

Alexandre Michetti Manduca
Filipe Del Nero Grillo

nUsp: 4890832
nUsp: 5378140

***Relatório do projeto final da disciplina de
Hipermissão***

São Carlos - SP, Brasil

12 de maio de 2011

Sumário

1	Introdução	p.2
1.1	AJAX	p.3
1.2	Websockets	p.3
1.3	Objetivos do trabalho	p.4
2	Preparação do experimento	p.6
2.1	Tecnologias utilizadas	p.6
2.2	Ambiente	p.7
2.3	Dados a serem coletados	p.8
3	Execução	p.10
4	Dados obtidos	p.11
4.1	Processo servidor	p.11
4.2	Processo cliente	p.13
5	Resultados obtidos	p.16

1 Introdução

Com a evolução da internet, as tecnologias que a compõem também estão em constante desenvolvimento. Em meados de 2004 surgia o termo *Web 2.0*, que foi cunhado para se referir a uma nova geração de serviços que surgiam utilizando a *Web* como plataforma. Uma das características marcantes dessa nova geração de serviços era o fato de se buscar que as aplicações *Web* possuísem uma interação com o usuário mais próxima da experiência de se utilizar um aplicativo *desktop*.

Uma das técnicas utilizadas para melhorar a interação com usuário é conhecida como AJAX (*Asynchronous JavaScript and XML*), que será explicado adiante na seção 1.1.

Seguindo esta evolução, no início de 2008 a W3C (*World Wide Web Consortium*)¹ anunciou a primeira especificação do HTML5, uma nova versão da linguagem de marcação que é utilizada para formatar e organizar o conteúdo que está disponível na internet. O HTML5 está previsto para ser finalizado no ano de 2012 e traz alterações muito interessantes no sentido de aproximar ainda mais a internet do comportamento de aplicações *desktop*. Dentre as novidades do HTML5 podemos destacar:

- Novas API's para uso de gráficos;
- Melhora no uso de multimídia sem necessidade de *softwares* de terceiros;
- Aprimoramento de uso *off-line*;
- Melhoria na depuração de erros;
- API para uso de canais de comunicação bidirecional (Websocket).

A *Web* vem sendo escolhida como plataforma de desenvolvimento e distribuição de uma gama cada vez maior de aplicações. Sistemas altamente interativos, que enviam

¹W3C é um consórcio de empresas de tecnologia que tem por objetivo desenvolver padrões para criação e interpretação de conteúdo para a *Web*

dados para os usuários em tempo real ou mesmo que permitem que diversos usuários interajam em tempo real são cada vez mais comuns, como por exemplo o Twitter² que mostra as atualizações dos seus amigos quase que instantaneamente, Foursquare³ que mostra os *check-ins* dos seus amigos quase na hora em que acontecem, entre outros.

O WebSocket foi desenvolvido para que se melhore ainda mais a comunicação entre o cliente e o servidor para aplicações de tempo real. O WebSocket será explicado na seção 1.2

1.1 AJAX

AJAX, sigla para *Assynchronous JavaScript and XML*, é na verdade um conjunto de tecnologias já existentes mas utilizadas de uma forma nova, tornando possível uma aplicação desenvolvida em JavaScript que esteja rodando no cliente se comunicar com o servidor de forma assíncrona. Dessa forma, o usuário pode continuar trabalhando e interagindo com a página, enquanto dados são enviados e requisitados do servidor.

Uma aplicação AJAX elimina a natureza interrompida de interação na *web* introduzindo uma camada intermediária, a aplicação AJAX, entre o usuário e o servidor. Essa camada torna a velocidade de resposta da aplicação mais rápida. Ao invés de carregar uma página web, no início da sessão o navegador carrega a aplicação codificada em JavaScript e ela fica responsável por renderizar a interface e fazer a comunicação com o servidor de acordo com as necessidades do usuário.

Como a aplicação AJAX está sempre ativa no *browser* do cliente, a interação é contínua, os dados serão carregados a medida que forem necessários sem impedir o usuário de continuar interagindo com a aplicação.

1.2 Websockets

As tecnologias utilizadas na *Web* atualmente, mesmo o AJAX, são construídas sobre o paradigma *request / response* imposto pelo HTTP. Nesse paradigma, os clientes iniciam as interações com o sistema, o que não é adequado para a obtenção de informações em tempo real, já que o servidor recebe primeiro as informações.

²<http://www.twitter.com>

³<https://foursquare.com/>

Para tentar contornar esses problemas, uma série de mecanismos construídos sobre a tecnologia AJAX e HTTP foram desenvolvidos, de modo a permitir a obtenção de dados em tempo real pelos clientes, ou pelo menos passar essa sensação para o usuário. Apenas para exemplificar, um mecanismo com esse fim conhecido como *Long Polling*, funciona mantendo conexões do cliente abertas até que um dado esteja disponível no servidor. Quando esse dado fica disponível e é enviado para o cliente, um novo *request* é feito, e sua conexão não é fechada até que um novo dados esteja disponível. O ciclo assim se repete passando a ilusão de que o servidor esta enviando em tempo real os dados para o cliente.

Contudo, todos esses mecanismos tem um problema, que é o *overhead* causado pelo HTTP. Por exemplo, cada *request/response* HTTP carrega uma série de informações de cabeçalho que podem tornar a latência dessas aplicações muito grande, tornando algumas aplicações inviáveis. Pense como exemplo, um jogo *multiplayer* de tiro que rode em um *browser*.

Visando esse tipo de aplicação, a W3C está padronizando uma nova API, que visa trazer conexões bidirecionais e *full-duplex* para os navegadores e servidores *Web* através de Sockets TCP. O *handshake* do WebSocket é feito via HTTP, que permite em sua versão 1.1 especificar cabeçalhos com essa finalidade. No entanto, a especificação WebSocket ainda tem algumas problemas até poder ser adotada de fato na *Web*, como o fato de que muitos navegadores, servidores de aplicação, servidores *proxy* e *firewalls* não estão preparados para ele.

1.3 Objetivos do trabalho

O objetivo deste trabalho é realizar uma análise comparativa entre as tecnologias AJAX e Websocket em uma aplicação que utiliza comunicação intensamente. Para tal serão desenvolvidas duas aplicações de bate-papo simples e construídas especificamente para o propósito desta análise.

Ambas as aplicações foram desenvolvidas com o objetivo de se manter o mesmo fluxo de interação e funcionarem de maneira exatamente igual aos olhos de um usuário leigo. Nas Figuras 1 e 2, podemos ver as *interfaces* das aplicações implementadas com Websocket a AJAX, respectivamente. Com exceção da palavra "ajax" no título da página, nenhuma outra distinção é visível para o usuário.

Holy Chat!

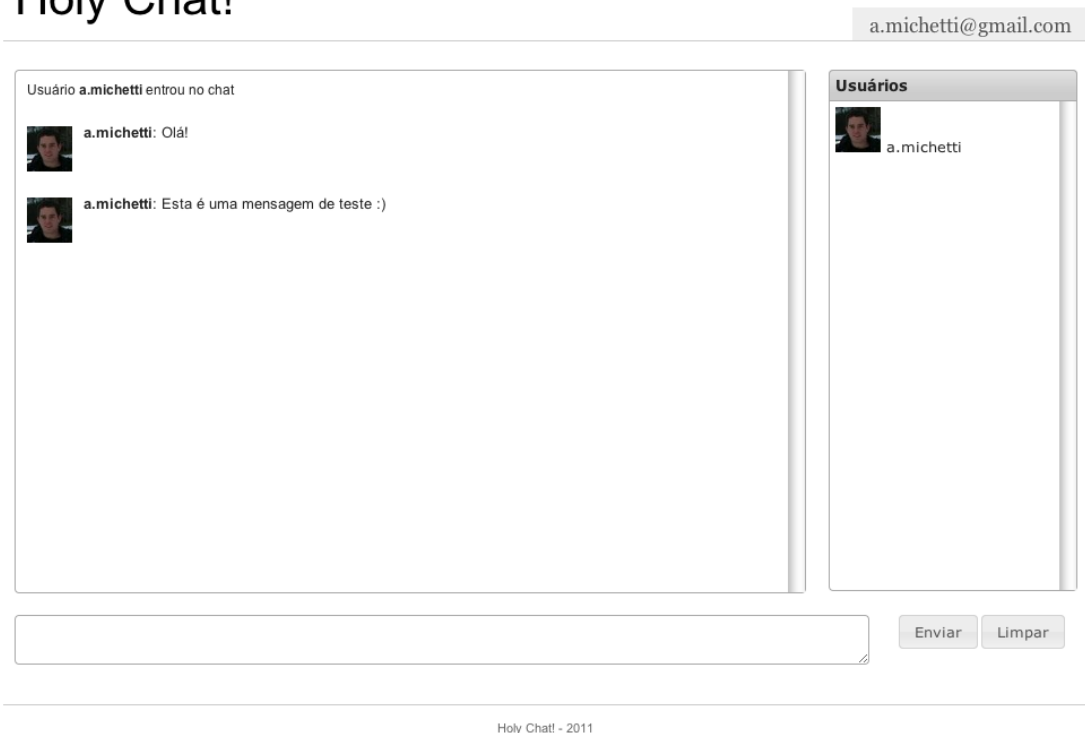


Figura 1: Interface do bate-papo desenvolvido com a tecnologia Websocket.

Holy ajax Chat!

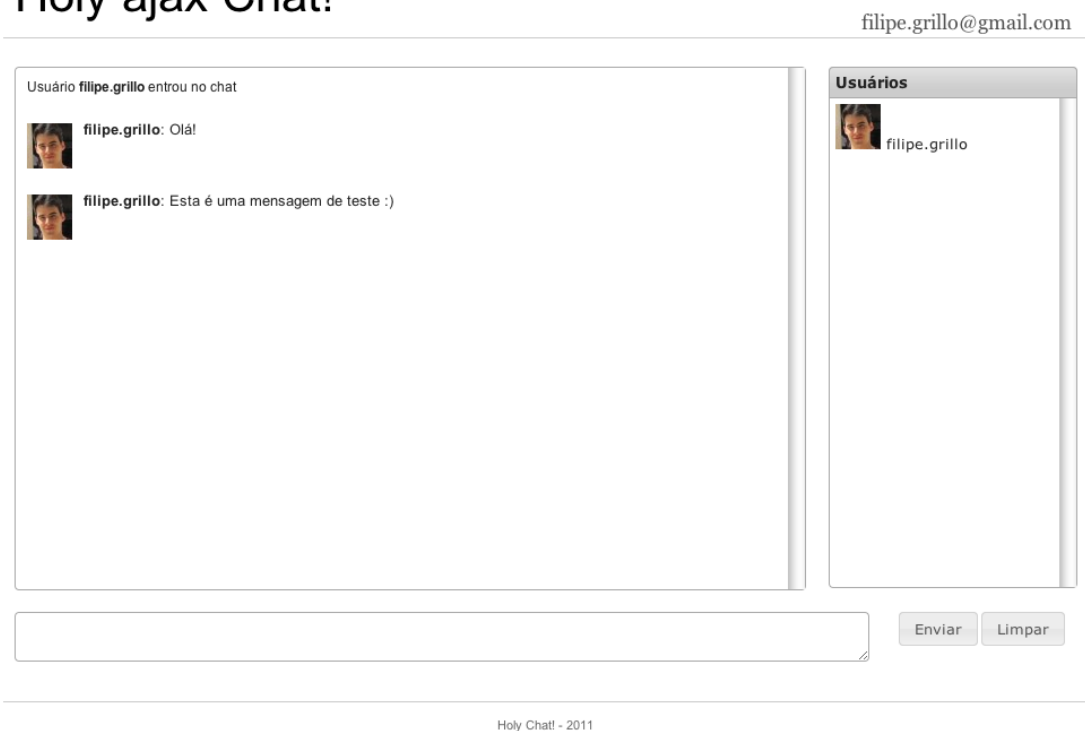


Figura 2: Interface do bate-papo desenvolvido com a tecnologia AJAX.

2 *Preparação do experimento*

Para a execução do experimento, foram construídas duas aplicações de chat, uma baseada em AJAX e outra em WebSockets, e também dois *scripts* para simularem múltiplos clientes enviando e recebendo mensagens ao mesmo tempo no servidor.

Tirando a parte visual da aplicação, as implementações do lado do servidor foram feitas de maneira a seguirem uma implementação usual de AJAX e WebSockets. Ou seja, não houve preocupação em manter semelhanças nas implementações, e sim em manter a mesma funcionalidade, com o modelo usual de desenvolvimento em cada tecnologia.

Na aplicação WebSockets, cada mensagem é repassada para todos os outros usuários no instante em que chega ao servidor. Enquanto que na implementação com AJAX, cada mensagem que chega é armazenada em um *array* em memória, e a cada 0,5 segundos cada cliente faz uma requisição para obter a lista de mensagens do servidor, passando como parâmetro a última mensagem que recebeu.

Dois servidores Ubuntu foram então preparados em uma solução de *Cloud Computing*, sendo que um executaria o servidor do *chat* e o outro o script para simular os clientes, de modo que um não interferisse nas medições do outro. Tanto o servidor quanto o *script* imprimiam linhas de log, que foram coletados para a geração das comparações e gráficos utilizados neste relatório.

2.1 Tecnologias utilizadas

Este trabalho se baseia na comparação entre as tecnologias AJAX e WebSockets. Como WebSocket é uma tecnologia ainda em desenvolvimento, existem poucas linguagens, navegadores e servidores de aplicação que a suportam. Com essas restrições, decidimos por utilizar o framework Node.js¹, que suporta nativamente AJAX e Web-

¹<http://www.nodejs.org>

Sockets através do módulo Socket.IO².

Node.js é um *framework* para IO (*Input/Output*) assíncrono, cuja linguagem de programação é o JavaScript, mas que é executado no servidor ao invés de em um navegador. JavaScript é ideal para IO assíncrono, já que é uma linguagem baseada em eventos e seu modelo de programação é não bloqueante. O motor JavaScript utilizado pelo Node.js é o Google V8³, que é muito rápido, e compila o JavaScript para código nativo antes de sua execução.

A aplicação *Web* foi construída utilizando o framework de aplicações *Web* para Node.js chamado Express⁴, juntamente com a linguagem de *templates* Jade. No lado do cliente, foram utilizados JQuery⁵, JQuery UI⁶ e o 52Framework⁷ para montagem do *layout* das páginas.

O navegador utilizado para os testes da aplicação *Web* foi o Google Chrome 11. Os servidores utilizados para execução dos testes de *performance* foram máquinas virtuais Ubuntu 10.10 no Rackspace Cloud.⁸

Para a execução dos testes de *performance*, foram implementados *scripts* em Node.js para as duas tecnologias. Com WebSockets, foi utilizado o módulo websocket-client, enquanto que para a implementação AJAX foram utilizados os recursos HTTP nativos do Node.js.

2.2 Ambiente

Para a realização dos testes foram criadas duas máquinas virtuais no Rackspace Cloud, sendo que uma máquina foi utilizada para executar a parte servidora da aplicação, e a outra, o script de *benchmark*. As duas máquinas tinham a mesma configuração, sendo ela:

- **Sistema Operacional:** Ubuntu 10.10 (Maverick Meerkat)
- **CPU:** Quad-Core AMD Opteron(tm) Processor 2374 HE, 2,2 Ghz, 512 Cache

²<http://socket.io>

³<http://code.google.com/p/v8/>

⁴<http://expressjs.com/>

⁵<http://jquery.com>

⁶<http://jqueryui.com/>

⁷<http://www.52framework.com/>

⁸<http://www.rackspace.com/cloud/>

- **Memória:** 256 Mb

Por ser uma solução de *Cloud Computing*, não temos algumas informações como velocidade da rede entre as duas máquinas e velocidade dos discos rígidos. Além disso, as máquinas virtuais não utilizam o poder total dos processadores. Quanto maior o plano contratado, maior o uso que se pode fazer do processador.

2.3 Dados a serem coletados

Durante a execução do experimentos, dados serão coletados tanto nos clientes quanto nos servidores. Em ambos os casos os dados são coletados através de impressões na tela com padrão CSV (*Comma Separated Values*) que são redirecionadas para um arquivo de texto no momento da execução.

Os servidores coletam dados em intervalos de 1 em 1 segundo e cada linha tem o seguinte formato:

```
"Timestamp,Número de usuários,Consumo de memória,Consumo de CPU,Total de mensagens,  
Número de mensagens desde o último log"
```

Formato:

1. **Timestamp:** HH:MM:SS:000, Hora, Minutos, Segundos e Milisegundos.
2. **Número de usuários:** O número de usuários conectados à sala de bate-papo no momento da impressão da linha.
3. **Consumo de memória:** A porcentagem de memória que o processo servidor está utilizando no momento da impressão.
4. **Consumo de CPU:** A porcentagem da CPU que está sendo utilizada pelo servidor no momento da impressão.
5. **Total de mensagens:** Soma de todas as comunicações realizadas com todos os clientes
6. **Número de mensagens desde o último log:** Número de comunicações com os clientes desde a impressão da linha de log anterior.

Os clientes coletam dados a cada comunicação com o servidor, seja ela recebendo ou enviando mensagens no bate-papo e cada linha tem o seguinte formato:

"Timestamp, ID do usuário, Operação, Número de usuários, Tamanho da mensagem"

1. **Timestamp:** Igual ao servidor: HH:MM:SS:000 - Hora, Minutos, Segundos e Milisegundos
2. **ID do usuário:** Qual o usuário que enviou a mensagem
3. **Operação:** Qual o tipo da mensagem: RECEIVE quando o cliente está recebendo mensagens do servidor e SEND quando o cliente está enviando mensagens para o servidor.
4. **Número de usuários:** O número de usuários conectados à sala de bate-papo no momento da impressão da linha.
5. **Tamanho da mensagem:** O tamanho total da mensagem referente à essa linha de log. No caso da implementação com AJAX inclui a soma dos cabeçalhos e conteúdos do *request* e *response* e no caso da implementação com Websockets conta o conteúdo e dados utilizados pelo protocolo de comunicação do Socket.IO.

Os dados relativos a entrada e saída da sala de bate-papo são desconsideradas neste experimento.

3 *Execução*

Os *scripts* foram executados de modo a simularem 40 clientes simultâneos, gerando uma mensagem por segundo, em um total de 200 mensagens cada. Lembrando que na aplicação AJAX, além do envio da mensagem, cada cliente fazia a requisição para novas mensagens a cada 0,5 segundos. Além disso, os 40 usuários foram inseridos a uma taxa de 1 novo usuário por segundo, até atingir os 40 usuários.

O número de 40 usuários foi escolhido, pois com mais de 50 usuários a implementação do script de benchmark com WebSockets e mostrou instável, devido ao grande número de mensagens enviadas, e pelo fato de que o script necessitava fazer um *marshalling* de cada mensagem para ver do que se tratava e então calcular os dados relevantes. Nós também optamos por adicionar 1 usuário a cada 1 segundo para não causar uma carga excessiva e repentina no servidor. Além disso, executamos o script de benchmark em uma máquina diferente da máquina servidora, de modo que um processo não interferisse no outro.

Os experimentos foram então executados, sendo que cada um levou cerca de 4 minutos para terminar. Os arquivos de log dos servidores e dos clientes foram coletados e utilizados para a análise dos dados e para geração dos gráficos. No entanto, para gerar gráficos satisfatoriamente, foi necessário selecionar uma amostra, já que esses os experimentos geraram uma quantidade muito grande de dados, principalmente na implementação com WebSockets. Assim, na implementação com WebSockets, 1 em cada 200 linhas dos arquivos de log foi utilizada na geração dos gráficos, enquanto que na implementação com Ajax, que teve menos registros nos logs, 1 em cada 20 linhas do arquivo de log foi utilizada.

4 *Dados obtidos*

A partir da execução dos experimentos e dos 4 arquivos coletados obtivemos os seguintes dados.

4.1 Processo servidor

No processo servidor, foram analisadas o consumo dos processos e a quantidade de comunicação entre cliente e servidor. A seguir temos alguns gráficos que foram gerados a partir dos dados obtidos.

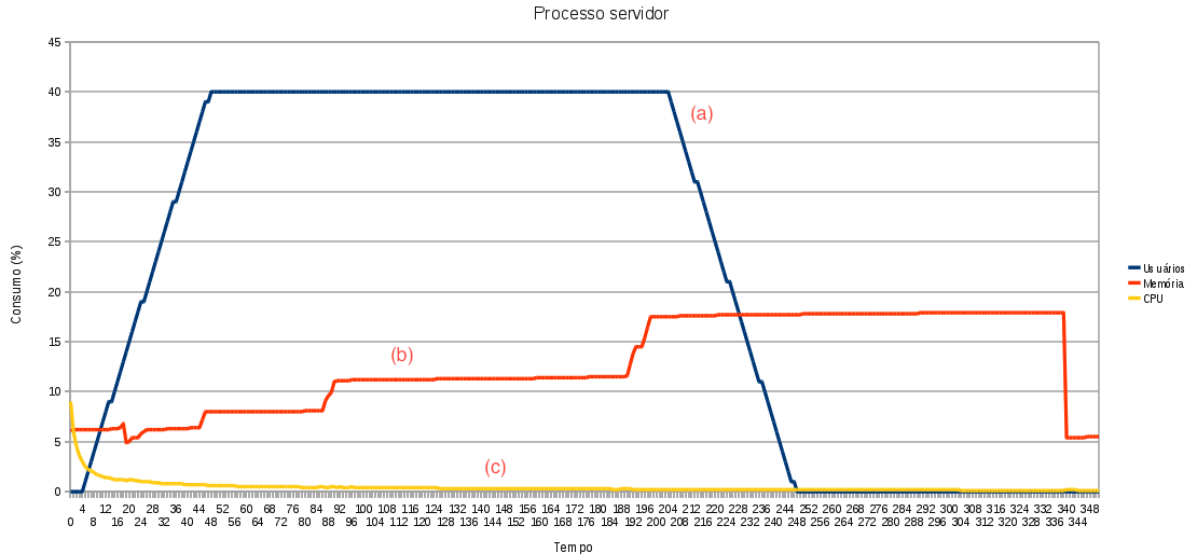


Figura 3: Dados de consumo para o processo servidor com a implementação AJAX. (a) - Número de usuários conectados no servidor ao longo do tempo. (b) Consumo de memória do processo servidor "%" no tempo. (c) - Consumo de CPU do processo servidor em "%" no tempo.

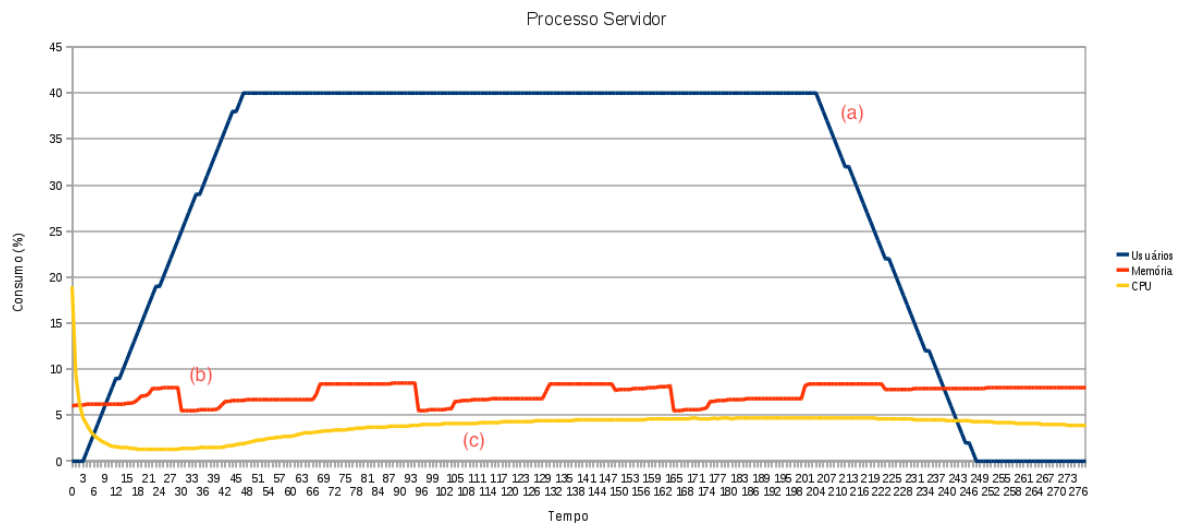


Figura 4: Dados de consumo para o processo servidor com a implementação Websocket. (a) - Número de usuários conectados no servidor ao longo do tempo. (b) Consumo de memória do processo servidor "%" no tempo. (c) - Consumo de CPU do processo servidor em "%" no tempo.

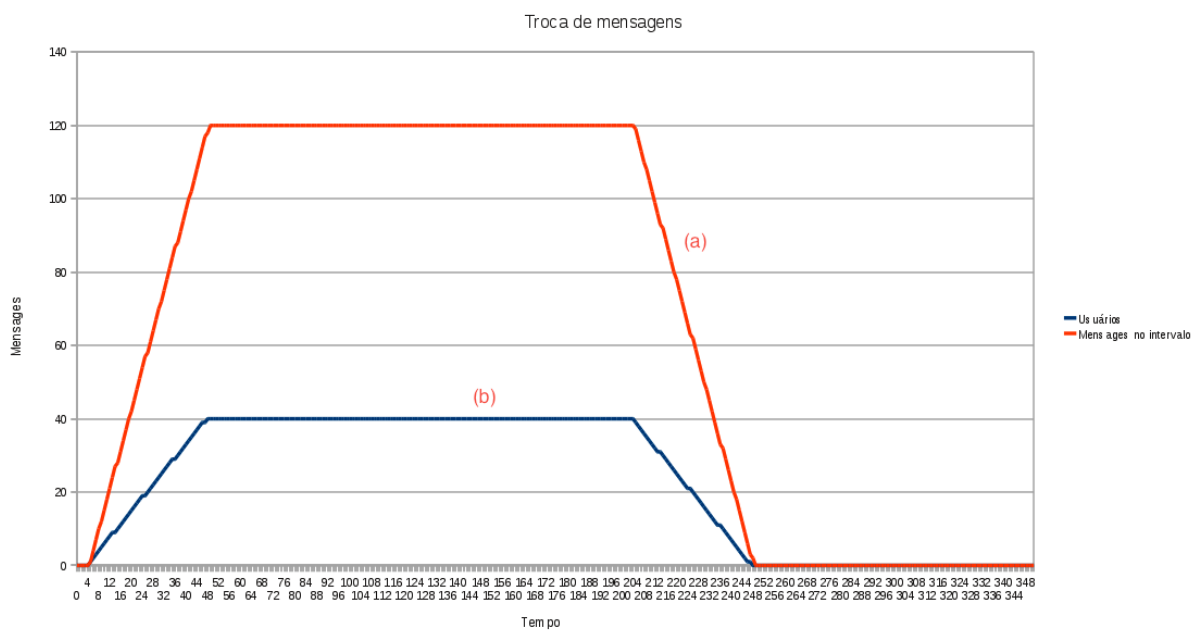


Figura 5: Quantidade de comunicação entre clientes e servidor na implementação com AJAX. (a) - Quantidade de comunicações por segundo ao longo do tempo da execução do experimento. (b) - Quantidade de usuários conectados no servidor ao longo do tempo.

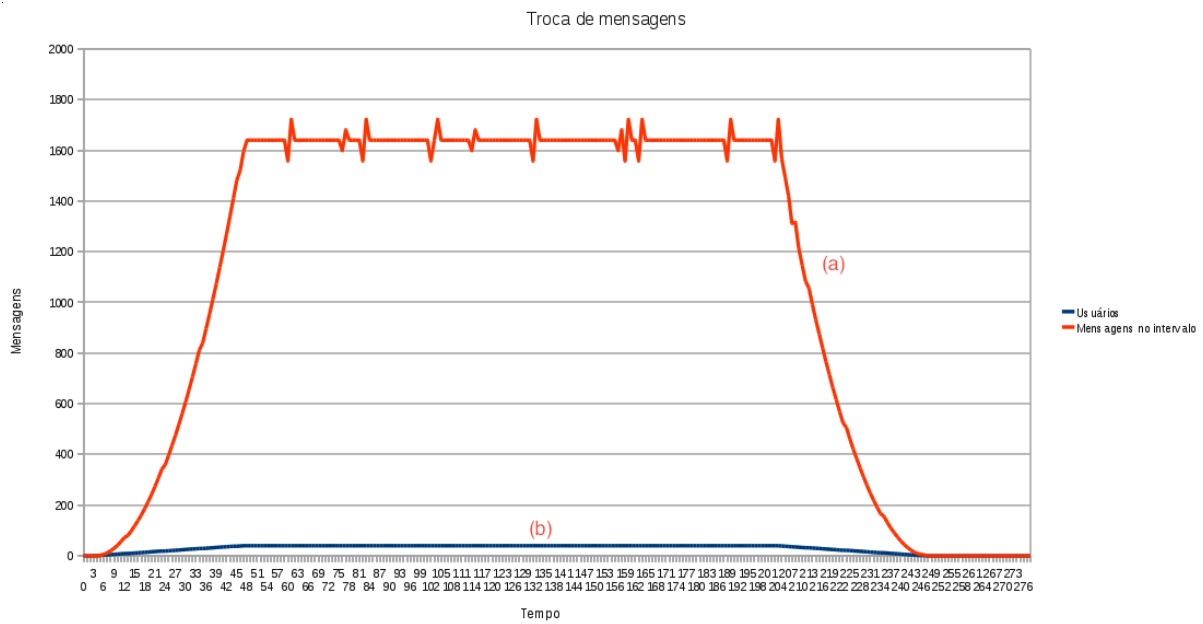


Figura 6: Quantidade de comunicação entre clientes e servidor na implementação com Websocket. (a) - Quantidade de comunicações por segundo ao longo do tempo da execução do experimento. (b) - Quantidade de usuários conectados no servidor ao longo do tempo.

4.2 Processo cliente

Nos processos cliente foi analisado principalmente o tamanho de cada comunicação entre cliente e servidor. O dados da Tabela 1 foram extraídos a partir do conjunto total de dados, enquanto os gráficos das Figuras 7 e 8 foram gerados a partir de amostras em função do número muito grande de dados.

Dado	AJAX	Websocket
Tempo inicial:	21:39:24	21:35:33
Tempo final:	21:43:26	21:39:56
Duração do experimento:	4m 2s (242s)	4m 22s (263s)
Número de operações SEND:	8000	8000
Número de operações RECEIVE:	16000	274115
Soma dos tamanhos dos SEND:	6178000 (bytes)	910255(bytes)
Soma dos tamanhos dos RECEIVE:	41394106 (bytes)	37250287 (bytes)
Média de tamanho dos SEND:	772,25 (bytes)	113,78 (bytes)
Média de tamanho dos RECEIVE:	2587,13 (bytes)	135,89 (bytes)
Variância dos tamanhos dos SEND:	842,02	865,14
Variância dos tamanhos dos RECEIVE:	158728,67	838,24
Desvio padrão dos tamanhos dos SEND:	29,02	29,41
Desvio padrão dos tamanhos dos RECEIVE:	398,41	28,95
Total de dados trafegados:	47572106 (bytes)	38160542 (bytes)
Taxa de dados:	196578,95 (bytes/s)	145097,12 (bytes/s)

Tabela 1: Dados coletados a partir de todo conjunto de dados do processo cliente implementado com AJAX

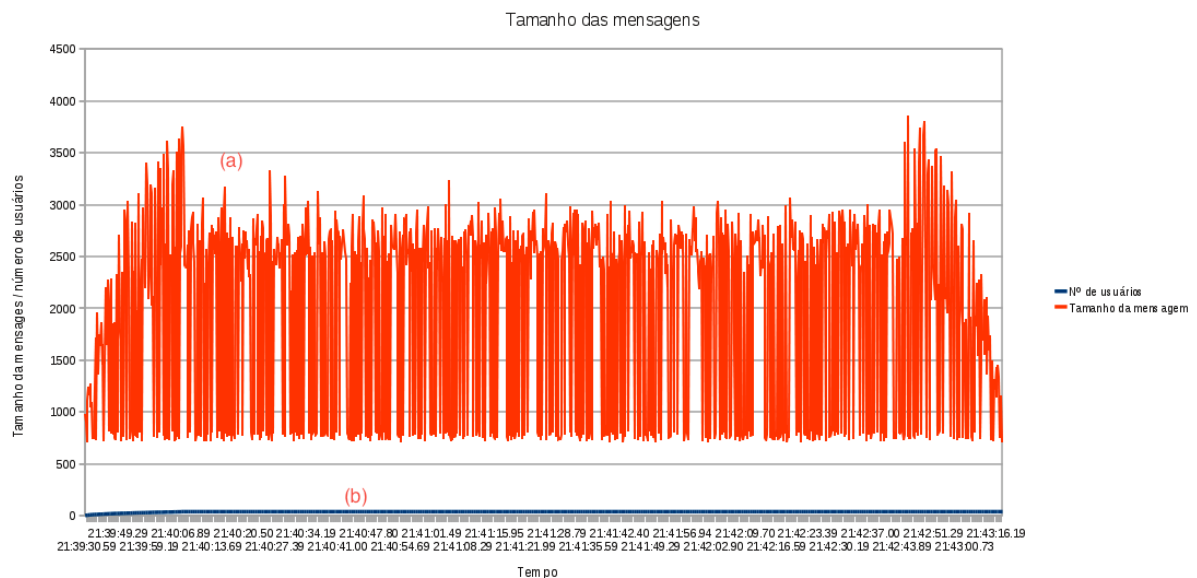


Figura 7: Tamanho total das comunicações na implementação AJAX. (a) - Tamanho das mensagens (bytes) tanto de *sends* quanto de *requests* ao longo do tempo. (b) - Número de usuários conectados no servidor ao longo do tempo. Este gráfico foi gerado a partir de uma amostra da massa de dados.

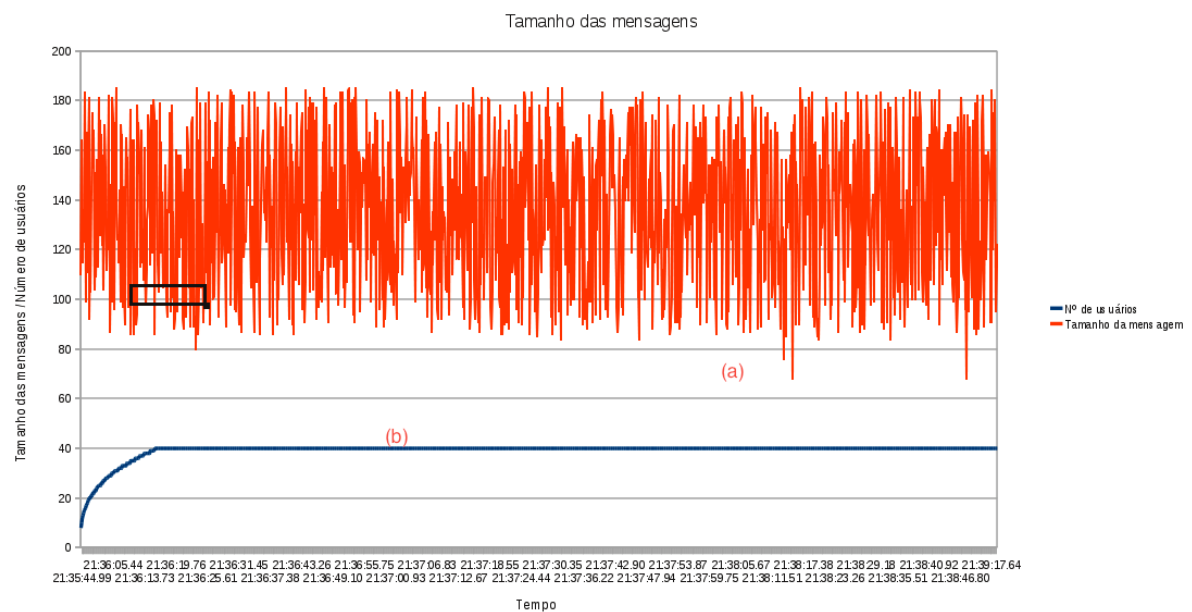


Figura 8: Tamanho total das comunicações na implementação com Websocket. (a) - Tamanho das mensagens (bytes) tanto de *sends* quanto de *requests* ao longo do tempo. (b) - Número de usuários conectados no servidor ao longo do tempo. Este gráfico foi gerado a partir de uma amostra da massa de dados.

5 *Resultados obtidos*

A partir dos dados coletados, podemos verificar que ambas as tecnologias tem vantagens e desvantagens. A tecnologia WebSocket proporciona a transmissão de mensagens entre os usuários do chat em tempo real, mas ao mesmo tempo, gera uma quantidade muito maior de mensagens quando comparado a uma implementação comum em AJAX, como pode ser visto na Tabela 1 onde temos que o *script* da implementação AJAX teve 16000 mensagens de RECEIVE enquanto o *script* da implementação Websocket tragefou 274115 mensagens deste mesmo tipo, muito embora cada mensagem tenha um tamanho menor nesta última implementação.

Isso se deve principalmente ao tamanho dos cabeçalhos que são transmitidos em cada requisição AJAX. Na Tabela 1 nota-se que enquanto uma mensagem de SEND da implementação AJAX possui em média 772,25 bytes, as mensagens de SEND da implementação Websocket possuem em média 113,78 bytes. Ambas possuem uma variação pequena que pode ser confirmado pelos seus desvios padrão 29,02 e 29,41 respectivamente e que pode ser explicada pelo tamanho do conteúdo randômico que foi utilizado nas implementações dos *scripts* de teste. Com esses dados, calculamos que a implementação Ajax trafegou um total de 45,37 Mb de dados, enquanto que a implementação em WebSockets trafegou 36,39 Mb. No entanto, é importante ressaltar que essa diferença entre as duas implementações tende a se tornar menos significativa conforme o tamanho do conteúdo das mensagens aumenta.

Pelos experimentos realizados, percebemos que a implementação em WebSockets como está se torna praticamente inviável, uma vez que o próprio *script* de *benchmark* teve trabalho para tratar todas as mensagens que chegavam. O servidor no entanto, não foi muito afetado por essa carga, uma vez que ele simplesmente despacha as mensagens, sem ter que processá-las. Essa característica fez com que o servidor em WebSockets utilize-se menos memória quanto comparado a implementação em Ajax, que precisa guardar as mensagens em memória até que os usuários faziam uma requisição Ajax para obtê-las. Em relação ao uso de CPU a implementação com Websockets

foi consideravelmente maior, embora os experimentos não tivessem objetivo de testar os limites de carga de cada implementação, notamos pela comparação das curvas (c) nas Figuras 3 e 4 que a diferença foi substancial, embora mais testes sejam necessários para se estabelecer até que ponto este consumo afetar o servidor com o crescimento do número de usuários e/ou mensagens.

Notamos que a implementação em Ajax como está, utilizando um *array* de mensagens, que é consultado pelos usuários de tempo em tempo, cria um mecanismo de *buffer/cache*, que melhora o uso do sistema trazendo vantagens quando comparados a implementação com WebSockets. Em função deste efeito de *buffer* nota-se através dos dados coletados que o tamanho das mensagens de RECEIVE virão muito no implementação com AJAX enquanto a implementação com AJAX essas mensagens possuem um tamanho mais bem comportado. Além disso, um tempo de 0,5 segundos para a obtenção das mensagens não é tão rápido quando na versão em WebSockets, mas esse tempo pode é bastante aceitável para aplicações como um bate-papo.

Seria diferente se estivéssemos comparando um jogo on-line onde diversos jogadores podem interagir uns com os outros, neste caso a implementação Websockets traria uma experiência de uso muito melhor do que a AJAX.

Por outro lado, a implementação em WebSockets poderia ser modificada, de modo a guardar as mensagens para cada usuário por um tempo, e estão despachá-las em lote para os usuários, em um mecanismo parecido com o Ajax. Teoricamente, isso melhoraria o desempenho da aplicação WebSockets, tirando ainda o *overhead* do HTTP, como latência e cabeçalhos desnecessários. Isso no entanto, eliminaria uma das principais vantagens desta API que é a comunicação em tempo real.

É importante salientar que a tecnologia de Websockets não foi criada com a intenção de substituir as técnicas AJAX, e sim para explorar alguns de seus pontos fracos. Portanto o que deve ser feito quando se planeja construir uma aplicação é ponderar sobre as vantagens e desvantagens de cada uma em relação as necessidades específicas do seu projeto.