tópico. Quando algo for publicado nesse tópico, o publisher vai escrever nesse arquivo. Assim, o cliente, para receber atualizações, precisará ficar lendo esse arquivo. O nome do arquivo deve conter o nome do tópico, para que o pub possa identificá-lo para receber atualizações. Além disso, deve ser único para cada cliente. Para isso, usaremos o pid. Nesse EP, usamos um padrão semelhante ao usado no arquivo ep1+pipe do enunciado: temp.mac0352.ep1.TOPIC_NAME.PID

Precisamos agora que o cliente que se inscreveu fique fazendo duas coisas:

1. Fique lendo do seu arquivo para verificar se chegou algo do pub
2. Fique lendo do socket para verificar se desconectou ou identificar erro no ping

Para isso, fizemos uso de threads (linhas de execução), para que o cliente consiga realizar essas duas tarefas de forma "simultânea".

Pingreq e Disconnect

Caso o primeiro byte do recvline seja 192, trata-se de um pacote PINGREQ.

Em resposta ao PINGREQ, o servidor deve mandar um pacote PINGRES, que possui o seguinte formato: (não possui *variable header* nem *payload*)

HEADER	BYTE	CONTEÚDO
FIXED	Byte 1	208
	Byte 2	Remaining Length (0)

O PINGREQ possui como byte 2 de seu *fixed header* a *remaining length*, que é sempre igual a zero para esse pacote. No EP, usamos essa informação para identificar um possível erro (caso *remaining length* seja diferente de zero).

Caso o primeiro byte do recvline seja 224, trata-se de um pacote DISCONNECT.

Caso o servidor receba um pacote DISCONNECT, ele deve sair do loop de ficar lendo conexões.

Análise e testes de desempenho

Para fazermos a análise e testes de desempenho, rodamos os clientes numa máquina virtual e o servidor numa outra máquina conectada na mesma rede. Usou-se um script para rodar as subscrições e publicações, com um loop rodando os comandos mosquitto_sub e mosquitto_pub. O servidor foi rodado na porta padrão do protocolo MQTT, para que o Wireshark identifique os pacotes como MQTT.

Consumo de Rede

O consumo de rede foi medido por meio do Wireshark, filtrando-se pelos envios por TCP na porta 1883. Em `statistics`, podemos ver tantos os packets utilizados quanto os bytes.

Para cada um dos casos de teste, foram feitas 5 medições, para conseguirmos ter alguma validade estatística para a média e o desvio padrão.

Análise e testes de desempenho

Consumo de CPU

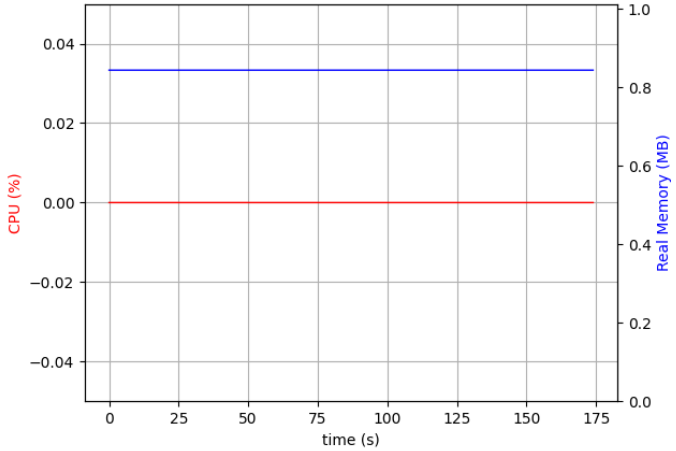
Para realizar os testes de consumo de CPU, usou-se a ferramenta psrecord. Ela mede tanto o consumo da CPU quanto de memória ao longo do tempo de execução de um determinado processo, com base no seu pid (foi pego o pid do servidor com o comando top). Além disso, podemos acionar a opção de incluir o consumo dos seus processos filhos. Na implementação do psrecord, os processos filhos são somados para apresentar o valor final. No entanto, devida à falta de precisão na medição, em grande parte das vezes os valores dos filhos gerados no EP são mostrados como zero, resultando numa soma geral zero.

O psrecord também possui a opção de gerar gráficos e logs das atividades. Usamos ambos, os gráficos para visualização ao longo do tempo e os logs para pegar os valores para calcular a média e desvio padrão.

Foram feitas 5 medições para validarmos os resultados, mas mostraremos, para efeitos de apresentação, apenas 1 caso de gráfico e log gerado para cada teste, já que eles se apresentam semelhantes entre si.

Testes só com o broker

Consumo de CPU



A porcentagem de uso da CPU foi zero. O uso de CPU foi mais baixo do que o medidor tem precisão para medir.

O uso de memória também permaneceu constante (não entrou nos condicionais do loop de leitura).

Consumo de Rede

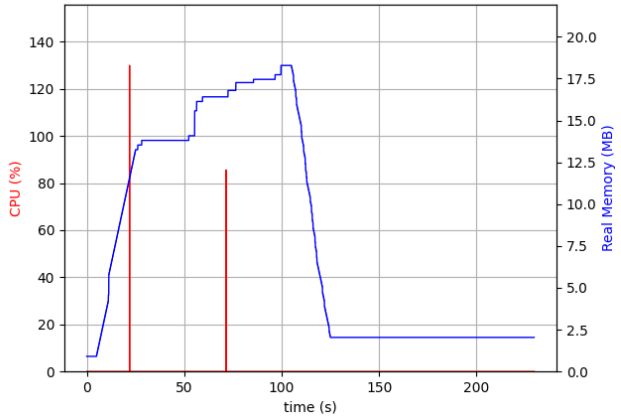
Pelo Wireshark, nenhum pacote foi identificado passando pela porta 1883, em nenhuma das medições.

Packets: 0 Média: 0

Bytes: 0 Desvio Padrão: 0

Testes com 100 pubs e 100 subs

Consumo de CPU



Percebe-se que, **na maior parte do tempo, o consumo de CPU foi igual a zero.** Ou seja, no hardware utilizado o consumo está bem baixo, e a ferramenta não consegue pegar com precisão cada processo filho na soma.

Porém, houve a presença de **2 picos**, cujos valores foram pegos do log gerado:129.900%, 85.500%. Provavelmente, nesses picos, foi possível medir um valor diferente de zero dos processo filhos e, ao somar, deu um valor bem maior do que antes.

Média: 0.94% de uso da CPU

Desvio Padrão: 10.26

Já o uso de memória foi aumentando na medida que foram alocados arquivos e vetores, voltando ao valor inicial no fim do processo.

Testes com 100 pubs e 100 subs

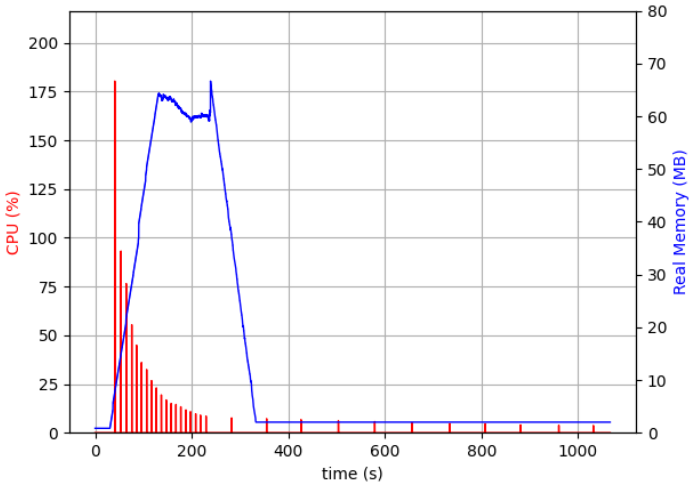
Consumo de rede

Amostra	1	2	3	4	5	Média	Desvio Padrão
Packets	21585	22788	23576	21505	21267	22144.2	994.4
Bytes	1621701	1976267	2109283	1621789	1640607	1793929.4	232108.59

A quantidade de packets trocados, nas 5 amostras, girou em torno da média, com valor mínimo de 21267 e máximo de 23576. O número de bytes também manteve-se dentro de um intervalo adequado, apesar de variar um pouco mais (com um mínimo de 1621701 e máximo de 2109283).

Testes com 1000 pubs e 1000 subs

Consumo de CPU



Percebe-se que, **na maior parte do tempo, o consumo de CPU foi igual a zero.** Ou seja, no hardware utilizado o consumo está bem baixo, e a ferramenta não consegue pegar com precisão cada processo filho na soma.

Mas houve, de forma a semelhante ao teste com 100 pubs e 100 subs, a **presença de alguns picos**, cujos valores foram pegos do log gerado.

Média: 0.83% de uso da CPU

Desvio Padrão: 7.61

Já o uso de memória foi aumentando na medida que foram alocados arquivos e vetores, caindo de volta no final do processo.

Testes com 1000 pubs e 1000 subs

Consumo de rede

Amostra	1	2	3	4	5	Média	Desvio Padrão
Packets	223945	224403	231902	227079	222843	226034.4	3630.84
Bytes	17139728	17410519	20910748	19050920	17189375	18340258	1638801.23

A quantidade de packets trocados, nas 5 amostras, girou em torno da média, com valor mínimo de 222843 e máximo de 231902. O desvio padrão relativo foi menor do que no teste com 100 pubs e 100 subs, o que era esperado do ponto de vista estatístico. O número de bytes também manteve-se dentro de um intervalo adequado.

Observações

Na realização dos testes, qualquer outro software que utilizasse a Internet estava fechado, para evitar interferências.

Também foram feitos testes com tópicos diferentes simultâneos. Observou-se o consumo da rede e constatou-se que, para o mesmo número de pubs e subs, o número de bytes e packets transferidos é menor do que com um tópico, o que era esperado, já que cada pub só precisará mandar a mensagem para os clientes inscritos naquele tópico.

Informações sobre como executar o programa podem ser achados no arquivo LEIAME.md presentes no projeto.

Referências

Documentação do MQTT:
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718063
Ferramenta psrecord:
https://github.com/astrofrog/psrecord
Implementação da biblioteca pthreads.h:
https://stackoverflow.com/questions/10818658/parallel-execution-in-c-program
https://man7.org/linux/man-pages/man7/pthreads.7.html
https://www.geeksforgeeks.org/multithreading-c-2/
Uso da função open():
https://linux.die.net/man/3/open
Implementação da biblioteca pthreads.h:
https://www.decodeschool.com/C-Programming/File-Operations/C-Program-to-read-all-the-files-located-in-the-specific-directory