Comparação de Desempenho entre Servidores MQTT

Bruno Carneiro da Cunha
<brunocarneirodacunha@usp.br>

4 de maio de 2020

1 Introdução

Nos últimos anos, aplicativos baseados em MQTT aumentaram bastante fazendo com que esse seja o protocolo pub/sub mais popular do mundo, com utilização em uma ampla variedade de domínios como saúde, automação residencial, transportes inteligentes e distribuição de energia. Apesar da popularidade, faltam soluções que garantam uma boa segurança do MQTT. De fato, o protocolo não foi desenvolvido com segurança em mente.

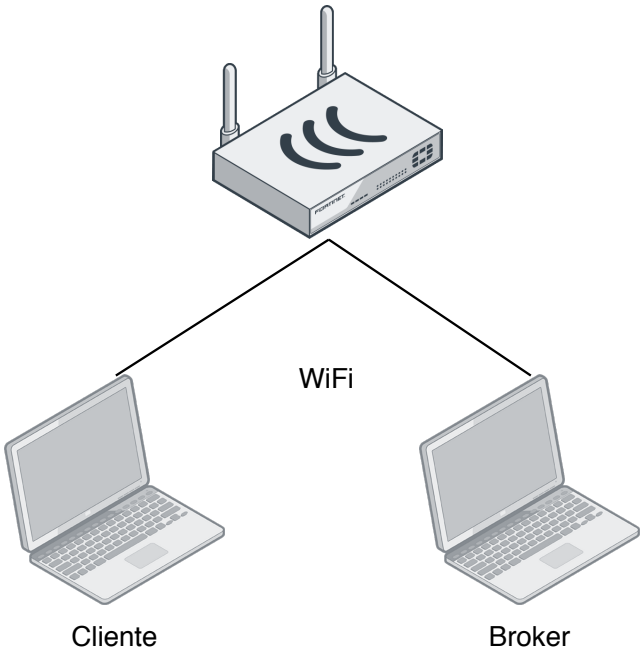
A fim de realizar experimentos para avaliar a segurança do MQTT, o mesmo deverá ser implantado em duas redes de experimentação que serão construídas na USP e na UNIFESP. O protocolo será integrado em uma plataforma para cidades inteligentes chamada InterSCity, que foi desenvolvida dentro do escopo do projeto de mesmo nome.

Este relatório apresenta uma comparação de performance entre três implementações de servidores, ou *brokers*, MQTT. Através dos testes e dos seus resultados, pretende-se justificar a escolha de uma implementação para a integração à plataforma.

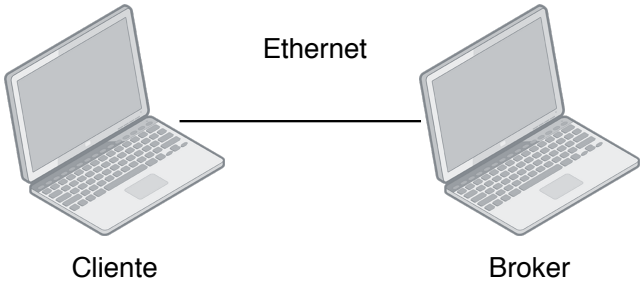
2 Ambiente de Experimentação

Cada teste foi executado em três ambientes de rede diferentes. No primeiro, os clientes MQTT e o servidor se comunicaram através da interface virtual de loopback de uma máquina com dois cores *Intel i5* de 1,6 GHz, com 4GB de memória RAM DDR3.

No segundo, os clientes executaram na mesma máquina, porém comunicaram-se através de uma rede Wi-Fi 2,4 GHz residencial com um *broker* que se encontrava em um computador de quatro cores *Intel i7* de 2,2 GHz, com 4GB de memória RAM DDR3. Ambos os computadores possuem interfaces de rede Wi-Fi do fabricante Broadcom, modelo BCM43xx 1.0. Também havia tráfego de outros computadores nessa rede.



No terceiro ambiente, conectou-se as duas máquinas mencionadas acima através de uma rede Ethernet direta, sem intermediário, com cabo *Cat 5*. A largura de banda do enlace medida pelo *iperf* foi de 94,5 Mbits/s. Os clientes e broker rodaram em seus respectivos *hosts*.



O computador de dois cores executou os seus programas no sistema operacional macOS 10.14.4, e o computador de quatro cores executou no sistema macOS 10.13.6. Os clientes de teste rodaram o script mqttperf, escrito em Ruby, usando a implementação de cliente MQTT do projeto Eclipse Paho.

3 Implementações de Brokers Testadas

Dadas as necessidades do projeto somente era interessante testar as implementações de código aberto, e que possuíam capacidade de servir clientes através de conexões seguras TLS/SSL. Por isso, de uma lista extensa de implementações, somente as seguintes foram consideradas:

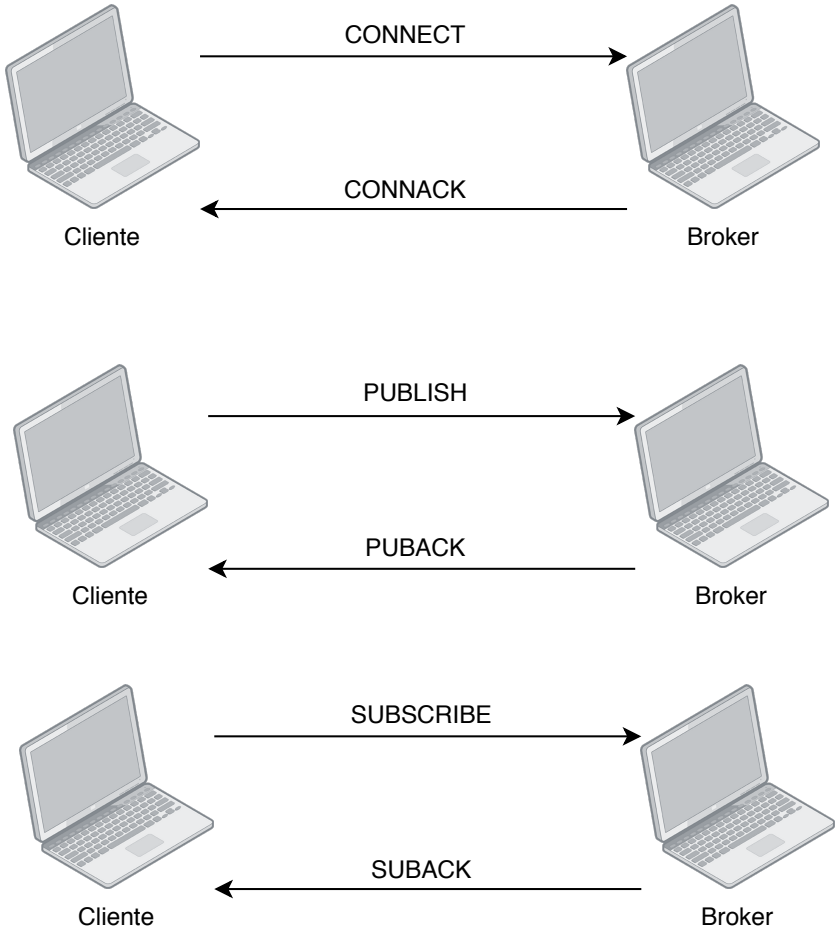
Nome	Desenvolvedor	Licença	Linguagem
EMQ	EMQ	Apache License version 2.0	Erlang
HiveMQ CE	dc-square GmbH	Apache License version 2.0	Java
VerneMQ	VerneMQ/Erlio	Apache License version 2.0	Erlang/OTP
Mosquitto	Eclipse	Eclipse Distribution License 1.0 (BSD)	C
Aedes	-	MIT License	JavaScript (nodejs)

Das escolhas acima, não foi possível instalar o broker EMQ, pois o link de download do binário no site oficial estava quebrado. Também não foi possível testar o servidor VerneMQ, pois o broker reiniciava (RESET) a conexão TCP assim que qualquer cliente tentava conectar-se à sua porta.

Logo, as implementações testadas foram Mosquitto v1.6.9, Aedes v0.41.0 e HiveMQ v1.10. As versões foram escolhidas por serem as mais recentes de cada implementação em Março de 2020.

4 Experimentos Planejados

O protocolo MQTT prevê as seguintes mensagens:

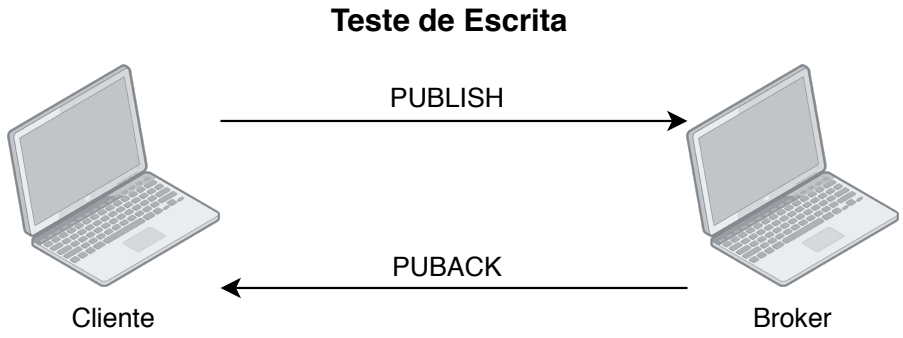


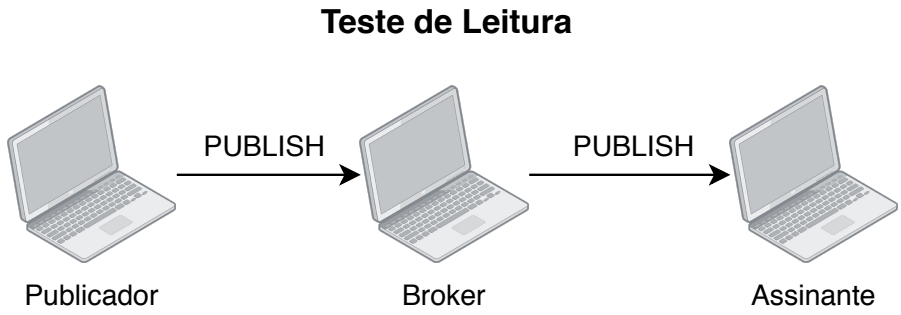
Na mensagem CONNECT, o cliente tenta se conectar ao broker, informando a versão do protocolo, um identificador e configurando algumas flags, enquanto o servidor responde com CONNACK, aceitando ou não a conexão.

Na mensagem PUBLISH o cliente publica uma mensagem em algum tópico, o servidor responde com PUBACK. A mensagem do tipo PUBLISH também é transmitida pelo broker aos assinantes de um determinado tópico, quando houver uma mensagem nova.

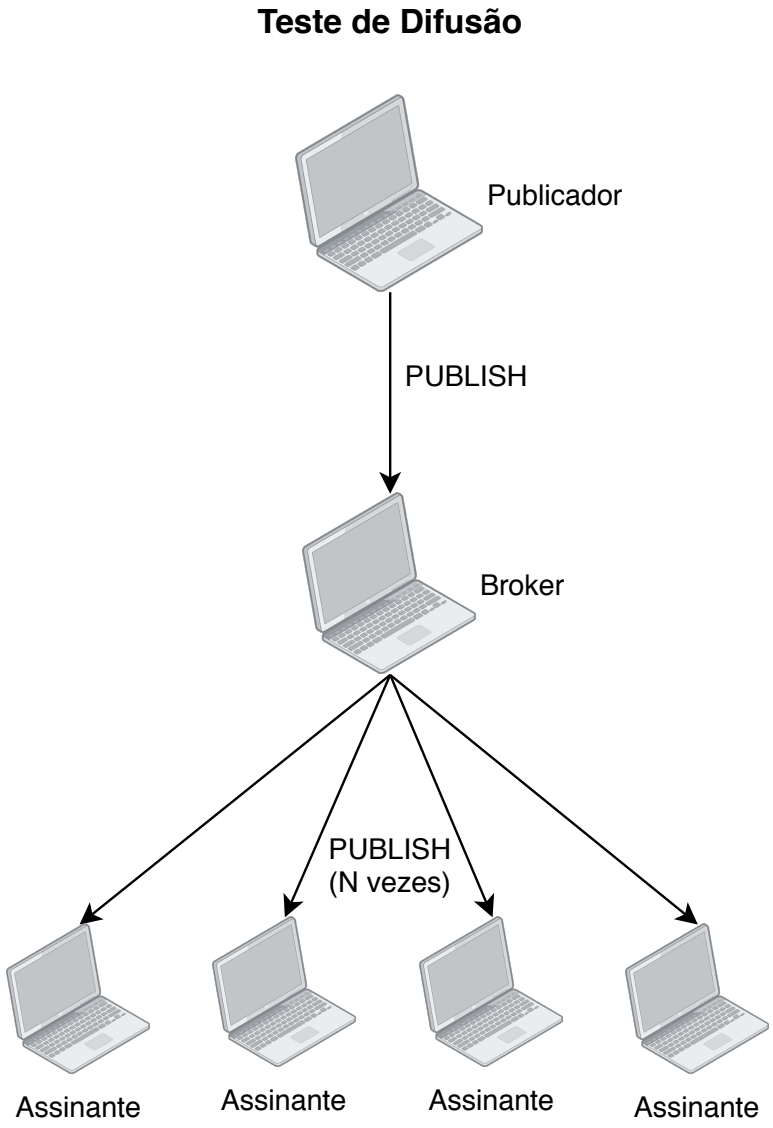
Na mensagem SUBSCRIBE, um cliente requisita assinar um tópico, ou seja, informa ao servidor que deseja receber todas as novas mensagens publicadas naquele tópico. O servidor responde com SUBACK se a requisição for aceita.

Planejou-se testar o desempenho dos brokers através do teste de escrita, de leitura e de difusão das mensagens, de acordo com os diagramas a seguir:





No teste de escrita, conta-se o intervalo de tempo entre o envio da mensagem PUBLISH, e o recebimento de PUBACK. Repetiu-se o teste 1000 vezes, calculando-se a média e o desvio padrão das amostras. O teste de leitura consiste do mesmo procedimento, mas conta-se o intervalo de tempo entre o envio da mensagem PUBLISH, e o recebimento da mesma pelo único cliente assinante do tópico. Ambos os conjuntos de testes foram feitos com mensagens de comprimento de 1 byte, e com mensagens de comprimento de 1000 bytes.



O teste de difusão consiste em contar o intervalo de tempo entre o envio de PUBLISH pelo cliente publicador, e o recebimento da mensagem por cada assinante daquele tópico. Calcula-se a média entre os intervalos contados por cada assinante. Esse teste foi repetido 20 vezes, calculando-se o desvio padrão entre as médias resultantes. O teste foi executado com mensagens de comprimento 1 byte, em tópicos com N assinantes, sendo $N = [10, 50, 100, 200]$.

Todos os testes de escrita, leitura e difusão foram rodados em duas modalidades: conexão insegura *plaintext*, e conexão criptografada através de TLS/SSL.

Novamente, o script usado para rodar os testes encontra-se no repositório mqttperf, e os dados brutos dos testes podem ser encontrados aqui.

5 Resultados Encontrados

5.1 Fluxo de Dados

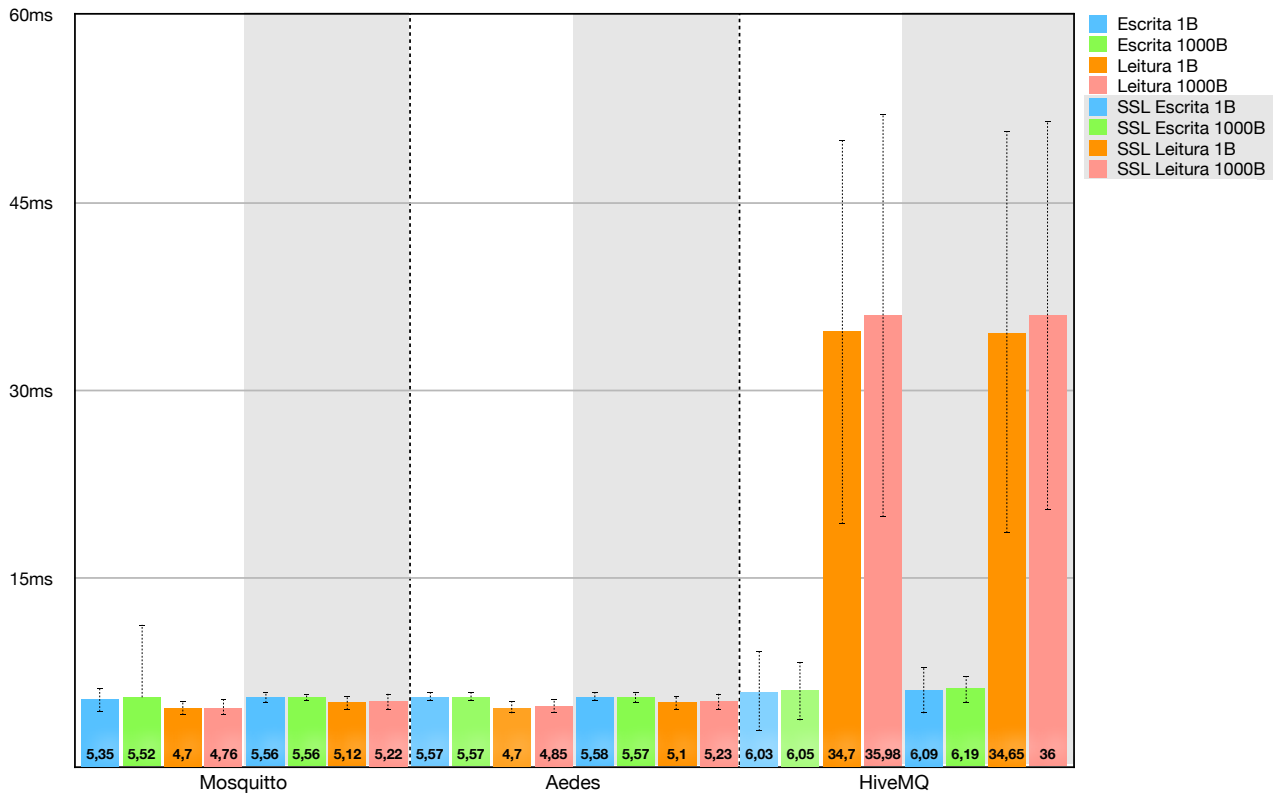
Os dados brutos gerados pelo script *mqttperf* foram individualmente guardados em arquivos de texto. Os resultados dos testes de escrita e leitura foram manualmente digitados na tabela de dados que gerou os gráficos abaixo. Os resultados dos testes de difusão foram extraídos através de scripts em *Python* e processados para achar os valores de média e desvio padrão. Esses valores então foram manualmente digitados. Todos os gráficos foram gerados através de tabelas criadas no programa *Numbers*. Todos os scripts estão disponíveis no repositório.

5.2 Observações sobre os gráficos

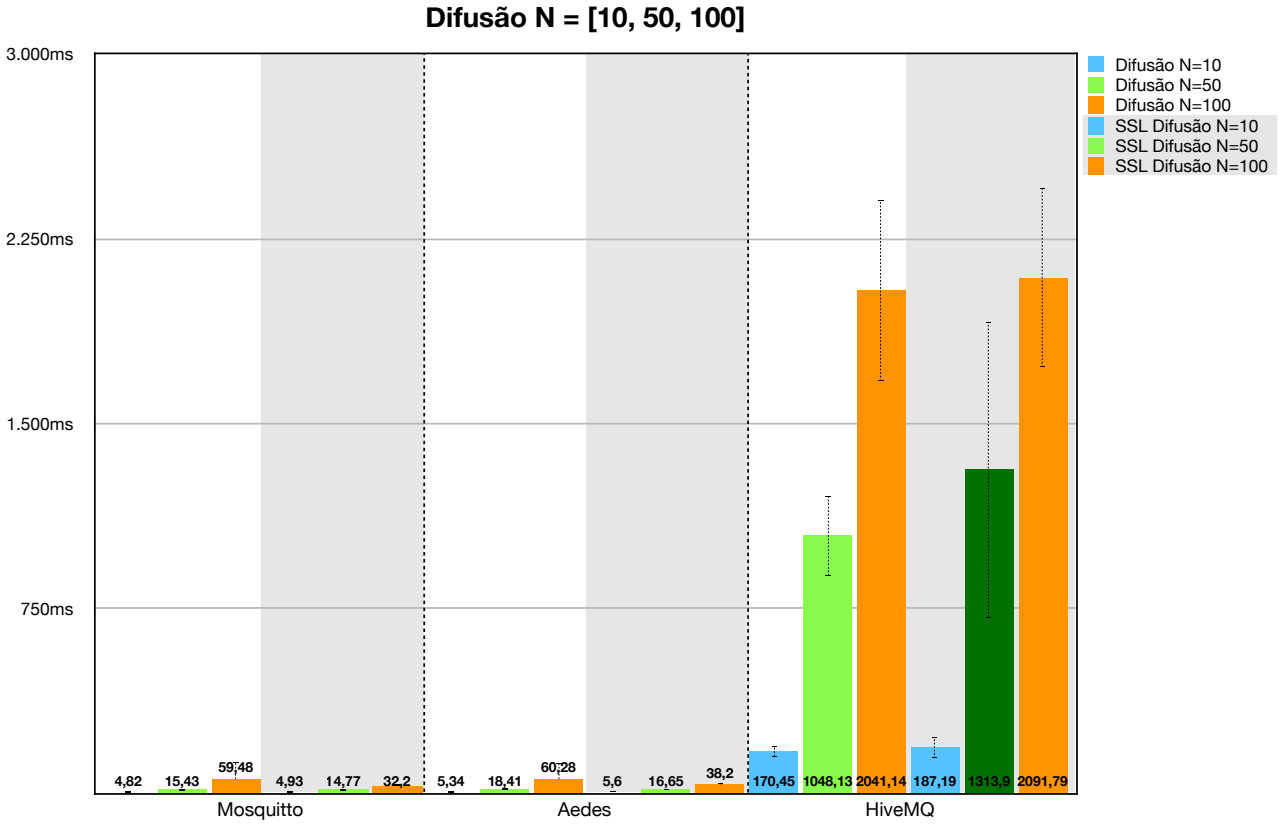
Os gráficos a seguir mostram o tempo de execução de cada teste como equivalente à altura de cada barra, em milisegundos. As linhas tracejadas acima e abaixo de cada barra representam o desvio padrão encontrado no teste. As barras que não possuem linhas tracejadas para baixo possuíam desvio padrão que alcançava intervalos de tempos negativos, por isso foram omitidas.

5.3 Localhost

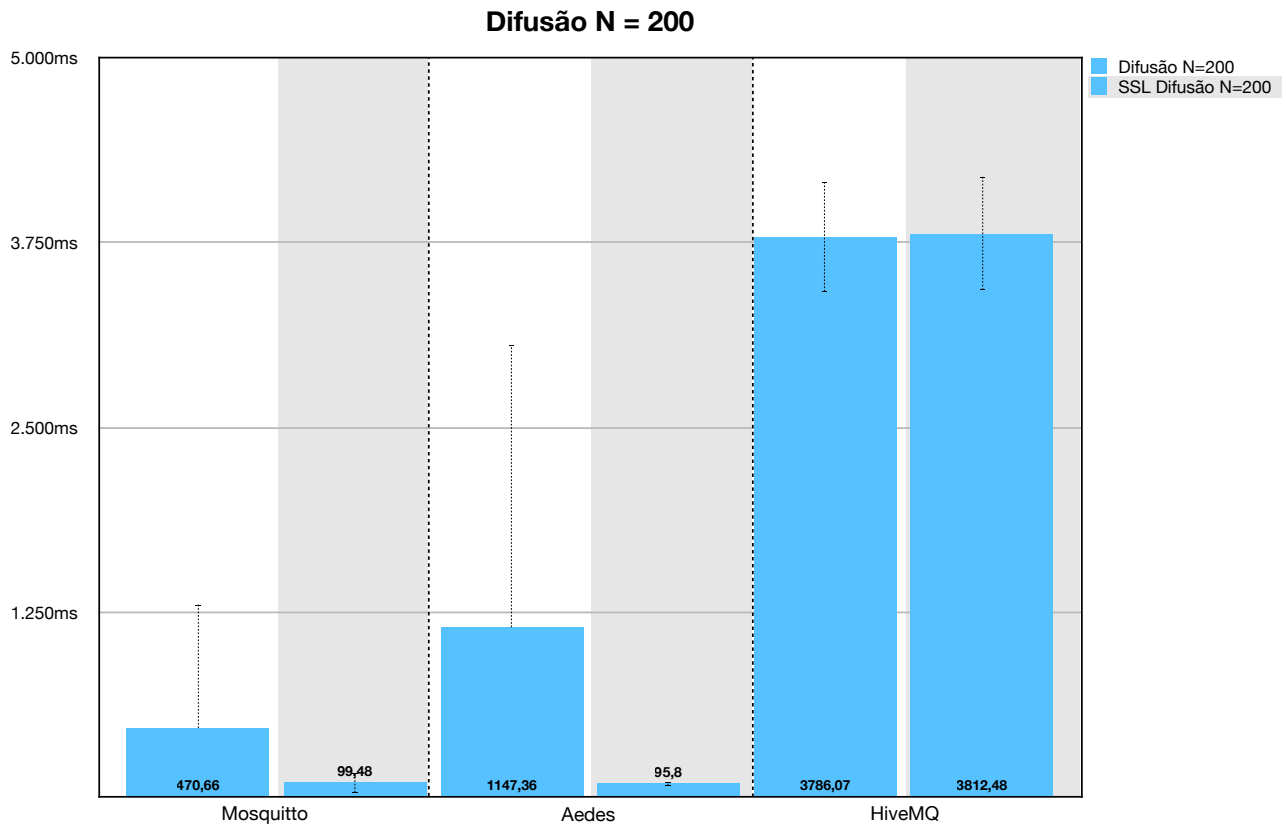
5.3.1 Escrita/Leitura



5.3.2 Difusão N = [10, 50, 100]

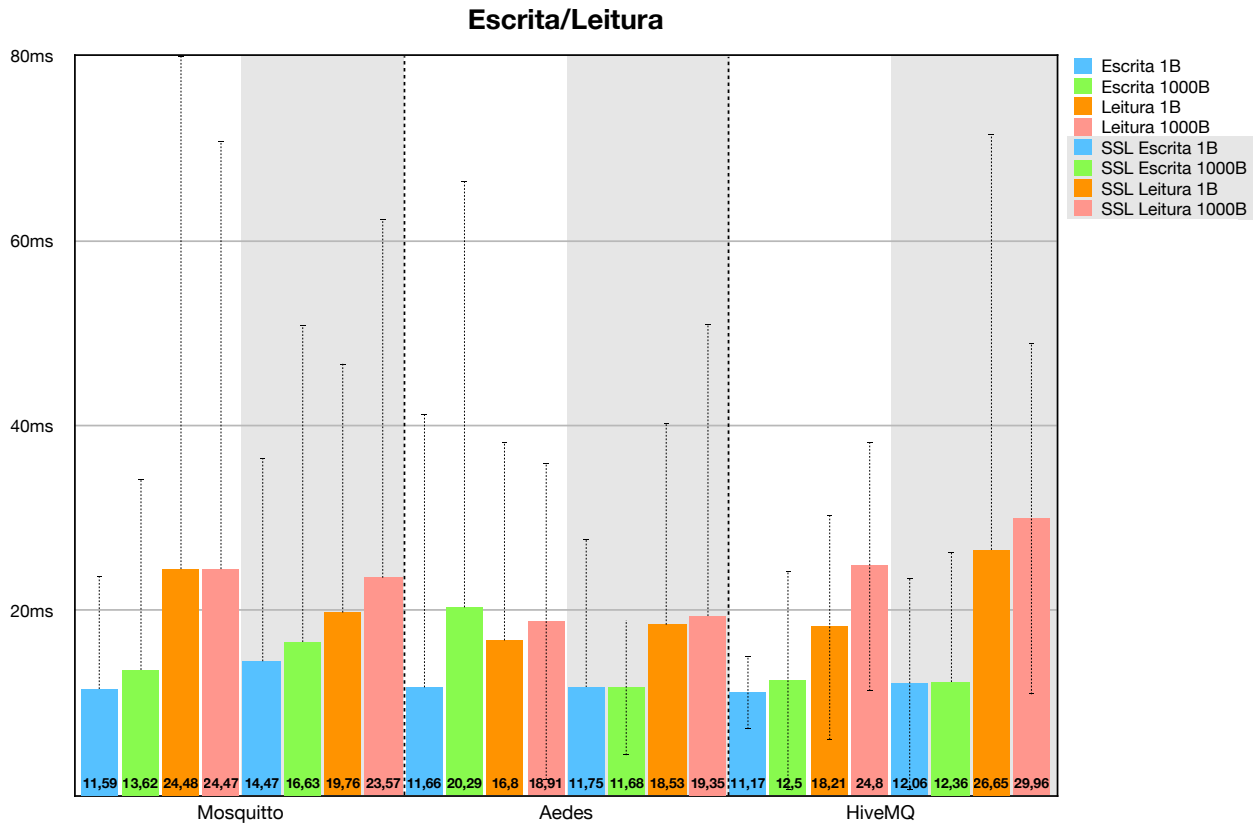


5.3.3 Difusão N = 200

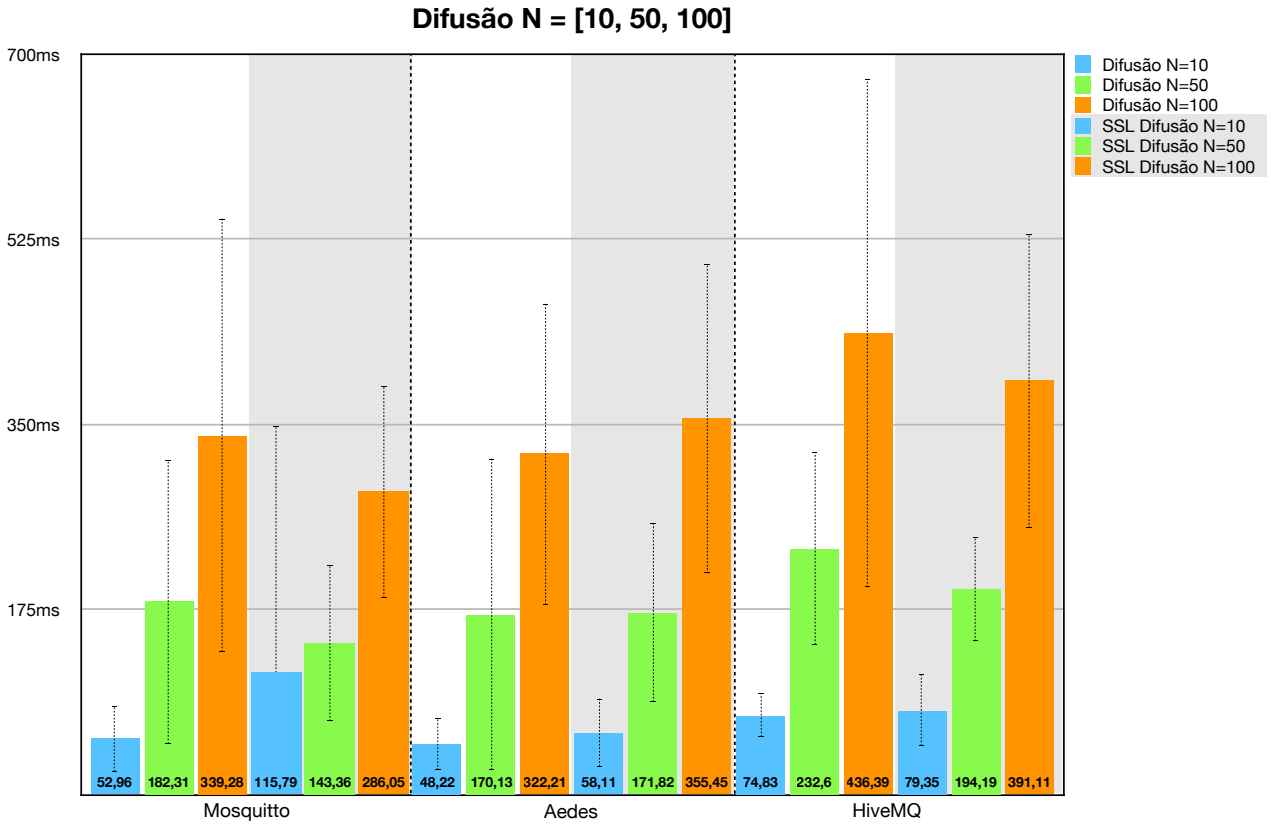


5.4 Wi-Fi

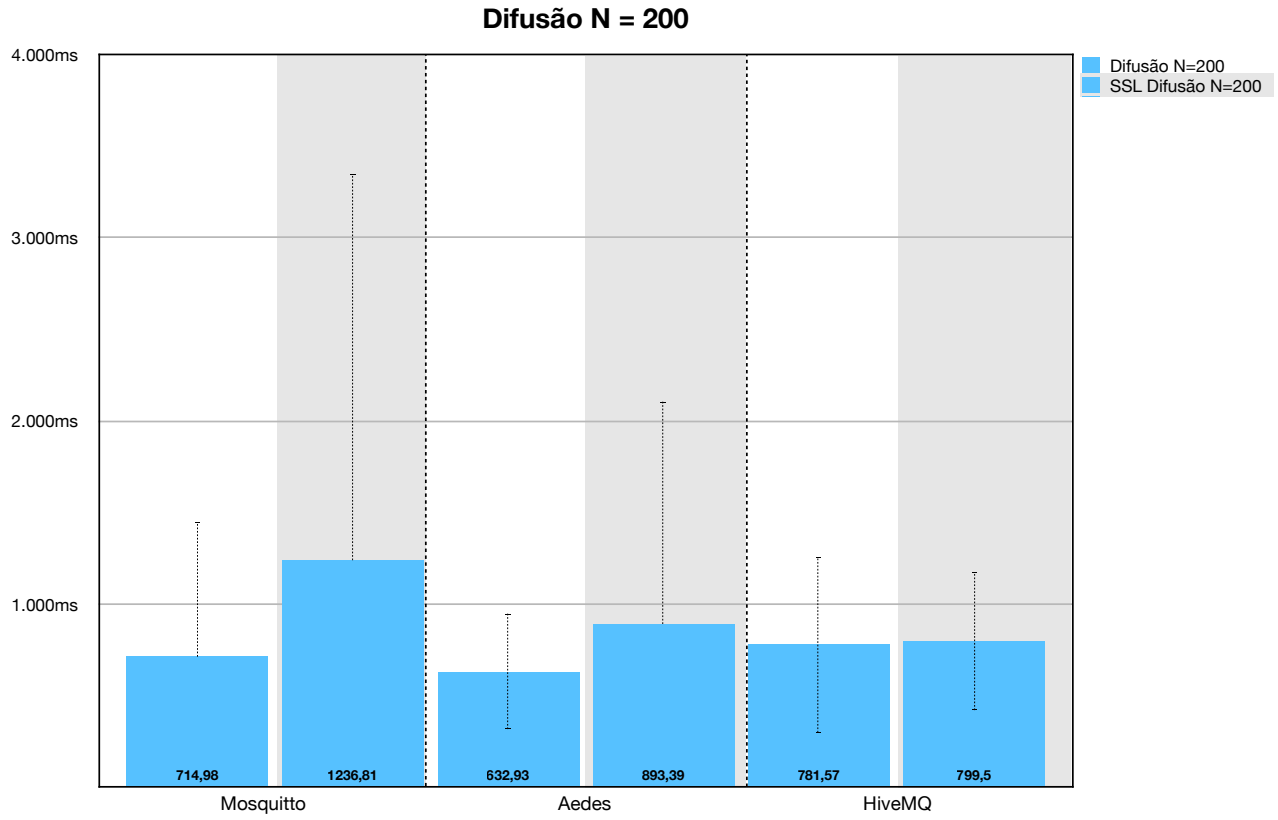
5.4.1 Escrita/Leitura



5.4.2 Difusão N = [10, 50, 100]

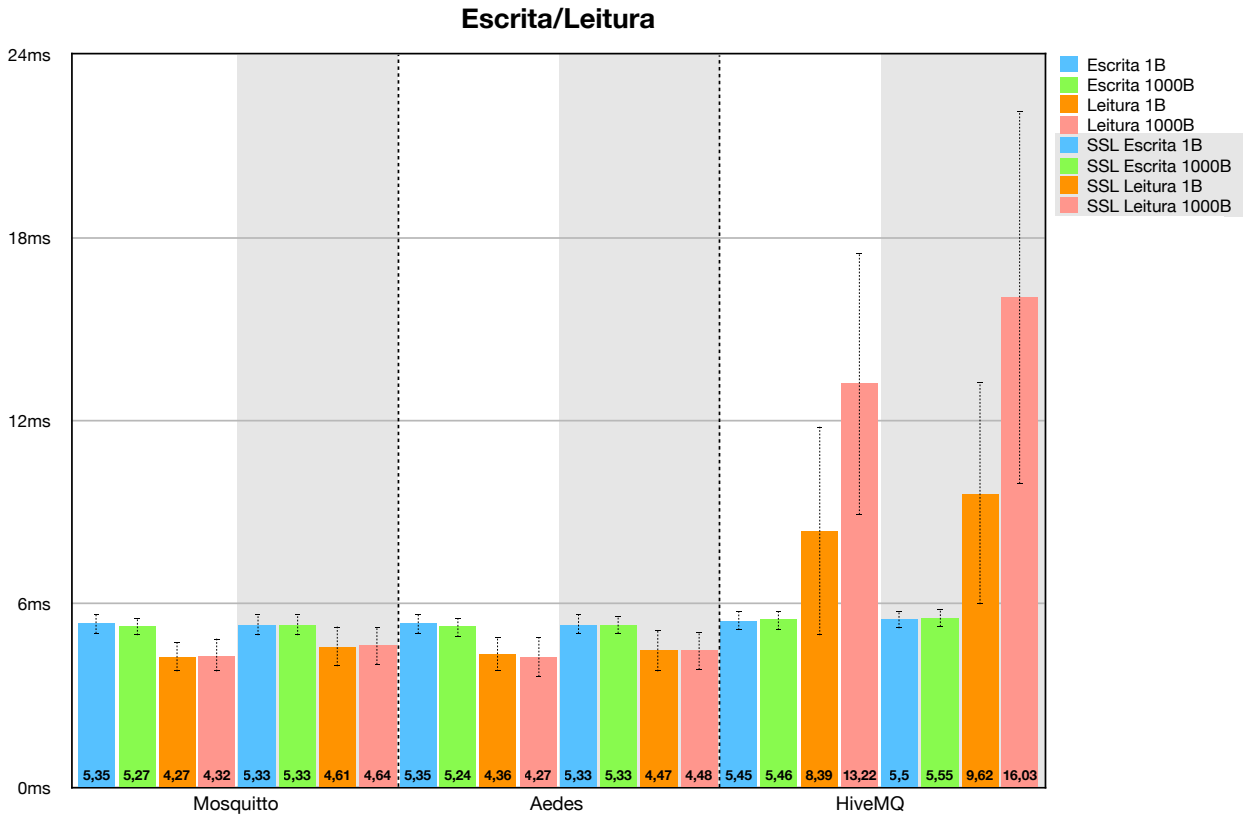


5.4.3 Difusão N = 200

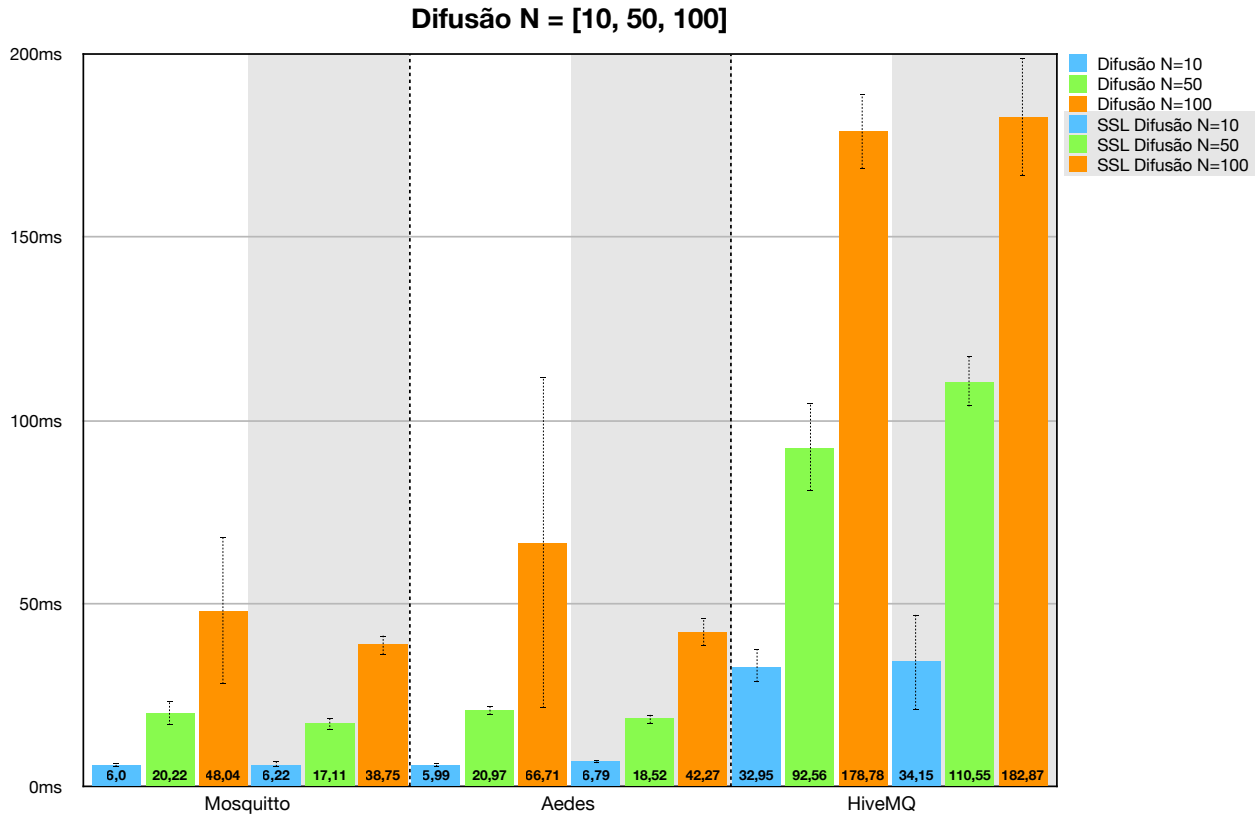


5.5 Ethernet

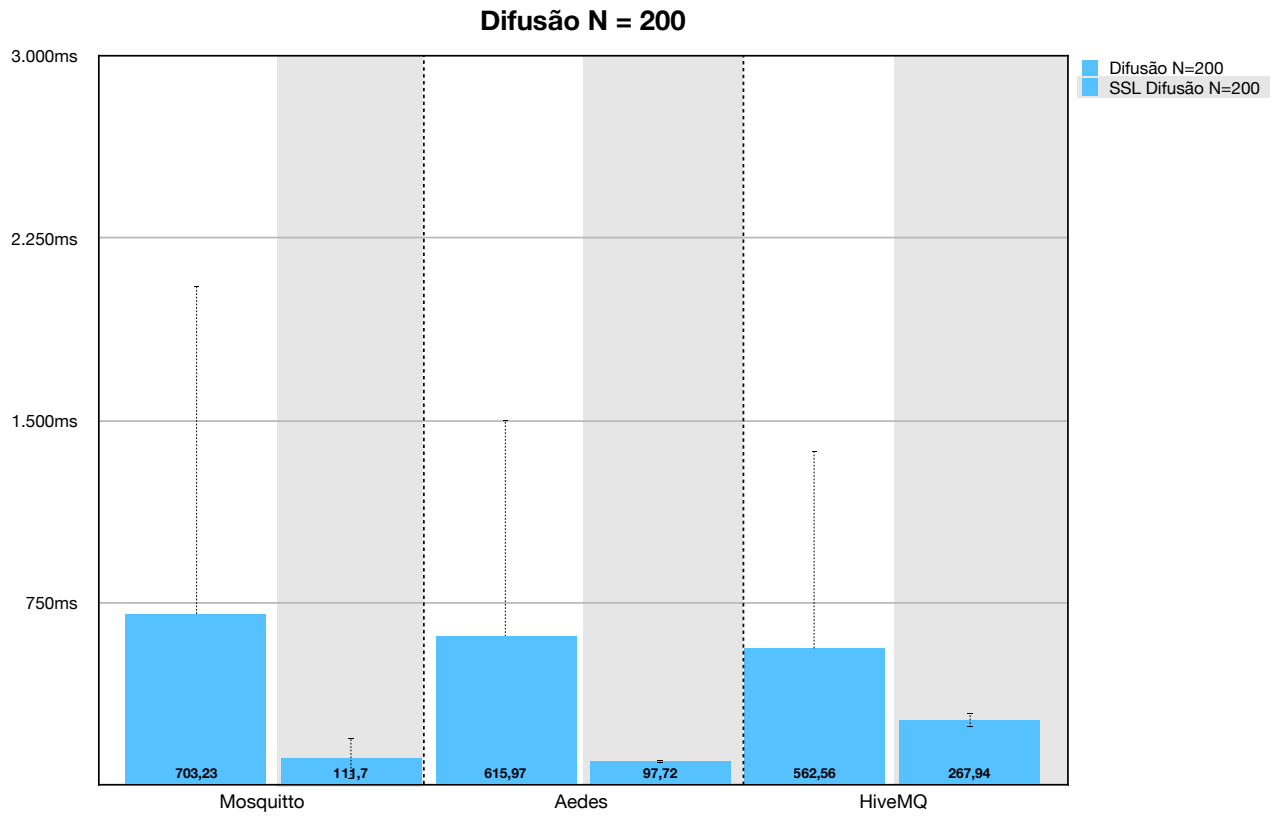
5.5.1 Escrita/Leitura



5.5.2 Difusão N = [10, 50, 100]



5.5.3 Difusão N = 200



6 Conclusões

As implementações *Mosquitto* e *Aedes* apresentaram desempenho muito semelhante em todos os testes. Além disso, nos testes de difusão, as duas apresentaram crescimento do tempo de resposta compatível com o crescimento do número de nós na rede.

A implementação *HiveMQ* apresentou desempenho significativamente mais lento do que as outras, por mais que tenha apresentado tempo de resposta até mais rápido em dois testes de difusão com $N = 200$. Também o desempenho extremamente lento na interface virtual de *loopback* indica que pode haver algum *bug* provocando inconsistência na resposta.

Recomendo a integração do servidor *Mosquitto* à plataforma, devido ao bom desempenho e à estabilidade demonstrada.