

# WEB EM TEMPO REAL: Soluções para comunicação em tempo real entre clientes e servidores.

Iosley Carlos Silva

12 de junho de 2017

**Resumo:** Lorem

**Palavras-chave:** web, websockets, sse, ajax.

## 1 INTRODUÇÃO

A internet<sup>1</sup> como é hoje passou por diversas transformações com a melhoria dos hardwares<sup>2</sup> necessários para seu funcionamento e a definição de protocolos formais para a intercomunicação entre diversos dispositivos (AGHAEI, 2012).

Porem, a transformação mais notável desde então, foi no tratamento dos dados. Hoje as informações estão acessíveis de qualquer lugar do mundo, isto trouxe também um aumento na retenção da informação e uma mudança drástica na maneira de manipular e transmitir estas informações (LEINER et al., 2009).

Hoje as páginas web são mais dinâmicas podendo ser desenvolvidas aplicações modernas, que antes precisavam ser instaladas no computador, rodando direto do navegador<sup>3</sup>, proporcionando a mesma experiência experimentada com as aplicações nativas (GARRETT, 2005).

Para ter uma experiência fluida nestas aplicações, os navegadores precisam se comunicar com os servidores<sup>4</sup> de maneira rápida e eficiente. Para isto, são usadas técnicas para a transmissão e recebimento de informações em tempo real. Estas técnicas mantem as informações sincronizadas entre a aplicação cliente e o servidor de maneira transparente para o usuário (OFFUTT, 2002).

---

<sup>1</sup> Conjunto mundial de redes de computadores.

<sup>2</sup> Parte física de um computador

<sup>3</sup> Programa de computador que possibilita a interação com documentos virtuais da Internet.

<sup>4</sup> Sistema de computação centralizada que fornece serviços a uma rede de computadores.

Estes avanços proporcionaram uma participação muito maior dos usuários que agora deixam de ser meros consumidores de conteúdo e passam a ser os maiores produtores de conteúdo multimídia (AGHAEI, 2012).

No entanto, entende-se que mudanças foram necessárias para acompanhar esta evolução. Mudanças que pudessem coexistir com a maneira já existente de transmissão de dados ao passo que tornasse mais eficiente e mais rápida a transmissão da informação.

## 2 REFERENCIAL TEÓRICO

Em agosto de 1962, foi previsto um conjunto global interconectado de computadores através do qual todos poderiam acessar dados e programas, descrito em memorandos escritos por J.C.R. Licklider do MIT<sup>5</sup> quando este discutiu sobre o conceito de “Rede Galáctica”, conceito este muito semelhante com a internet de hoje (LEINER et al., 2009, p. 2).

Para a intercomunicação de computadores de diferentes fabricantes, se fez necessário uma arquitetura aberta que foi primariamente introduzida por Kahn Shortly da DARPA<sup>6</sup> em 1972 com o *Netware Core Protocol* (NCP), um protocolo para comunicação bi-direcional identificada por um par de números de soquetes<sup>7</sup> (LEINER et al., 2009, p. 4).

Sucedendo e substituindo o protocolo NCP, o Protocolo de Controle de Transmissão (TCP)<sup>8</sup>, vinha sendo implementado desde 1980 mas foi somente em 1983 que a transição definitiva aconteceu, exigindo que todos os *hosts*<sup>9</sup> convertessem simultaneamente para que continuassem funcionando (LEINER et al., 2009, p. 7).

Nesta época, já era possível via Internet, entrar em sessões com máquinas remotas e trocar mensagens, porem, de acordo com Aghaei (2012), somente em 1989, Tim Berners-Lee sugere a criação de um espaço de hipertexto<sup>10</sup> global na qual qualquer informação acessível seria referida por um único Identificador de Documento Universal (UDI), este espaço seria posteriormente conhecido por Word Wide Web (WWW) ou simplesmente Web.

Em 1991 Tim Berners-Lee do CERN<sup>11</sup>, na Suíça, apresentou um novo sistema de informação baseado na Internet (WWW) tornando-se possível criar servidores de informação,

<sup>5</sup> Instituto de Tecnologia de Massachusetts é uma universidade privada de pesquisa localizada em Cambridge, Massachusetts, Estados Unidos.

<sup>6</sup> Agência de Projetos de Pesquisa Avançada de Defesa dos Estados Unidos.

<sup>7</sup> Mecanismo de comunicação que permite a troca de mensagens entre os processos de uma máquina/aplicação servidor e uma máquina/aplicação cliente.

<sup>8</sup> Padrão que define como estabelecer e manter uma conversa de rede através da qual os programas de aplicativos podem trocar dados.

<sup>9</sup> É qualquer máquina ou computador conectado a uma rede, podendo oferecer informações, recursos, serviços e aplicações aos usuários ou outros nós na rede.

<sup>10</sup> Apresentação de informações, organizada de tal maneira que o leitor tem liberdade de escolher vários caminhos, a partir de sequências associativas possíveis entre blocos vinculados por remissões, sem estar preso a um encadeamento linear único.

<sup>11</sup> Organização Europeia para a Pesquisa Nuclear, é o maior laboratório de física de partículas do mundo.

onde se incluem textos, imagens e multimídia (GOETHALS; AGUIAR; ALMEIDA, 2000).

Segundo Aghaei (2012), as principais tecnologias da Web eram:

**Hypertext Markup Language (HTML)** que segundo Berners-Lee e Connolly (1993): é uma linguagem de marcação usada para criar documentos de hipertexto;

**Identificador Uniforme de Recurso (URI)** que segundo Connolly e Masinter (2000): é o identificador de fragmento que designa o elemento com o nome correspondente;

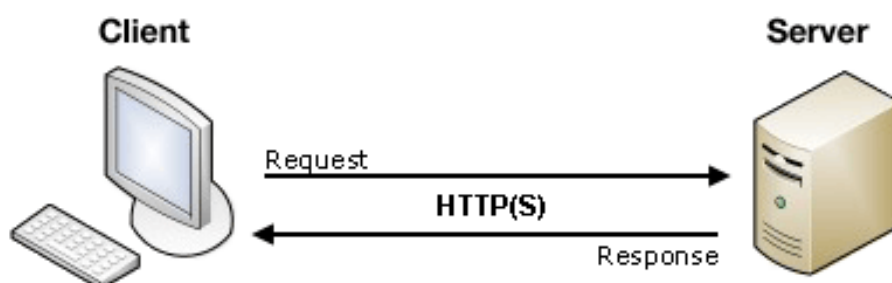
**Protocolo de Transferência de Hipertexto (HTTP)** que segundo Fielding, Irvine e Gettys (1999, p. 7): “O HTTP é um protocolo de nível de aplicação para sistemas de informação distribuídos, colaborativos e hipermídia”.

## 2.1 O protocolo HTTP

Em uso desde 1990, teve sua primeira versão referida como HTTP / 0.9. Era um protocolo simples para transferência de dados através da internet. A partir da versão 1.0, o protocolo foi melhorado permitindo modificadores sobre a semântica “requisição / resposta” para que duas aplicações determinassem as capacidades verdadeiras de cada uma (FIELDING; IRVINE; GETTYS, 1999, p. 7).

No modelo HTTP padrão, um servidor não inicia uma conexão com um cliente, enviando respostas somente quando solicitado. Assim, não é possível que um servidor envie eventos assíncronos para aplicações clientes, forçando o cliente à pesquisar periodicamente por novos conteúdos no servidor, o que consome uma quantidade significativa de tráfego de dados e reduz a capacidade de resposta da aplicação, pois o servidor precisa ser requisitado para enviar as atualizações (LORETO et al., 2011).

Figura 1 – Modelo de requisição HTTP



Fonte – Mills (2017)

## 2.2 A Evolução da WEB

A Web no início era somente leitura, estática e mono-direcional. O principal objetivo era publicar informações para e estabelecer uma presença on-line. Os sites<sup>12</sup> eram estáticos e não interativos. Os Usuários não poderiam fazer contribuições nem interagir com os sites, sendo estes meros panfletos digitais (AGHAEI, 2012, p. 2-3).

A medida que as os desenvolvedores começaram a criar páginas cada vez mais dinâmicas, foi se fazendo necessário o uso de técnicas para melhorar a comunicação. Em 1999 quando o Internet Explorer 5<sup>13</sup> implementou o AJAX<sup>14</sup> pela primeira vez (ASLESON; SCHUTTA, 2006), as páginas web já podiam ser muito mais flexíveis e rápidas, já não era mais preciso sair da página para buscar informação no servidor, mas ainda não havia uma maneira do servidor enviar mensagens espontaneamente para o cliente.

## 2.3 Soluções paliativas

Várias técnicas foram implementadas nos últimos anos para permitir que um servidor web envie atualizações para clientes. Dentre as principais, destacam-se:

**HTTP Polling:** Consiste de uma sequência de requisições que o cliente emite para o servidor com intervalo determinado, recebendo uma resposta vazia caso novas mensagens não estejam disponíveis (PIMENTEL; NICKERSON, 2012).

**Long Polling HTTP:** O servidor tenta “manter aberta” (não responder imediatamente) a cada solicitação HTTP, respondendo apenas quando há novos dados para entregar. Desta forma, existe sempre um pedido pendente ao qual o servidor pode responder com o objetivo de disparar eventos à medida que ocorrem, minimizando assim a latência na entrega de mensagens (AGHAEI, 2012).

**HTTP Streaming<sup>15</sup>:** O servidor mantém uma solicitação aberta indefinidamente, ou seja, nunca finaliza a resposta ou fecha a conexão, mesmo depois de enviar dados para o cliente.

Segundo Loreto et al. (2011, p. 3), “Esses mecanismos podem fornecer atualizações aos clientes de forma mais atempada, evitando a latência experimentada pelas aplicações clientes devido à frequente abertura e fechamento de conexões necessárias para periodicamente pesquisar dados”.

É importante entender que estas soluções nem sempre são eficazes e para alguns casos uma busca periódica pode até ser mais eficiente, como é o exemplo de aplicações com alto

<sup>12</sup> Local na Internet identificado por um nome de domínio, constituído por uma ou mais páginas de hipertexto.

<sup>13</sup> Uma série de navegadores web gráficos desenvolvidos pela Microsoft.

<sup>14</sup> Técnicas para programação que utilizam tecnologias como JavaScript e XML para carregar informações de forma assíncrona.

volume de mensagens, onde o Long Polling não oferece melhorias de desempenho se comparado a sondagem tradicional porque a reconexão é constante (WANG; SALIM; MOSKOVITS, 2013).

Já o Streaming, mesmo sendo uma grande solução, entrega mensagens de maneira imprevisível. Alguns proxies<sup>16</sup> e firewalls<sup>17</sup> podem guardar em memória a resposta o que pode resultar em uma maior latência, não sendo recomendada para redes onde existam firewalls ou proxies (WANG; SALIM; MOSKOVITS, 2013, p. 6).

## 2.4 Soluções atuais

Para eliminar estes problemas, a sessão de conexão do HTML5<sup>18</sup> inclui o WebSocket. Segundo Pimentel e Nickerson (2012, p. 47, Tradução nossa), “O protocolo WebSocket fornece um canal de comunicação bi-direcional, que opera através de um único soquete e pode ajudar a criar aplicativos escaláveis em tempo real na Web”.

O protocolo consiste de um *handshake*<sup>19</sup> de abertura seguido pelo enquadramento básico da mensagem, em camadas sobre TCP para permitir uma comunicação bidirecional entre um cliente e servidor. Utiliza o HTTP como uma camada de transporte para se beneficiar da infraestrutura existente, porém não se limita ao HTTP e implementações futuras podem usar um *handshake* mais simples sem reinventar todo o protocolo (SAINT-ANDRE, 2011).

Vale ressaltar que, Websockets se mostra o melhor cenário para uma conexão full-duplex<sup>20</sup> entre cliente e servidor, no entanto, se o serviço apenas transmite informações para seus clientes e não requer qualquer interatividade, usar a Interface de Programação de Aplicativos (API) de *EventSource* fornecida pelo *Server-Sent Events* (SSE), que faz parte da especificação HTML5, pode ser uma boa opção pois é possível usar o SSE como uma sintaxe comum interoperável para *Ajax Polling*, *Long Polling* e *Streaming* (WANG; SALIM; MOSKOVITS, 2013, p. 10-11).

## 3 METODOLOGIA

O estudo aborda conhecimento a respeito da comunicação em tempo real entre servidores e clientes de aplicações web, para isso se fez necessário direcionar a abordagem em base da utilização de material teórico por meio de uma pesquisa documental, que segundo (MARCONI; LAKATOS, 2003), tem documentos como fonte primária de informação.

Buscando uma melhor análise histórica do desenvolvimento de aplicações em tempo real para a web, este trabalho é pautado na investigação qualitativa a respeito do tema proposto,

<sup>16</sup> Intermediários entre o usuário e seu servidor.

<sup>17</sup> Dispositivo de segurança da rede que monitora o tráfego de rede de entrada e saída e decide permitir ou bloquear tráfegos específicos de acordo com um conjunto definido de regras de segurança.

<sup>18</sup> Quinta versão da linguagem HTML.

<sup>19</sup> Processo pelo qual duas máquinas afirmam uma a outra que a reconheceu e está pronta para iniciar a comunicação.

<sup>20</sup> Quando receptor e transmissor podem transmitir dados simultaneamente.

desenvolvendo conceitos, ideias e entendimentos a partir de fontes secundárias (PRODANOV; FREITAS, 2013).

Do ponto de vista de sua natureza, está é uma pesquisa básica pois tem como objetivo gerar conhecimentos úteis para o avanço da ciência sem aplicação prática prevista (PRODANOV; FREITAS, 2013).

Visando compreender as limitações e em busca por soluções para comunicação em tempo real na web, este estudo tem cunho exploratório pois proporciona maior familiaridade com o tema, envolvendo levantamento bibliográfico e documental (GIL, 2002).

#### 4 RESULTADOS E DISCUSSÕES

Nos primeiros dias da *World Wide Web*, um navegador precisava apresentar apenas alguns tipos de dados aos usuários. Para não ser limitado por esses tipos de dados, os desenvolvedores trabalharam duro para estender os navegadores para que dados em outros formatos pudessem ser renderizados no computador cliente. Uma maneira de resolver o problema era permitir que o navegador, ao reconhecer um arquivo recebido de um tipo específico, iniciar um aplicativo separado na máquina cliente para renderizar o conteúdo. Desde que este aplicativo auxiliar tenha sido instalado no computador cliente, o navegador iniciará o programa e enviará o arquivo recebido para esse programa (ZAMMETTI, 2007).

A primeira solução para tornar a web mais dinâmica foi o “*Common Gateway Interface*” (CGI), que permite a criação de programas que executam quando um usuário faz uma requisição. Porém, CGI não é a solução mais segura para criação de páginas web pois permite que seja executado um programa em seu sistema operacional e usuários maliciosos podem explorar isto com algum exploit<sup>21</sup> e executar operações indesejadas (ASLESON; SCHUTTA, 2006).

Ainda segundo Asleson e Schutta (2006), em maio de 1995 John Gage e Andreessen anunciam o nascimento da linguagem de programação Java. O navegador Netscape era dominante na época e ofereceria suporte para esta nova linguagem. Dentro de alguns meses, milhares de pessoas já haviam baixado o Java em seus computadores, abrindo novos caminhos para páginas web dinâmicas.

Applets<sup>22</sup> permitem que pequenas aplicações Java possam ser incluídas nas páginas web e executadas através da *Java Virtual Machine* (JVM). Applets são executadas no modelo de segurança de “caixa de areia” (*sandbox*), não podem carregar bibliotecas nativas e são tipicamente impedidas de ler ou gravar no disco (ASLESON; SCHUTTA, 2006).

Ainda em maio de 1995, Brendan Eich, um funcionário da Netscape<sup>23</sup> na época, desen-

---

<sup>21</sup> Uma sequência de comandos que tomam vantagem de um defeito, falha ou vulnerabilidade a fim de causar um comportamento acidental ou imprevisto.

<sup>22</sup> Pequeno software que executa uma atividade específica dentro de outro programa maior.

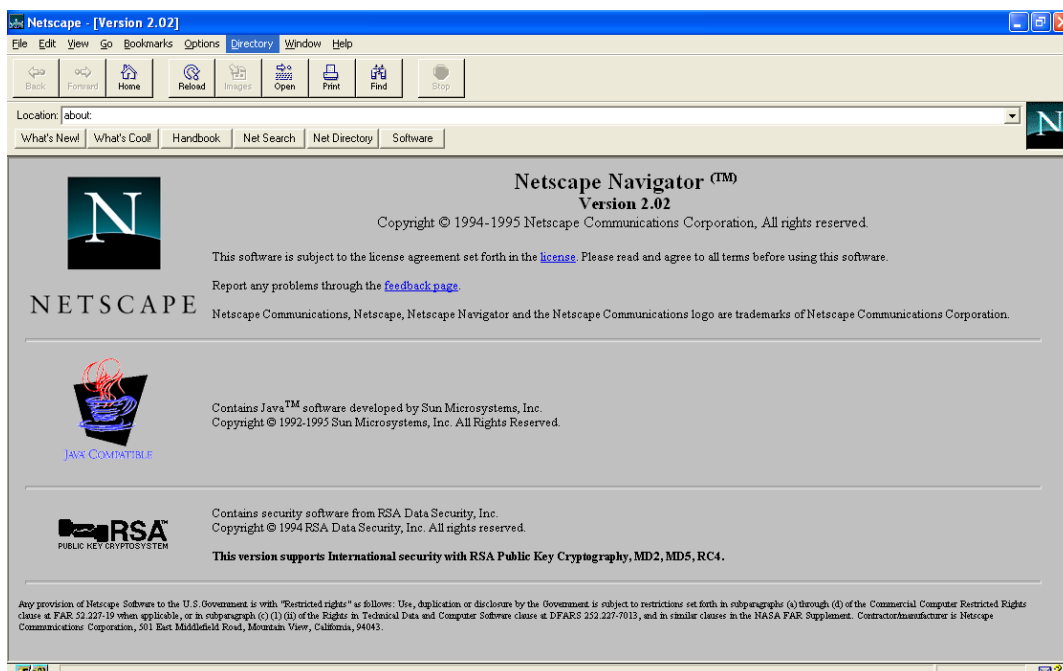
<sup>23</sup> Empresa de serviços de computadores nos EUA.

volveu uma linguagem de script em dez dias que se tornou conhecida como Mocha. Este nome original foi dado pelo fundador da Netscape. Pouco depois, o nome foi avaliado e renomeado como Livescript. Mais tarde naquele ano em dezembro, a Netscape recebeu uma licença de marca registrada da Sun<sup>24</sup>. Desta vez, o nome mudou para JavaScript (NEER; III, 2013).

JavaScript foi concebido para fins muito diferentes do Java, essencialmente para funcionar como uma linguagem de programação integrada em documentos HTML e não como uma linguagem para escrever applets que ocupam uma área retangular fixa na página (GOODMAN, 2007). O JavaScript tinha um pequeno vocabulário e um modelo de programação mais facilmente digerível que o Java, com sua abordagem orientada a objetos. Nesse contexto Crockford (2008) afirma que com JavaScript é possível programar sem saber muito sobre a linguagem, ou mesmo saber muito sobre programação.

A primeira versão, o JavaScript 1.0, estreou no navegador Netscape 2 ainda em 1995. No momento do lançamento do JavaScript 1.0, o Netscape dominava o mercado de navegadores. A Microsoft estava lutando para recuperar seu próprio navegador, Internet Explorer e seguiu rapidamente a liderança da Netscape ao lançar sua própria linguagem VBScript<sup>25</sup>, juntamente com uma versão do JavaScript chamada JScript, com a entrega do Internet Explorer 3 (KEITH; SAMBELLS, 2010).

Figura 2 – Navegador Netscape 2



Fonte – SoftWsp (2016)

A Netscape enviou a linguagem para padronização à Associação Europeia de Fabricante de Computadores (ECMA) e devido a problemas de marca registrada, a versão padronizada

<sup>24</sup> Fabricante de computadores, semicondutores e software com sede em Santa Clara, Califórnia, no Silicon Valley.

<sup>25</sup> Versão interpretada da linguagem Visual Basic para construção dinâmica de página HTML

da linguagem estava presa com o nome estranho “ECMAScript”. Pelos mesmos motivos de marca registrada, a versão da Microsoft da linguagem É formalmente conhecida como “JScript” (FLANAGAN, 2011).

O ECMAScript como foi padronizado não se destina a ser computacionalmente auto-suficiente, espera-se que o ambiente computacional de um programa ECMAScript forneça certos objetos específicos do ambiente; um navegador da Web fornece um ambiente de *host* para computação do lado do cliente, incluindo, por exemplo, objetos que representam janelas, menus, pop-ups<sup>26</sup>, caixas de diálogo, áreas de texto, âncoras, quadros, histórico, cookies<sup>27</sup> e entrada/saída; um servidor web fornece um ambiente de *host* diferente para a computação do lado do servidor, incluindo objetos que representam requisições (ECMAScript; ASSOCIATION et al., 2016).

#### 4.1 Pré Ajax

Em 1995, já era possível construir *Single Page Applications* (SPAs) via frames/framesets<sup>28</sup> e `url="javascript:..."`. Requisições assíncronas também eram possíveis utilizando JavaScript e manipulando elementos HTML que realizavam requisições HTTP; praticamente qualquer tag que possa ser configurado para fazer referência a um URL pode ser empregado para tarefas de comunicação com base em JavaScript (POWELL, 2008).

Ainda segundo Powell (2008), utilizando-se de tags que referenciam URL é possível gerar uma requisição de via única ao servidor para indicar que algum evento aconteceu através de campos adicionados ao URL (*Query Strings*). Para isso, utiliza-se de parâmetros passados via URL por uma tag<sup>29</sup> (*img* por exemplo), cabe ao servidor tratar a requisição e retornar o dado esperado ou uma resposta vazia com status 204, informando ao cliente que a solicitação ocorreu sem erros mas que não há conteúdo na resposta.

Quanto ao uso da tag *img* para comunicação bi-direcional Powell (2008) completa:

Parece que o uso de uma imagem provavelmente não é a melhor maneira de transmitir informações bidirecionais. Considere que, se você pedir uma imagem, você estará recebendo uma imagem, provavelmente em formato GIF, JPEG ou PNG para exibição. Como exemplo, você pode solicitar ao usuário alguns dados e, em seguida, gerar uma imagem personalizada para eles. A transmissão dos dados fornecidos pelo usuário é através da sequência de consulta como anteriormente, mas desta vez o servidor responderá não com um código 204, mas com uma imagem real para usar. Você pode usar o DOM e inseri-lo na página.

<sup>26</sup> Janela que abre no navegador da internet quando se acessa uma página na web ou algum link de redirecionamento.

<sup>27</sup> Grupo de dados trocados entre o navegador e o servidor de páginas, colocado num arquivo de texto criado no computador do utilizador.

<sup>28</sup> Divisões internas dentro de uma mesma janela do navegador, onde são carregados outros documentos HTML.

<sup>29</sup> Estruturas de linguagem de marcação contendo instruções, tendo uma marca de início e outra de fim para que o navegador possa renderizar uma página.



Visto que um navegador permanecerá na mesma página quando receber uma resposta HTTP de status 204, ele pode ser usado para fingir ir a uma URL apenas para enviar alguns dados; isto é feito em JavaScript com uma atribuição direta para *window.location* enviando os dados pela URL, através das *Query Strings*. Porém as *Query Strings* são limitadas ao tamanho máximo da URL permitido pelo navegador.

Para envio de uma grande quantidade de dados o desenvolvedor precisaria realizar uma requisição HTTP POST com uma técnica de iframes<sup>30</sup> escondidos. Esta técnica consiste da criação de campos de formulário a serem enviados com o formulário inserido no iframe. Uma vez que o formulário é preenchido com os dados desejados, é desencadeado o envio do formulário via JavaScript.

A utilização de iframe é flexível no que pode receber em comparação com alguns dos outros métodos, permitindo a comunicação bi-direcional através de respostas em XHTML<sup>31</sup>, XML<sup>32</sup>, JSON<sup>33</sup> ou outro formato de codificação a serem tratado com JavaScript no navegador (POWELL, 2008).

## 4.2 Ajax

O termo Ajax é um acrônimo para JavaScript assíncrono e XML que segundo Garrett (2005), descreve uma maneira de realizar uma comunicação HTTP a partir de uma aplicação JavaScript em páginas web.

Ajax é uma mistura do antigo com o novo, porque as tecnologias já existentes são combinadas em técnicas que pouco se considerava anteriormente, trazendo uma nova geração de aplicações e idéias (GROSS, 2006).

Nesse contexto, Asleson e Schutta (2006) afirmam:

Honestamente, o Ajax não é nada novo. Na verdade, a tecnologia “mais nova” relacionada ao termo - o objeto XMLHttpRequest (XHR) - tem ocorrido desde o Internet Explorer 5 (lançado na primavera de 1999) como um controle ActiveX<sup>34</sup>. No entanto, o que é novo é o nível de suporte do navegador. Originalmente, o objeto XHR era suportado apenas no Internet Explorer (limitando assim seu uso), mas, com o Mozilla 1.0 e Safari 1.2, o suporte é generalizado.

Criado pela Microsoft no fim da década de 90, XMLHttpRequest é a interface através da qual o navegador pode fazer requisições HTTP com JavaScript. Quando a interface XMLHttpRequest

<sup>30</sup> Elemento HTML que permite carregar as informações de um documento HTML separado em um documento HTML existente.

<sup>31</sup> Linguagem Extensível para Marcação de Hipertexto.

<sup>32</sup> Linguagem de marcação para a criação de documentos com dados organizados hierarquicamente.

<sup>33</sup> Notação de Objetos JavaScript, é uma formatação leve de troca de dados.

<sup>34</sup> Framework criado pela Microsoft que adapta as antigas versões das plataformas COM - Component Object Model e OLE - Object Linking and Embedding para conteúdo disponível online, especialmente aplicações web e cliente/servidor.

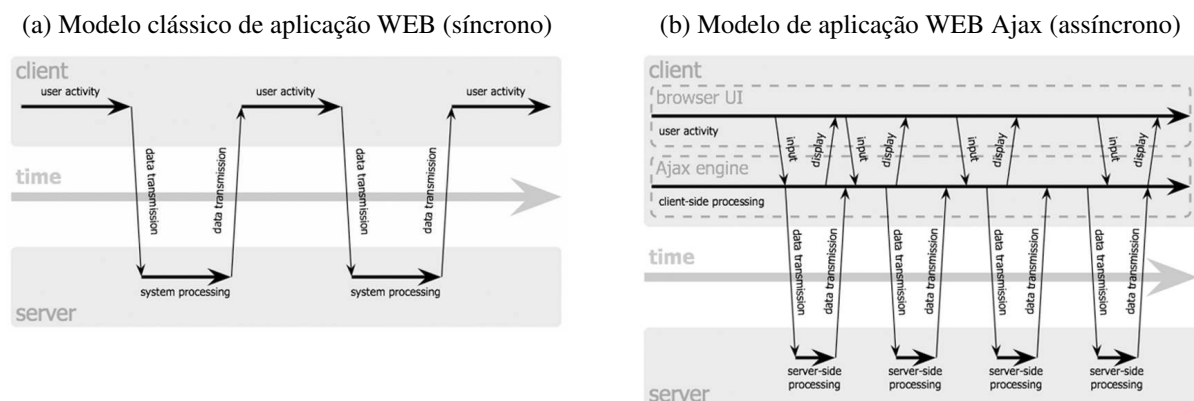
quest foi adicionada ao Internet Explorer, ela permitiu que os desenvolvedores fizessem coisas com o JavaScript que antes era muito difícil (HAVERBEKE, 2014).

Segundo Grigorik (2013), o XHR não apenas habilitou a comunicação assíncrona no navegador, mas também tornou mais simples esta, pois o XHR é uma API de aplicativos fornecida pelo navegador, o que significa que o navegador cuida automaticamente de todos os gerenciamentos de conexão de baixo nível.

Ainda segundo Grigorik (2013), o XHR tornou-se um padrão de fato em todos os principais navegadores porem o documento de especificação oficial só foi publicado em 2006 pelo *World Wide Web Consortium (W3C)*<sup>35</sup>, bem depois de ter sido amplamente utilizado.

De fato, o envio de dados assíncronos ao servidor e a recepção de dados adicionais são o objetivo final do Ajax. Todos os processos do Ajax começam com uma conexão com o servidor. As conexões ao servidor geralmente são organizadas através do objeto XMLHttpRequest (RESIG, 2007).

Figura 3 – Comparação entre o modelo clássico e o modelo Ajax



Fonte – Garrett (2005)

As versões iniciais do XHR proporcionaram recursos limitados: transferências de dados baseadas em texto, suporte restrito para o processamento de uploads e incapacidade de lidar com solicitações entre domínios<sup>36</sup>. Para resolver essas falhas, o rascunho “XMLHttpRequest Level 2” foi publicado em 2008, que adicionou uma série de novos recursos (GRIGORIK, 2013, P. 262-263).


Todos os novos recursos e recursos XHR2 são oferecidos através da mesma API XMLHttpRequest. Hoje, há apenas uma especificação XHR unificada, pois em 2011 a especificação “XMLHttpRequest Level 2” foi mesclada com a especificação original (GRIGORIK, 2013).

Uma lista de compatibilidade com os principais navegadores do mercado, pode ser conferida na Figura 4.

<sup>35</sup> Principal organização de padronização da World Wide Web.



<sup>36</sup> Nome que serve para localizar e identificar conjuntos de computadores na internet

Figura 4 – Navegadores compatíveis com XHR2

XMLHttpRequest advanced features 

Adds more functionality to XHR (aka AJAX) requests like file uploads, transfer progress information and the ability to send form data. Previously known as [XMLHttpRequest Level 2](http://www.w3.org/TR/2012/WD-XMLHttpRequest-20120117/), these features now appear simply in the XMLHttpRequest spec.

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Chrome for Android
9	13	51	57	9.1	43	9.3		4.4	
10	14	52	58	10	44	10.2		4.4.4	
11	15	53	59	10.1	45	10.3	all	56	59
	16	54	60	11	46	11			

  Partial Support

Global: 88.62% + 6.09% = 94.71%

Data from caniuse.com | Embed from caniuse.bitsofco.de

Fonte – Deveria (2014)

### 4.3 HTTP Polling

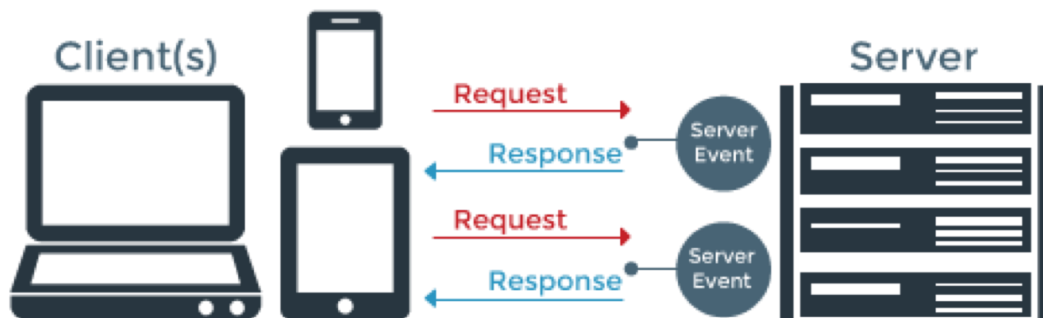
Segundo Grigorik (2013, P. 273), “o HTTP não fornece nenhuma maneira para o servidor iniciar uma nova conexão com o cliente”. Para isso, o cliente deve pesquisar por atualizações no servidor.

Para uma atualização constante de informações entre o cliente e servidor, XHR nos fornece uma maneira simples. O cliente faz uma solicitação regular ao servidor solicitando atualizações e o servidor responde, podendo a resposta não conter dado. Esta técnica é conhecida por *Polling HTTP* (ou *Ajax Polling*). Tende a ser um desperdício de recursos de rede e de servidor, porem é fácil de implementar (MCCARTHY; CRANE, 2009).

No entanto, de acordo com Pimentel e Nickerson (2012), *Polling HTTP* é considerada uma boa solução para a entrega de informações em tempo real se o intervalo de entrega da mensagem for conhecido e a taxa de transmissão de dados for constante. Porem, a cada consulta HTTP, repete-se informações de cabeçalho aumentando assim a latência.

Para reduzir a latência sem aumentar a carga do servidor, a solução é criar dois fluxos de comunicação entre o cliente e o servidor em uma técnica conhecida por *Long Polling HTTP*. O primeiro fluxo é usado para receber mensagens e o segundo fluxo é usado para enviar mensagens. Para receber mensagens, o cliente pesquisa o servidor com um pedido de mensagens; se não houver mensagens, o servidor não responde com uma resposta imediatamente. O servidor coloca o pedido de pesquisa em espera por um período de tempo específico ou até que uma mensagem seja gerada pelo servidor (Figura 5). A redução da latência é significativa ao converter o tempo inoperante em uma espera criada pelo servidor, enquanto o cliente aguarda o potencial de uma mensagem que está sendo gerada (GROSS, 2006).

Figura 5 – Long Polling HTTP



Fonte – Hanson (2014)

Ainda segundo Gross (2006), o principal problema desta abordagem é que milhares de threads<sup>37</sup> podem estar aguardando mensagens que possam ou não ser geradas, consumindo uma grande quantidade de Memória RAM<sup>38</sup>.

Nesta técnica, a conexão entre o cliente e servidor fica ociosa até expirar ou até que o servidor tenha dados para enviar. Cabe ao cliente a responsabilidade pela reconexão para aguardar a próxima atualização. Esta técnica é mais eficiente que *Ajax Polling*, quando os dados são enviados com pouca frequência (GUTWIN; LIPPOLD; GRAHAM, 2011).

#### 4.4 HTTP Streaming

Esta técnica se aproveita de um tipo de conteúdo HTTP chamado “*multipart*” que permite que um servidor web envie conteúdo para um navegador em várias partes (*chunked encoding*), projetada para o carregamento incremental de documentos muito grandes (GUTWIN; LIPPOLD; GRAHAM, 2011).

A implementação mais simples desta técnica é conhecida por *ForeverFrame*, onde tem-se um iframe dentro de uma página HTML recebendo atualizações constantes através do recebimento de partes de dados contendo uma mensagem completa em cada parte. Inicialmente esta técnica foi bastante condenada no Internet Explorer, pois este trata cada fragmento recebido como uma carga de página, disparando então, som de clique do carregamento de página. A solução veio com a utilização do objeto *htmlfile*<sup>39</sup> do ActiveX, tornando a técnica bastante popular (SOUDERS, 2009).

Esta técnica também pode ser explorada em uma requisição XHR (*XHR Streaming*), permitindo que um servidor web envie conteúdo para um navegador em várias partes. Essencialmente, o navegador é enganado para manter a conexão do soquete aberta, com o servidor

<sup>37</sup> Pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se auto dividir em duas ou mais tarefas.

<sup>38</sup> Tecnologia que permite o acesso aos arquivos armazenados no computador, responsável pela leitura dos conteúdos quando requeridos.

<sup>39</sup> Interface para obter informações, examinar e modificar documentos HTML.

enviando cada atualização como parte de um todo. O uso de uma única conexão de soquete também reduz o número de cabeçalhos HTTP que precisam ser enviados para pedidos e respostas, reduzindo assim, a latência (GUTWIN; LIPPOLD; GRAHAM, 2011).

Porem, se uma requisição *XHR Streaming* continuar aberta por muito tempo, o navegador pode sofrer com o acúmulo excessivo de memória RAM. No entanto, isto pode ser corrigido facilmente, simplesmente terminando a requisição e criando uma nova solicitação em um período de tempo definido ou após certa quantidade de mensagens recebidas (SOUDERS, 2009).

#### 4.5 Server-Sent Event (SSE)

Proposto desde 2009, define uma API para abrir uma conexão HTTP para receber notificações *push*<sup>40</sup> de um servidor na forma de eventos DOM, permitindo que os servidores enviem dados para páginas da Web por meio de HTTP ou usando protocolos dedicados de envio (HICKSON, 2015).

Normalizado pelo W3C em fevereiro de 2015, apresenta a interface *EventSource*; é projetada de modo que possa ser estendida para funcionar com outros esquemas de notificação *push*, como *Push SMS*<sup>41</sup> (HICKSON, 2015).

Sob o capô, SSE oferece uma implementação eficiente e cruzada do *XHR Streaming*; A entrega real das mensagens é feita através de uma única conexão HTTP de longa duração. No entanto, ao contrário de lidar com transmissão de XHR por conta própria, o navegador lida com todo o gerenciamento de conexão e análise de mensagens, permitindo que nossos aplicativos se concentrem na lógica de negócios! Em suma, a SSE torna o trabalho com dados em tempo real simples e eficiente (GRIGORIK, 2013, P. 279).

Suportado nativamente pela maioria dos navegadores modernos (vide Figura 6); foi uma adição antecipada à especificação HTML5. No entanto, a interface *EventSource* é simples, de modo que pode ser emulada através de uma biblioteca de JavaScript opcional (*Polyfill*<sup>42</sup>) para navegadores que não o suportam nativamente (GRIGORIK, 2013).

---

<sup>40</sup> Sistema de distribuição de conteúdo da Internet em que a informação sai de um servidor para um cliente, com base em uma série de parâmetros estabelecidos pelo cliente.

<sup>41</sup> Serviço utilizado para o envio de mensagens de texto curtos, através de telefones celulares.

<sup>42</sup> Biblioteca JavaScript que implementa o padrão HTML5, quer seja um padrão estabelecido para todos os navegadores ou não.

Figura 6 – Navegadores compatíveis com Server-Sent Event

Server-sent events [↗](#)

Method of continuously sending data from a server to the browser, rather than repeatedly requesting it (EventSource interface, used to fall under HTML5)

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Chrome for Android
6		48	54	7.1	39	8		4	
7		49	55	8	40	8.4		4.1	
8	12	50	56	9	41	9.2		4.3	
9	13	51	57	9.1	42	9.3		4.4	
10	14	52	58	10	43	10.2		4.4.4	
11	15	53	59	10.1	44	10.3	all	56	57
		54	60	11	45	11			

☒ ☒ ☒ Partial Support

Global: 88.79% + 0.04% = 88.83%

Data from [caniuse.com](#) | Embed from [caniuse.bitsofco.de](#)

Fonte – Deveria (2014)

## 4.6 WebSockets

O WebSocket oferece um canal de comunicação bidirecional que opera através de um único soquete HTTP, por onde cliente e servidor podem trocar mensagens simultaneamente. WebSocket também opera através do Protocolo de Camada de Soquetes Segura (SSL)<sup>43</sup> ou com Segurança da Camada de Transporte (TLS)<sup>44</sup> (LOMBARDI, 2015).

O WebSocket foi primeiro definido como “TCPConnection” na seção de Comunicação da especificação HTML5 por Ian Hickson<sup>45</sup>. A especificação evoluiu e mudou para WebSocket, se tornando uma especificação independente, para manter a discussão focada (LUBBERS et al., 2011). O protocolo define também duas nova URIs: “ws://” e “wss://” (para SSL ou TLS) tendo como portas padrão 80 (“ws://”) e 443 (“wss://”), as mesmas do protocolo HTTP (SAINT-ANDRE, 2011).

Permitindo a transmissão bidirecional e orientada para mensagens de textos e dados binários entre cliente e servidor, é a API mais próxima de uma conexão pura entre o navegador e o servidor, exceto pela abstração da complexidade por trás de uma API simples fornecida pelo navegador, que oferece uma série de serviços adicionais como a interoperabilidade com a infra-estrutura HTTP existente e a capacidade de negociar subprotocolos (GRIGORIK, 2013).

Nesse contexto, Grigorik (2013) ressaltar que o WebSocket é um protocolo totalmente

<sup>43</sup> Protocolo criptográfico que confere segurança de comunicação na Internet.

<sup>44</sup> Sucessor do SSL, é um protocolo de criptografia projetado para internet.

<sup>45</sup> Principal escritor da especificação HTML5

funcional e autônomo que pode ser usado fora do navegador. No entanto, seu principal uso é no transporte bidirecional para aplicativos baseados na Web. Com uma API orientada à mensagem, o remetente fornece dados em texto ou binários e o destinatário é notificado da entrega quando toda a mensagem estiver disponível.

Ainda segundo Grigorik (2013), para habilitar isso, o WebSocket usa um formato de enquadramento binário personalizado (Figura 7), dividindo a mensagem em um ou mais quadros (*Frames*) e transportando para o destinatário, que por sua vez, junta as partes e notifica o receptor uma vez que a mensagem inteira for recebida.

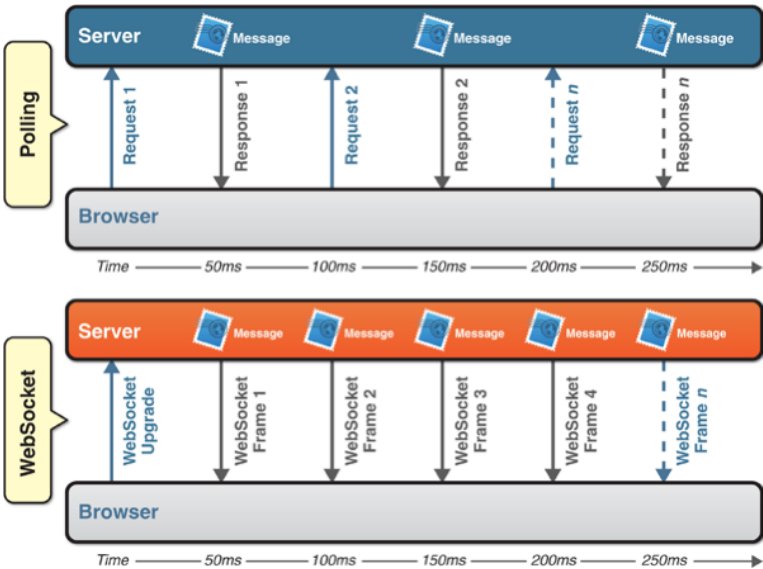
Figura 7 – Frames do WebSocket

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						

Fonte – (GRIGORIK, 2013)

O protocolo WebSocket combinado com a API de WebSocket (WSAPI) traz uma alternativa ao *HTTP Polling* para comunicação bi-direcional entre cliente e servidor, em uma única conexão TCP (SAINT-ANDRE, 2011). Com o WSAPI, realiza-se uma única requisição HTTP para abrir uma conexão WebSocket e reutiliza a mesma conexão no envio de mensagens do cliente para o servidor e do servidor para o cliente (Figura 8) (WANG; SALIM; MOSKOVITS, 2013).

Figura 8 – Comparativo entre os cenários de Polling HTTP e WebSocket



Fonte – Wang, Salim e Moskovits (2013)

Segundo Flanagan (2011), não é seguro dar acesso a soquete TCP de baixo nível para o cliente, no entanto, a API WebSockets define uma alternativa segura, permitindo que o cliente crie conexões bidirecionais por soquetes com servidores que suportem o protocolo WebSocket.

O WebSocket se beneficia por aproveitar da infra-estrutura do HTTP pois pode ser executado na porta 80 e 443, que frequentemente são as únicas portas abertas no servidor (GRIGORIK, 2013). Para estabelecer uma conexão WebSocket, um cliente abre uma conexão com um *handshake*, em uma requisição HTTP do tipo GET *”upgrade”*(Figura 9a). Se a conexão ocorrer sem problemas, o servidor deve responder com o status 101 (*”Switching protocols”*) (Figura 9b). Qualquer resposta diferente disso indica erro no *handshake* (SAINT-ANDRE, 2011).

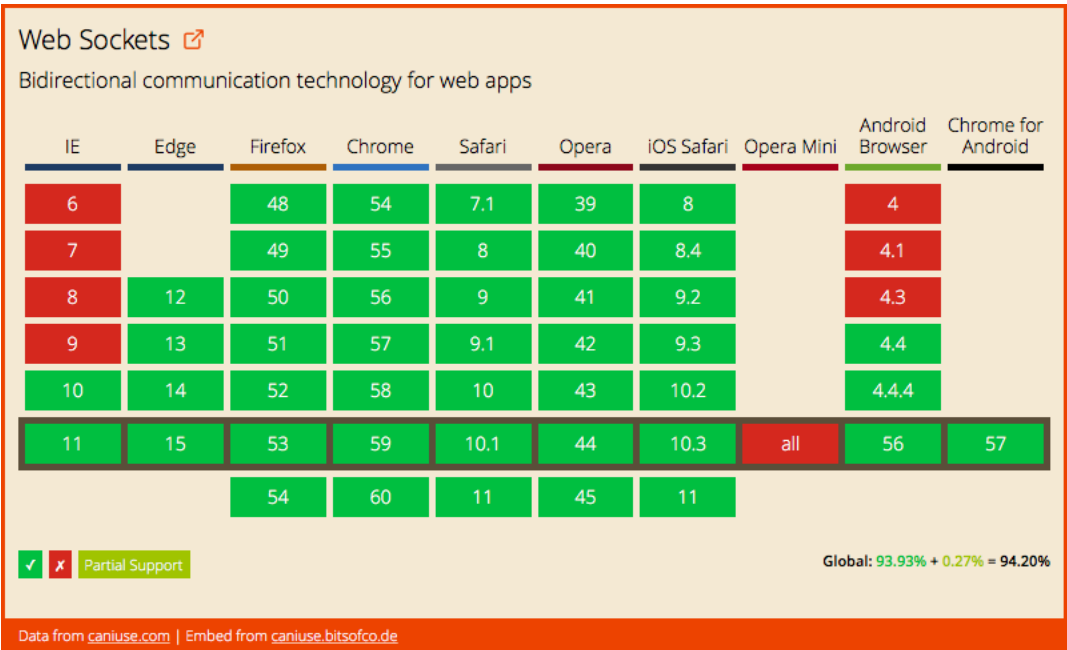
Figura 9 – Exemplo de *handshake* WebSocket

(a) Cabeçalho HTTP da requisição	(b) Cabeçalho HTTP da resposta
GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13	HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade Sec-WebSocket-Accept: s3pFLMBiTxq9kYGzZhZrBk+x0o= Sec-WebSocket-Protocol: chat

Fonte – Saint-Andre (2011)

Tal como a adoção do HTML5, o suporte ao WebSockets é relegado aos navegadores modernos (Figura 10). Isso significa que não há suporte para nenhum usuário com o Internet Explorer 9 ou inferior (LOMBARDI, 2015).

Figura 10 – Navegadores com suporte para WebSockets



Fonte – Deveria (2014)



## 5 CONCLUSÃO

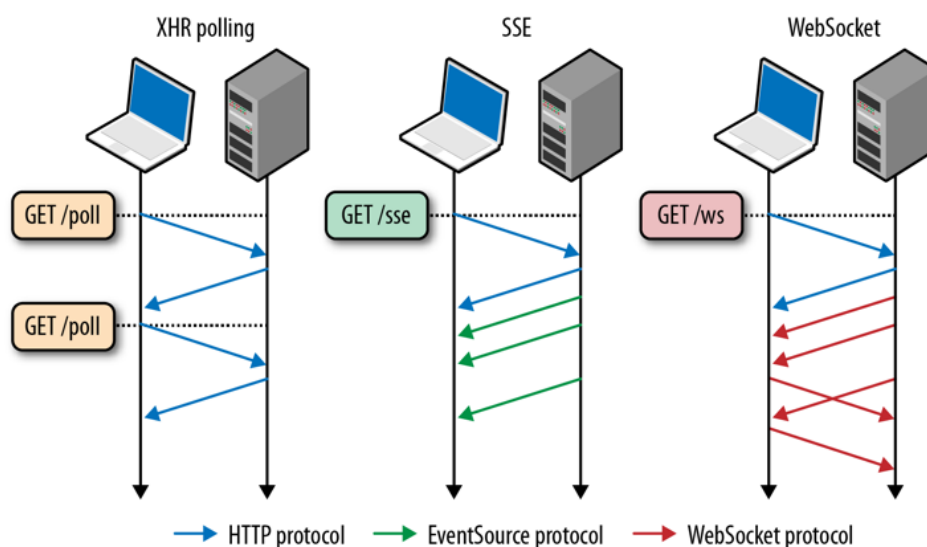
Os navegadores da Web estão cada vez mais se tornando uma plataforma para oferecer aplicações ricas que apresentam experiência de uso semelhante a aplicativos *stand alone*<sup>46</sup>. No entanto, a maneira tradicional em que esses aplicativos eram implantados, usando *plug-ins*, levou diversos desenvolvedores a buscar por outras soluções, o que resultou em uma melhoria gradual da comunicação entre uma aplicação Web e um servidor de dados.

Para aplicações em que a comunicação se baseia no modelo de requisição/resposta, o XHR se mostra bastante versátil, com alto desempenho. Porém, adiciona alguns *bytes*<sup>47</sup> extras no cabeçalho da requisição, o que pode trazer um aumento da latência na comunicação.

Resolvendo este problema, SSE traz uma API consistente com reconexão automática, notificações e recebimento de mensagens como eventos DOM, sem repetição de grandes cabeçalhos, baixa latência e sem sobrecarga de memória; se tornando uma ferramenta indispensável para aplicações que necessitam de notificações em tempo real. No entanto, não tem uma aceitação tão grande por parte dos navegadores, se comparado com WebSockets que é uma solução bem mais completa (vide Figuras 6 e 10), além de está limitada ao recebimento de mensagens de texto, sendo necessário utilizar do XHR para envio de mensagens.

O WebSocket é a única API que permite a comunicação bidirecional em um único soquete TCP (Figura 11), fornecendo entrega de textos e dados binários com baixa latência em ambas as direções, permitindo que o cliente e o servidor enviem mensagens ao mesmo tempo. Tem boa aceitação por navegadores modernos, estando disponível para uso em versões a partir do Internet Explorer 10 (Figura 10).

Figura 11 – Comparativo das principais tecnologias



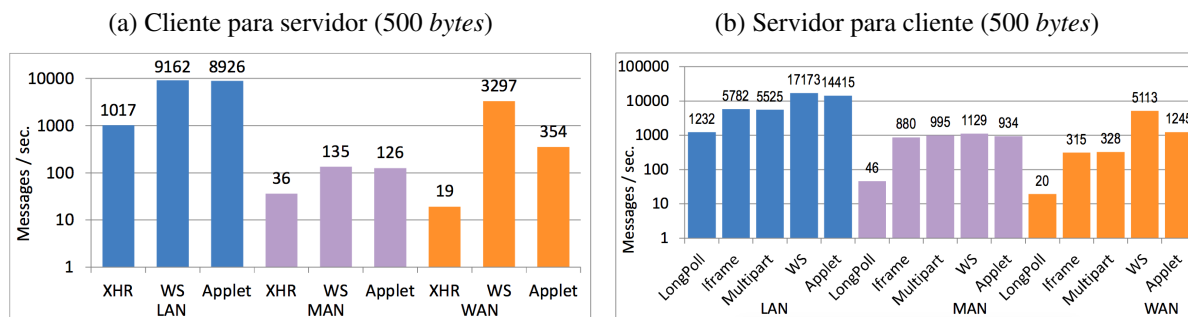
Fonte – Grigorik (2013)

<sup>46</sup> Programas completamente autossuficientes, que não necessitam de um software auxiliar.

<sup>47</sup> Unidade de informação digital equivalente a oito bits.

Testes de latência, realizados por Gutwin, Lippold e Graham (2011) mostraram ainda que WebSockets tem a maior taxa de transferência no envio e recebimento de mensagens através Web (Figura 12), se mostrando, na maioria dos casos, a melhor solução para comunicação em tempo real na Web.

Figura 12 – Testes comparativos de latência



Fonte – Saint-Andre (2011)

Vale ressaltar que WebSocket não é um substituto para XHR ou SSE, e para o melhor desempenho é fundamental que aproveitamos os pontos fortes de cada API. Para casos onde apenas se necessita do recebimento de mensagens do servidor para o cliente (sistemas de notificações, por exemplo), SSE se mostra uma ótima escolha.

## Referências

- AGHAEI, S. Evolution of the World Wide Web : From Web 1.0 to Web 4.0. **International journal of Web & Semantic Technology**, v. 3, n. 1, 2012. ISSN 09762280. Disponível em: <http://www.airccse.org/journal/ijwest/papers/3112ijwest01.pdf>.
- ASLESON, R.; SCHUTTA, N. T. **Foundations of Ajax**. [S.l.]: Springer, 2006. ISBN 1590595823.
- BERNERS-LEE, T.; CONNOLLY, D. Hypertext Markup Language (HTML). **Internet Engineering Task Force (IETF) Draft**, 1993. Disponível em: <http://tools.ietf.org/html/draft-ietf-iiir-html-00>.
- CONNOLLY, D. W.; MASINTER, L. The 'text/html' Media Type. **Request for Comments**, v. 2854, n. June, 2000. Disponível em: <https://tools.ietf.org/html/rfc2854>.
- CROCKFORD, D. **JavaScript: The Good Parts: The Good Parts**. [S.l.]: "O'Reilly Media, Inc.", 2008. v. 6.
- DEVERIA, A. **Can I Use**: Browser support tables for modern web technologies. 2014. Disponível em: <http://caniuse.com>.
- ECMASCRIPT, E.; ASSOCIATION, E. C. M. et al. **Ecmascript language specification**. 2016.
- FIELDING, R.; IRVINE, U. C.; GETTYS, J. Hypertext Transfer Protocol - HTTP/1.1. **Ietf**, 1999. ISSN 2070-1721. Disponível em: <https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>.
- FLANAGAN, D. **JavaScript: The definitive guide: Activate your web pages**. [S.l.]: "O'Reilly Media, Inc.", 2011.
- GARRETT, J. J. Ajax: A New Approach to Web Applications. 2005. Disponível em: <http://www.adaptivepath.com/publications/essays/archives/000385print.php>.
- GIL, A. C. Como elaborar projetos de pesquisa. **São Paulo**, v. 4, 2002.
- GOETHALS, K.; AGUIAR, A.; ALMEIDA, E. História da internet. **Faculdade de Engenharia da Universidade do Porto, Mestrado em Gestão da Informação**, 2000.
- GOODMAN, D. **JavaScript bible**. [S.l.]: John Wiley & Sons, 2007.
- GRIGORIK, I. **High Performance Browser Networking: What every web developer should know about networking and web performance**. [S.l.]: "O'Reilly Media, Inc.", 2013.
- GROSS, C. Introduction to ajax. **Ajax Patterns and Best Practices**, Springer, p. 1–18, 2006.
- GUTWIN, C. A.; LIPPOLD, M.; GRAHAM, T. Real-time groupware in the browser: testing the performance of web-based networking. In: ACM. **Proceedings of the ACM 2011 conference on Computer supported cooperative work**. [S.l.], 2011. p. 167–176.
- HANSON, J. **What is HTTP Long Polling?** 2014. Disponível em: <https://www.pubnub.com/blog/2014-12-01-http-long-polling/>.

HAVERBEKE, M. **Eloquent javascript: A modern introduction to programming**. [S.l.]: No Starch Press, 2014.

HICKSON, I. **Server-Sent Events. Recommendation**. [S.l.]: W3C, 2015.

KEITH, J.; SAMBELLS, J. **DOM scripting: web design with JavaScript and the Document Object Model**. [S.l.]: Apress, 2010.

LEINER, B. et al. **Brief History of the Internet — Internet Society**. 2009. Disponível em: <http://www.isoc.org/internet/history/>.

LOMBARDI, A. **WebSocket: Lightweight Client-Server Communications**. [S.l.]: "O'Reilly Media, Inc.", 2015.

LORETO, S. et al. **RFC 6202 - Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP**. 2011. Disponível em: <https://tools.ietf.org/pdf/rfc6202.pdf>.

LUBBERS, P. et al. **Pro HTML5 programming**. [S.l.]: Springer, 2011.

MARCONI, M.; LAKATOS, E. **Fundamentos de metodologia científica**. [S.l.]: 5. ed.-São Paulo: Atlas, 2003. ISSN 9788522457588. ISBN 8522433976.

MCCARTHY, D.; CRANE, C. **Comet and reverse ajax: The next-generation ajax 2.0**. [S.l.]: Apress, 2009.

MILLS, C. (Ed.). **Sending form data**. 2017. Disponível em: [https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending\\_and\\_retrieving\\_form\\_data](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data).

NEER, K.; III, W. F. L. History of javascript. **History**, 2013.

OFFUTT, J. Quality attributes of web software applications. **IEEE software**, IEEE, v. 19, n. 2, 2002.

PIMENTEL, V.; NICKERSON, B. G. Communicating and displaying real-time data with WebSocket. **IEEE Internet Computing**, v. 16, n. 4, 2012. ISSN 10897801.

POWELL, T. **Ajax: the complete reference**. [S.l.]: McGraw-Hill, Inc., 2008.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico-2ª Edição**. [S.l.]: Editora Feevale, 2013.

RESIG, J. **Pro JavaScript Techniques**. [S.l.]: Apress, 2007.

SAINT-ANDRE, P. RFC Standard 6455– The WebSocket Protocol. **RFC 6455 (Proposed Standard)**, 2011. ISSN 1098-6596. Disponível em: <http://datatracker.ietf.org/doc/rfc6455/>.

SOFTWSP. **Download Netscape Navigator**. 2016. Disponível em: <http://linux.softwsp.com/browsers/netscape-navigator/>.

SOUDERS, S. **Even faster web sites: performance best practices for web developers**. [S.l.]: "O'Reilly Media, Inc.", 2009.

WANG, V.; SALIM, F.; MOSKOVITS, P. **The Definitive Guide to HTML5 WebSocket**. [S.l.]: Springer, 2013. ISBN 978-1-4302-4740-1.

ZAMMETTI, F. W. A brief history of javascript. **Practical JavaScript™, DOM Scripting, and Ajax Projects**, Springer, p. 3–27, 2007.

## Lista de ilustrações

Figura 1 – Modelo de requisição HTTP . . . . .	3
Figura 2 – Navegador Netscape 2 . . . . .	7
Figura 3 – Comparação entre o modelo clássico e o modelo Ajax . . . . .	10
Figura 4 – Navegadores compatíveis com XHR2 . . . . .	11
Figura 5 – Long Polling HTTP . . . . .	12
Figura 6 – Navegadores compatíveis com Server-Sent Event . . . . .	14
Figura 7 – Frames do WebSocket . . . . .	15
Figura 8 – Comparativo entre os cenários de Polling HTTP e WebSocket . . . . .	15
Figura 9 – Exemplo de <i>handshake</i> WebSocket . . . . .	16
Figura 10 – Navegadores com suporte para WebSockets . . . . .	16
Figura 11 – Comparativo das principais tecnologias . . . . .	17
Figura 12 – Testes comparativos de latência . . . . .	18